机器学习

（编程练习）

安德鲁吴斯坦福大学

内容

编程练习1：线性回归

机器学习

介绍

在这个练习中，您将实现线性回归，并能看到它的数据。开始对这种编程练习之前，我们强烈建议观看视频讲座，并完成对相关主题的复习题。

要开始使用的练习中，您将需要的内容下载启动代码，并解压到你想要完成练习的目录。如果需要，可使用 光盘 在八度命令来开始这个练习前切换到该目录。

您也可以连接供课程网站上的“八音安装”页面上安装第二八音说明。

文件包含在此锻炼

ex1.m - 八音脚本，将帮助您逐步完成练习
EX1 multi.m - 对于这次演习的后面部分八度脚本
ex1data1.txt - 数据集用于与一个变量线性回归
ex1data2.txt - 数据集用于与多个变量线性回归
submit.m - 提交脚本发送您的解决方案，我们的服务器
warmUpExercise.m [?] - 在八度简单的例子功能
plotData.m [?] - 功能来显示数据集
computeCost.m [?] - 函数来计算线性回归的成本
gradientDescent.m [?] - 函数运行梯度下降
[†] computeCostMulti.m - 多变量成本函数
[†] gradientDescentMulti.m - 梯度下降的多个变量
[†] featureNormalize.m - 功能正常化的功能
[†] normalEqn.m - 函数来计算法方程

? 表明科幻LES，你将需要完成
† 表明额外的信用演习

在整个练习中，您将使用脚本 ex1.m 和 EX1 multi.m。 _
这些脚本建立数据集的问题，并作出你会写函数的调用。你并不需要修改其中任何。您只需要修改的功能在其他科幻LES，按照此任务的指示。

对于这种编程练习中，您只需要完成练习的第一个部分来实现线性回归一个变量。演习中，你可以完成额外信贷的第二部分，涵盖了多变量线性回归。

从哪里获得帮助

本课程的练习用八度，[1] 一个高级语言非常适合于数字计算。如果您没有安装倍频做什么，请参阅安装 instructons举行的“八度安装”页面的课程网站上。

在八度的命令行，输入 救命 随后是一个内置函数的函数名称显示的文档。例如， 帮助情节 将弹出帮助信息绘制。用于八度音功能的进一步文档可在找到 八度文档页面 。我们还大力鼓励使用在线 Q&A 论坛 讨论与其他学生练习。但是，不看别人写的任何源代码或与他人分享你的源代码。

1个简单倍频功能

的第一个部分 ex1.m 为您提供了八度语法和作业提交过程练习。在网络连接文件 warmUpExercise.m，你会科幻ND的八度音功能的轮廓。修改它以在下面的代码返回通过网络连接灌装的5×5单位矩阵：

```
A =眼（5）;
```

当你连接nished，运行 ex1.m（ 假设你是在正确的目录，键入“ EX1 ”在八度提示），你应该会看到类似以下的输出：

[1] Octave是一个免费的替代MATLAB。对于编程练习，您可以自由使用任何八度或MATLAB。

```
ANS =

对角矩阵

     1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
     0 1 0 0 0 0 0 1
```

现在 ex1.m 将暂停，直到你按任意键，然后将运行代码的分配的一个部分。如果你想退出，打字 CTRL-C 将在其运行到一半时停止该程序。

1.1提交解决方案

在完成工作的一部分后，您可以通过键入提交您的解决方案，为分级 提交 在八度命令行。提交脚本会 提示您输入您的用户名和密码，并询问你网络莱要提交哪些。您可以从该网站的"编程练习"页面提交密码。

现在，您应该提交的热身运动。

你被允许多次提交您的解决方案，我们将只需要得分最高的考虑。为了防止速效音响再猜测，该系统强制最小的提交之间5分钟。

2线性回归用一个可变

在这个练习中的这一部分，您将实现线性回归一个变量来预测亲科幻TS用于食品的卡车。假设你是吴宗宪的CEO，并考虑双FF erent城市打开了新的出路。这家连锁店已经在多个城市的卡车，你有从城市亲音响TS和人口数据。

你想用这个数据来帮助您选择扩大到下一哪个城市。

该网络文件 ex1data1.txt 包含我们的线性回归问题的数据集。的第一列是一个城市的人口，第二列是在这个城市食品卡车的亲科幻吨。亲科幻吨负值表示的损失。

该 ex1.m 剧本已经被设置为加载此数据为您服务。

2.1绘图数据

在开始任何任务之前，常常是有用的可视化，理解数据。对于这个数据集，你可以使用一个散点图以可视化的数据，因为它只有两个属性绘制（PRO网络连接T和人口）。（你会在现实生活中遇到的诸多难题是多维，不能在2 d绘制曲线）。

在 ex1.m， 数据集从数据网络连接到文件中的变量加载 *X* 和 *Y* :

```
数据=负载（ 'ex1data1.txt' ）;                         %读取逗号分隔数据
X =数据（ : ，1）; Y =数据（ : ，2）; M =长度（Y）;
                                                       的训练实例数%
```

接下来，脚本调用 plotData 函数来创建数据的散点图。你的任务是完成 plotData.m 绘制情节; 修改下面的代码的网络文件和FI LL：

```
积（X，Y， 'RX' ， 'MarkerSize' ，10）;                      %绘制数据
ylabel（ "利润$ 10,000s" ）;                                 %设定Y - 轴标签
xlabel（ "城市的人口在10,000s" ）; %设置X - 轴标签
```

现在，当你继续运行 ex1.m， 我们的最终结果应该像图1中，用相同的红色"X"标记和轴标签。

要了解更多关于绘图命令，你可以键入 帮助情节 在八度命令提示符或用于绘图文档在网上搜索。（要更改标记为红色的"X"，我们使用了选项"RX"的情节命令在一起，即， 情节（.. [这里你的选择]，... ，'RX'）;）

2.2梯度下降

在这一部分，你将科幻t时的线性回归参数 θ 我们的数据集采用梯度下降。

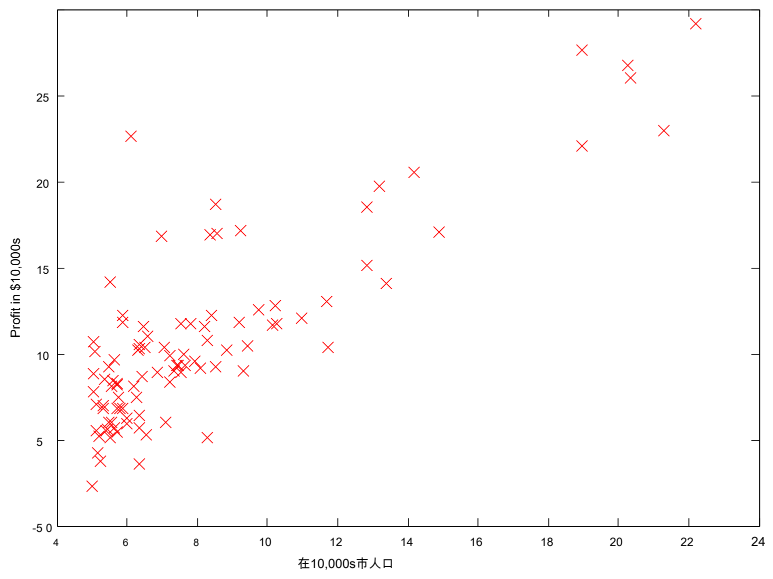


图1：训练数据的散点图

2.2.1更新方程

线性回归的目标是最小化的成本函数

$$\hat{J}(\theta) = 1 \frac{\sum^{米}}{2\,米}_{I=1} (H_{\theta}(X(一世)) - Y(一世))_2$$

其中所述假设 $H_{\theta}(X)$ 由线性模型给出

$$H_{\theta}(X) = \theta_{T} X = \theta_{0} + \theta_{1} X_{1}$$

回想一下，你的模型的参数是 θ_{J}值。这些都是你会调整将成本降至最低值 $\hat{J}(\theta)$。 **要做到这一点的**方法之一是使用批处理梯度下降算法。在分批梯度下降，每次迭代执行更新

$$\theta_{Y} := \theta_{J} - \alpha 1 \frac{\sum^{米}}{米}_{I=1} (H_{\theta}(X(一世)) - Y(一世))_{J} X(一世)$$ **(同时更新 θ_{J}对全部 j) 的。**

随着梯度下降，你的参数每一步 θ_{J}前来接近

最优值，将实现最低成本 *Ĵ (θ)。*

实现注意： 我们每个示例存储为在该行 X
矩阵八度。考虑到截距项（ θ_0 ）, 我们添加一个额外的第一列到 X 并将其设置为全1。这使我们能够治疗 θ_0 作为只是另一种“功能”。

2.2.2实施

在 ex1.m， 我们已经建立了线性回归的数据。在下面的行，我们添加了另一个维度我们的数据，以适应

θ_0 截距项。我们还初始化的初始参数为0，学习速率 α 0.01。

```
X = [者（M，1），数据（：，1）]; %加入那些与X的列
THETA =零（2，1）; %初始化参数拟合

迭代= 1500; 阿尔法= 0.01;
```

2.2.3计算成本 *Ĵ (θ)*

当你进行梯度下降学习最小化成本函数 *Ĵ (θ)，*
它有助于通过计算成本来监视收敛。在本节中，您将实现一个函数来计算 *Ĵ (θ)* 这样你就可以检查你的梯度下降实现的融合。

你的下一个任务是完成代码的网络文件 computeCost.m， 这是计算的函数 *Ĵ (θ)。* 当你这样做时，请记住变量 *X* 和 *ÿ* 不是标量的值，但其矩阵行表示从训练集中的例子。

一旦你完成的功能，下一步在 ex1.m 会跑
computeCost 一旦使用 *θ* 初始化为零，你会看到显示在屏幕上的成本。

你应该会看到的成本 32.07。

您现在应该提交“计算成本”线性回归的一个变量。

2.2.4梯度下降

接下来，您将实现在网络文件梯度下降 gradientDescent.m。
循环结构已经写了你，你只需要提供更新 θ 每个迭代内。

至于你的程序，确保你了解你们什么优化，什么正在更新。请记住，成本 $\hat{J}(\theta)$ 由矢量参数 θ，不 X 和 年。也就是说，我们尽量减少的值 $\hat{J}(\theta)$

通过改变矢量的值 θ，不改变 X 要么 年。参考公式在这个讲义和视频讲座，如果你是不确定的。

验证梯度下降是否正常工作的一个好方法是看价值 $\hat{J}(\theta)$ 并检查它是否与每个步骤递减。对于启动代码 gradientDescent.m 电话 computeCost 在每次迭代并打印成本。假设你已经实现了梯度下降和

computeCost 正确，你的价值 $\hat{J}(\theta)$ 不应该增加，并应通过算法结束收敛到一个稳定值。

之后你是科幻nished， ex1.m 将使用您的网络连接最终参数，绘制直线科幻吨。结果应该类似于图 2：

为您的网络连接最终值 θ 也将被用来制作上亲科幻TS预测在35000和70000人的地方。需要注意的方式，下面几行
ex1.m 采用矩阵乘法，而不是明确的总和或循环，来计算预测。这是在八度代码矢量的一个例子。

现在，您应该提交梯度下降线性回归的一个变量。

```
predict1 = [1，3.5] *峰; predict2 = [1,7] *峰;
```

2.3调试

这里有一些事情要记住当你实现梯度下降：

?? 八度数组索引从一个，而不是从零开始。如果你存储 θ_0 和 θ_1 在一个名为向量 θ表示 该值将是 THETA（1） 和 THETA（2）。

?? 如果你在运行时看到了许多错误，检查你的矩阵运算，以确保你加法和乘法兼容维度的矩阵。打印的变量与尺寸 尺寸

命令会帮你调试。

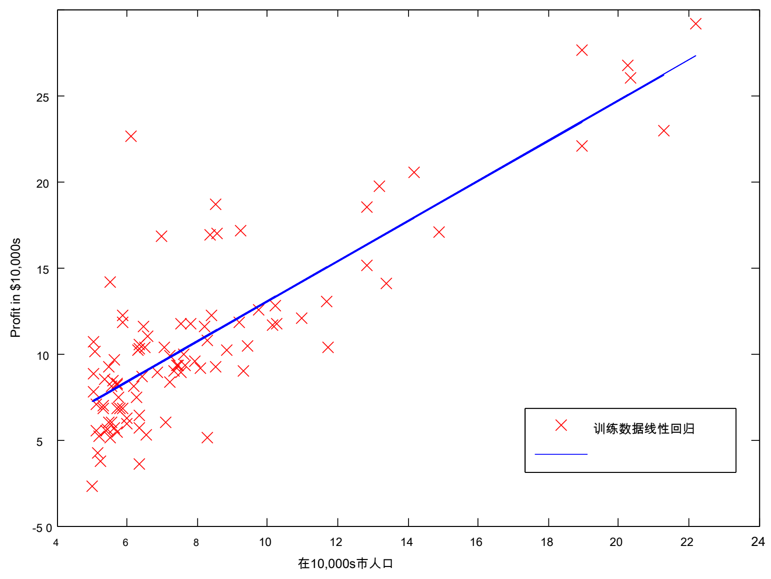


图2：用线性回归音响T个训练数据

?? 默认情况下，倍频解释数学运算符是矩阵运营商。这是大小不兼容错误的常见原因。如果你不想矩阵乘法，你需要添加“点”表示法指定这八度。例如， A * B 确实的矩阵相乘，而 A. * B 确实逐元素乘法。

2.4可视化 $\hat{J}(\theta)$

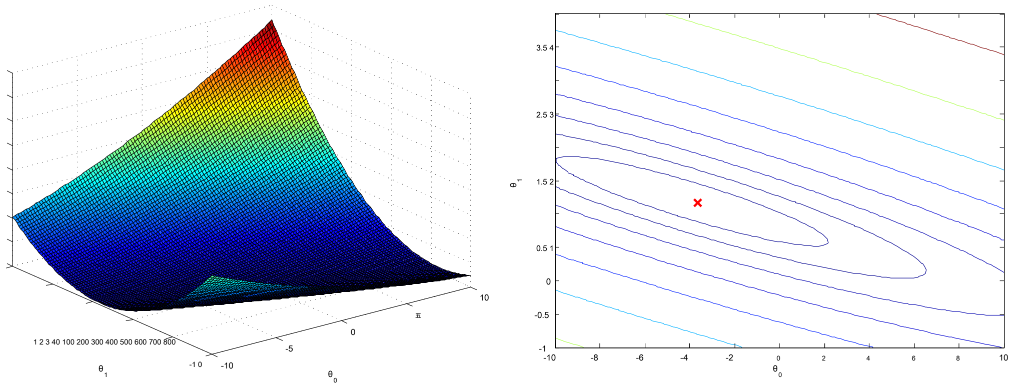
要了解成本函数 $\hat{J}(\theta)$ 好，你现在将绘制在成本的2维网格 θ_0 和 θ_1 值。您不需要为这部分代码什么新的东西，但你应该了解你已经编写的代码是创建这些图像。

在接下来的步骤 ex1.m， 有代码设置为计算 $\hat{J}(\theta)$ 过使用值的格 computeCost 你写的功能。

```
%初始化Ĵ瓦尔斯为0的矩阵
Ĵ瓦尔斯=零（长度（theta0瓦尔斯），长度（theta1瓦尔斯））;

%填写Ĵ瓦尔斯
对于 I = 1：长度（theta0瓦尔斯）
      对于 J = 1：长度（theta1瓦尔斯）
          T = [theta0瓦尔斯（i）; theta1瓦尔斯（J）]; Ĵ瓦尔斯（I，J）= comput
          eCost（X，Y，T）;
      年底结
束
```

这些线被执行之后，你将有一个2 d阵列 $\hat{J}(\theta)$ 值。剧本 ex1.m 然后将使用这些值以产生的表面和等高线图 $\hat{J}(\theta)$ 使用 冲浪 和 轮廓 命令。该地块应该看起来像图3：



（a）表面　　　　（b）中等值线，显示出最小

图3：成本函数 $\hat{J}(\theta)$

这些图的目的是向您展示如何 $\hat{J}(\theta)$ 随着变化而变化 θ_0 和 θ_1。成本函数 $\hat{J}(\theta)$ 呈碗状，并具有全球性的最小值。（这是比在所述3D表面图更容易在等值线图看）。这个最低是最佳点 θ_0 和 θ_1，和梯度下降每一步移动靠近该点。

加分练习（可选）

如果您已成功完成上述材料，恭喜！现在你明白了线性回归，应该能够开始使用它自己的数据集。

对于这种编程练习的其余部分，我们已包括以下可选的额外信贷演习。这些练习将帮助你获得的材料有更深的了解，如果你能做到这一点，我们鼓励你去完成它们。

3线性回归多变量

在这一部分中，您将实现多变量线性回归预测房屋的价格。假设你卖你的房子，你想知道一个良好的市场价格会是什么。要做到这一点的方法之一是第一个收集出售房屋最近的信息，使住房价格的典范。

该网络文件 ex1data2.txt 包含在俄勒冈州波特兰市住房价格的训练集。的第一列是房子的大小（平方英尺），第2栏是卧室的数目，而第三列是房子的价格。

该 EX1 multi.m 剧本已经成立，以帮助您完成这个练习步骤。

3.1特征正规化

该 EX1 multi.m 脚本将通过加载并从该数据集显示某些值开始。通过观察值，注意房子大小卧室的约1 000倍。当量级的特点迪FF呃，第一个执行缩放功能可以使梯度下降收敛更加迅速。

在这里你的任务是完成代码 featureNormalize.m 至

?? 减去从数据集中的每个要素的平均值。

?? 减去平均值，另外量表（分）通过它们各自的特征值之后“标准偏差”。

的标准偏差是测量的一种方式存在的特定特征的值的范围多少变化（最数据点将内位于 ± 平均值2个标准差）; 这是采取的值的范围（最大值 - 最小值）的替代品。在八度，你可以使用“ STD ”函数来计算标准偏差。例如，内 featureNormalize.m，

数量 X（：，1） 包含的所有值 X_1（ 房子的大小）在训练集，所以 STD（X（：，1）） 计算房子的大小的标准偏差。在时间 featureNormalize.m 被调用时，1的对应的额外的列 $X_0 = 1$还没有被添加到 X （ 看到 EX1 multi.m 了解详细信息）。

–

你将要为所有的功能做到这一点，你的代码应与所有尺寸（任意数量的功能/例）的数据集工作。需要注意的是矩阵中的每一列 X 对应于一个功能。

现在，您应该提交功能正常化。

实现注意： 当标准化的特征，它存储用于标准化的值很重要 - *平均值* 和 *标准偏差* 用于计算。从模型的学习参数后，我们经常要预测我们以前没有见过的房子的价格。鉴于新 X 值（客厅和卧室的数量），我们必须第一个规范 X 使用我们以前训练组计算的平均值和标准偏差。

3.2梯度下降

此前，你实施了单变量回归问题的梯度下降。唯一的双FF erence现在是，还有一个功能在矩阵 X。 该假说功能和批量梯度下降更新规则保持不变。

你应该完成的代码 computeCostMulti.m 和 gradientDescentMulti.m
以实现用于多变量线性回归的成本函数和梯度下降。如果你在前面的部分（单变量）的代码已经支持多个变量，你可以在这里使用它。

确保您的代码支持任意数量的特性，并充分量化。您可以使用 ' 尺寸（X，2） “到FI ND有多少功能是存在于数据集。

现在，您应该提交的计算成本和梯度下降的多变量线性回归。

实现注意： 在多变量情况下，成本函数还可以写成以下形式矢量：

$$\hat{J}(\theta)=1\ \frac{}{2\text{米}(X\theta-\frown Y)_{T(}X\theta-\frown Y)}$$

哪里

$$X=\begin{bmatrix} -(X(_{1)})\bar{T}- \\ -(X(_{2)})\bar{T}- \\ \vdots \\ -(X(\text{公吨}- \end{bmatrix} \qquad \frown Y=\begin{bmatrix} Y(_{1)} \\ Y(_{2)} \\ \vdots \\ Y(_{M)} \end{bmatrix}\text{。}$$

该矢量版本是当你与数值计算工具，如倍频工作êFFI cient。如果你是通过矩阵操作方面的专家，你能证明自己这两种形式是等价的。

3.2.1可选（未分级）运动：选择学习速率

在这部分练习中，你会得到尝试的数据集和网络找到一个学习率迅速收敛迪FF erent学习速率。您可以通过修改改变学习率 EX1 multi.m 并改变设置学习速率代码的一部分。

_

下一阶段 EX1 multi.m 会打电话给你 gradientDescent.m 功能和用于在所选择的学习率约50次迭代运行梯度下降。该函数还应该返回的历史 $\hat{J}(\theta)$ 在一个矢量值

J. 最后一次迭代之后， EX1 multi.m 脚本绘制了 Ĵ 针对迭代的次数的值。

如果一个良好的范围内挑选了学习率，你的情节类似于图4.如果你的图形看起来非常迪FF erent，特别是如果你的价值 $\hat{J}(\theta)$
增加甚至炸毁，调整你的学习速度，然后再试一次。我们建议您尝试学习率的值 α 在对数尺度，在上一次的值的约3倍的乘法步骤（即，0.3%，0.1%，0.03，0.01等）。您可能还需要调整您正在运行的迭代次数是否会帮助你看到在曲线的整体趋势。

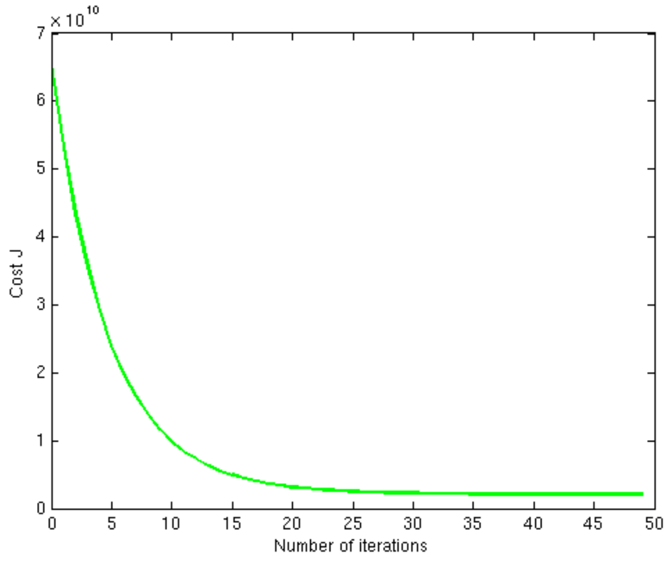


图4：用适当的学习率的梯度下降收敛

实现注意： 如果你的学习速度过大， *Ĵ(θ)* 能发散和“炸毁”，致使这对计算机的计算值过大。在这些情况下，八度往往会返回NaN的。为NaN表示“不是一个数字”，但通常被涉及理解过程科幻定义操作引起 -∞ 和+ ∞。

八度提示： 来比较迪FF erent学习学习率FF ECT收敛，它有助于绘制 Ĵ 对同一网络古尔几个学习速率。在八度，这可以通过多次与执行梯度下降来完成 '坚持，稍等' 地块之间的命令。具体而言，如果你已经尝试阿尔法三个迪FF erent值（你应该尝试比这更多的值）并存储在成本 J1，J2 和 J3， 您可以使用下面的命令来绘制它们在同一网络古尔：

情节（1:50，J1（1:50）， 'B'）; 坚持，稍等
;
情节（1:50，J2（1:50）， 'R'）; 情节（1:
50，J3（1:50）， 'K'）;

该网络最终参数 'B'， 'R'， 和 数k 指定地块迪FF erent颜色。

13

注意在收敛曲线，学习率的变化的变化。用一个小的学习速度，你应该科幻ND的是梯度下降需要很长的时间才能收敛到最优值。相反，具有较大的学习率，梯度下降可能不收敛，甚至可能偏离！

使用您找到最好的学习速度，运行 EX1 multi.m 脚本运行梯度下降直至收敛到Fi十二次网络的最终值 *θ*。 接下来，使用这个值 *θ* 预测房子价格1650平方英尺，3间卧室。您将使用值后，以检查你的正规方程的实施。不要忘了，当你做出这个预测一定要规范你的功能！

你不需要提交任何解决方案，这些可选的（未分级）练习。

3.3一般方程

在讲座的视频，你了解到的封闭形式的解决方案，以线性回归

$$\theta = (X^{\check{T}} X)^{-1} X^{T\sim} \text{年。}$$

使用这个公式不需要任何特征缩放，你会在一个计算得到一个确切的解决方案：有没有"循环，直至收敛"像梯度下降。

完成代码 normalEqn.m 使用上面的公式来计算 *θ*。 请记住，当你没有需要扩展功能，我们还需要1周的一列添加到X矩阵有一个截距项（ θ_0）。

中的代码 ex1.m 将1周的的列添加到X你。

现在，您应该提交正规方程函数。

可选（未分级）练习： **现在，一旦你发现 *θ* 使用这种方法，用它使一个1650平方英尺的房子的价**格预测有3间卧室。你应该科幻ND，让同样的预测价格为您获得使用该模型科幻吨，梯度下降（第3.2.1节）的值。

提交和分级

在完成任务的各部分后，请务必使用 提交
功能系统，您的解决方案提交到我们的服务器。以下是这项工作的每个部分是如何拿下了故障。

部分	提交文件	点
热身运动	warmUpExercise.m	10点
对一个变量计算成本	computeCost.m	40分
梯度下降的一个变量	gradientDescent.m 50分	
总积分		100点

加分练习（可选）

功能正常化	featureNormalize.m	10点
多个变量计算成本	computeCostMulti.m	15分
梯度下降的多个变量	gradientDescentMulti.m 15分	
普通方程	normalEqn.m	10点

你被允许多次提交您的解决方案，我们将只需要得分最高的考虑。为了防止速效音响再猜测，该系统强制最小的提交之间5分钟。

编程练习2：Logistic回归

机器学习

介绍

在这个练习中，您将实现回归，并应用两个双FF erent数据集。开始的编程练习之前，我们强烈建议观看视频讲座，并完成对相关主题的复习题。

要开始使用的练习中，您将需要的内容下载启动代码，并解压到你想要完成练习的目录。如果需要，可使用 光盘 在八度命令来开始这个练习前切换到该目录。

您也可以连接供课程网站上的“八音安装”页面上安装第二八音说明。

文件包含在此锻炼

ex2.m - 八音脚本，将帮助您逐步完成练习
EX2 reg.m - 对于这次演习的后面部分八度脚本
ex2data1.txt - 对于第一个半运动的训练集
ex2data2.txt - 培训设置了锻炼下半年
submit.m - 提交脚本发送您的解决方案，我们的服务器
mapFeature.m - 函数生成多项式功能
plotDecisionBounday.m - 功能绘制CLASSI科幻读者的决策边界
plotData.m [？] - 功能绘制2D CLASSI网络阳离子数据
sigmoid.m [？] - 双曲线函数
costFunction.m [？] - Logistic回归成本函数
predict.m [？] - Logistic回归预测功能
costFunctionReg.m [？] - 正则Logistic回归成本

? 表明科幻LES，你将需要完成

在整个练习中，您将使用脚本 ex2.m 和 EX2 reg.m. _

这些脚本建立数据集的问题，并作出你会写函数的调用。你并不需要修改其中任何。您只需要修改的功能在其他科幻LES，按照此任务的指示。

从哪里获得帮助

本课程的练习用八度，[1] 一个高级语言非常适合于数字计算。如果您没有安装倍频做什么，请参阅安装 instructons举行的“八度安装”页面的课程网站上。

在八度的命令行，输入 救命 随后是一个内置函数的函数名称显示的文档。例如， 帮助情节 将弹出帮助信息绘制。用于八度音功能的进一步文档可在找到 八度文档页面 。我们还大力鼓励使用在线 Q＆A 论坛 讨论与其他学生练习。但是，不看别人写的任何源代码或与他人分享你的源代码。

如果遇到网络错误使用 提交 脚本，你也可以使用在线表单提交您的解决方案。要使用此 *替代* 提交界面，运行 submitWeb 脚本生成提交文件连接（例如，

提交EX2 part1.txt）。 然后，您可以通过编程练习页面网页提交表单提交这个文件（去编程练习页面，然后选择您要提交的锻炼）。如果您有没有问题，使用通过标准提交系统提交 提交 脚本，你做 *不* 需要使用这种替代提交界面。

---

1 Logistic回归

在这部分练习中，你将建立一个逻辑回归模型来预测一个学生是否被录取到一所大学。

假设你是一个大学院系的管理员，您要确定录取的每个申请人的机会根据自己的

[1] Octave是一个免费的替代MATLAB。对于编程练习，您可以自由使用任何八度或MATLAB。

结果在两次考试。你必须从以前的申请人可以为Logistic回归训练集使用的历史数据。对于每一个训练例如，您有申请人的两项考试和录取分数决定。

你的任务是建立一个基于估计来自这两个考试分数录取的申请人的概率CLASSI网络阳离子模式。本大纲和框架代码 ex2.m 将指导您完成练习。

1.1可视化数据

开始实施任何学习算法之前，这是一件好事，如果可能的可视化数据。在的第一个部分 ex2.m， 该代码将加载的数据，并通过调用函数显示它在2维图表 plotData。

现在，您将完成代码 plotData 以便它显示一个音响古尔像图 1 ，其中所述轴是两个考试的分数，以及正和负的例子被示出具有二FF erent标记。

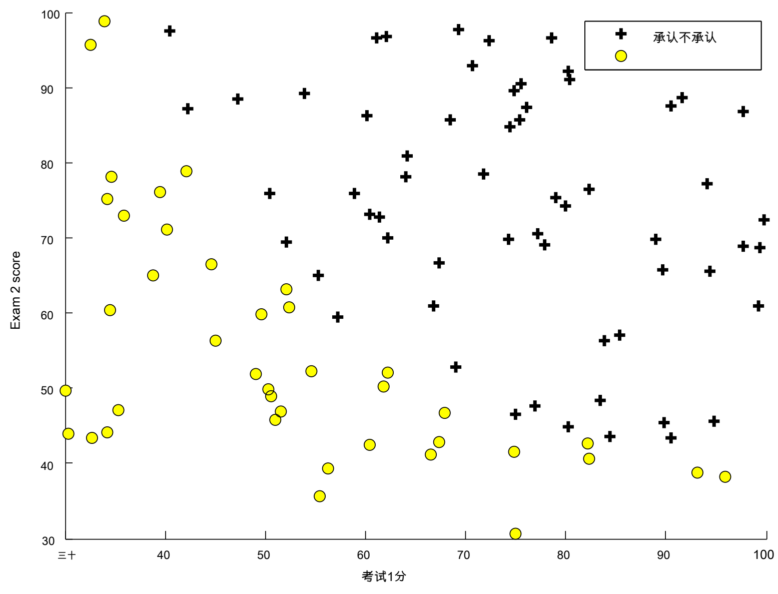


图1：训练数据的散点图

为了帮助您更熟悉绘图，我们已经离开 plotData.m
空的，所以你可以尝试实现它自己。然而，这是一个 *可选的（未分级）运动。* 我们还为实现以下这样你就可以将它复制或引用它。如果你选择复制我们的例子中，请确保您了解每个公司的命令是由咨询倍频文档做的事情。

```
查找%正负面的指标
POS =查找（Y == 1）; NEG =查找（Y == 0）;

%绘制实例
积（X（POS，1），X（位置，2），数k +'，'行宽'，2，...
          'MarkerSize'，7）;
积（X（负，1），X（NEG，2），'KO'，'MarkerFaceColor'，'Y'，...
          'MarkerSize'，7）;
```

1.2实施

1.2.1热身运动：双曲线函数

在开始使用的实际成本函数，记得回归假设去网络定义为：

$$H_{\theta}(X) = G(\theta^{\dagger} X),$$

其中功能 *G* 是双曲线函数。乙状结肠功能去连接的定义为：

$$克(Z) = 1 \frac{}{1 + \ddot{E}_{-ž}}$$

你的第一步是实现这一功能 sigmoid.m 因此它可以通过你的程序的其余部分被调用。当你连接nished，试图通过调用测试几个值 乙状结肠（x）的 在八度命令行。对于很大的正值 X， 乙状结肠应接近1，而对于大的负的值，S形应接近于0评价 乙状结肠（0） 应恰好为0.5给你。你的代码也应与向量和矩阵的工作。 对于一个矩阵，你的函数应该每个元素上执行双曲线函数。

您可以通过键入提交您的解决方案打分 提交 在八度命令行。提交脚本会提示您输入您的用户名和密码，并询问你网络莱要提交哪些。您可以从该网站提交密码。

现在，您应该提交的热身运动。

1.2.2成本函数和梯度

现在，您将实现logistic回归的成本函数和梯度。完成代码 costFunction.m 返回的成本和梯度。

回想一下，在回归的成本函数

$$\hat{J}(\theta) = 1 \frac{}{米} \sum_{I=1}^{米} [-Y(-世) 日志(H_\theta(X(-世))) - (1 - Y(-世)) 日志(1 - H_\theta(X(-世)))],$$

和成本的梯度是相同的长度的矢量 θ 其中， $\hat{J}$日
元件（ $J = 0, 1, 。。。, N$）是定义如下德网络连接：

$$\frac{\partial J(\theta)\partial\theta}{\hat{j}} = 1 \frac{}{米} \sum_{I=1}^{米} (H_\theta(X(-世)) - Y(-世)) X(-世)$$

注意，尽管该梯度看起来等同于线性回归梯度，该公式实际上是二FF erent因为线性和对数回归具有二FF erent德音响nitions $H_\theta(X)$。

一旦你完成， ex2.m 会打电话给你 costFunction 使用的初始参数 θ。 您应该看到的是，成本大约是 0.693。

现在，您应该提交的成本函数和梯度logistic回归。提出两点意见：一为成本函数和一个渐变。

使用1.2.3学习参数 fminunc

在以前的分配，你通过实施gradent血统发现线性回归模型的最佳参数。你写了一个成本函数和计算梯度，然后相应地采取了梯度下降的一步。这一次，而不是采取梯度下降步骤，你将使用一个倍频程称为内置函数 fminunc。

八度的 fminunc 是一种优化求解器，网络连接NDS最小的无约束的[2] **功能。对于逻辑回归，要优化成本函数 $\hat{J}(\theta)$ 与参数 θ。**

具体而言，您要使用 fminunc 到Fi次最佳参数 θ
对于回归成本函数，给一个固定的数据集（中 X 和 $\ddot{y}$
值）。你会传递到 fminunc 以下输入：

?? 参数的初始值，我们正在设法优化。

[2] 在优化约束经常提及的约束的参数，例如，结合的可能值的约束 θ 可以采取（例如， $\theta \leq 1$）。Logistic回归并没有因为这样的限制 θ 允许采取任何实际价值。

?? ，考虑到训练集和特别是当一个函数 *θ*， 计算逻辑回归成本和梯度相对于 *θ* 为数据集（ *X*， *Y*）

在 ex2.m， 我们已经编写的代码调用 fminunc 以正确的参数。

```
对于fminunc%设置选项
选项= optimset（ 'GradObj' ， '上' ， 'MAXITER' ，400）;

%运行fminunc以获得最佳THETA%这个函数将返回theta和成本

的2θ，成本] = ...
        fminunc（@（T）（costFunction（T，X，Y））， 初始THETA，选项）;
```

在此代码片段中，我们定义科幻首先去连接的选项一起使用 fminunc。
具体来说，我们设置了 GradObj 选项 上， 它告诉 fminunc 我们的函数返回的成本和梯度。这允许 fminunc 最小化功能时使用的梯度。此外，我们设置

MAXITER 选项400，使 fminunc 至多400步在终止前运行。

要指定我们正在减少的实际功能，我们使用了"速记"为与@指定功能（ 吨）（costFunction（T，X，Y））。 这就形成了一个功能，有说法 T， 这就要求你costFunction。这使我们能够包住 costFunction 与使用 fminunc。

如果你已经完成了 costFunction 正确， fminunc 将汇聚在正确的优化参数和返回的成本最终科幻和值 *θ*。 注意，通过使用 fminunc， 你没有写任何环路自己，或者设置一个学习速度像你这样的梯度下降。这一切所做 fminunc： 你只需要提供一个函数计算成本和梯度。

一旦 fminunc 完成后， ex2.m 会打电话给你 costFunction 使用的最佳参数函数 *θ*。 您应该看到的是，成本大约是
0.203。
这最终科幻 *θ* 然后值将被用于绘制在训练数据的决策边界，造成类似于图的Fi古尔 2 。我们也鼓励您看看代码 plotDecisionBoundary.m 来看看如何使用绘制这样的边界 *θ* 值。

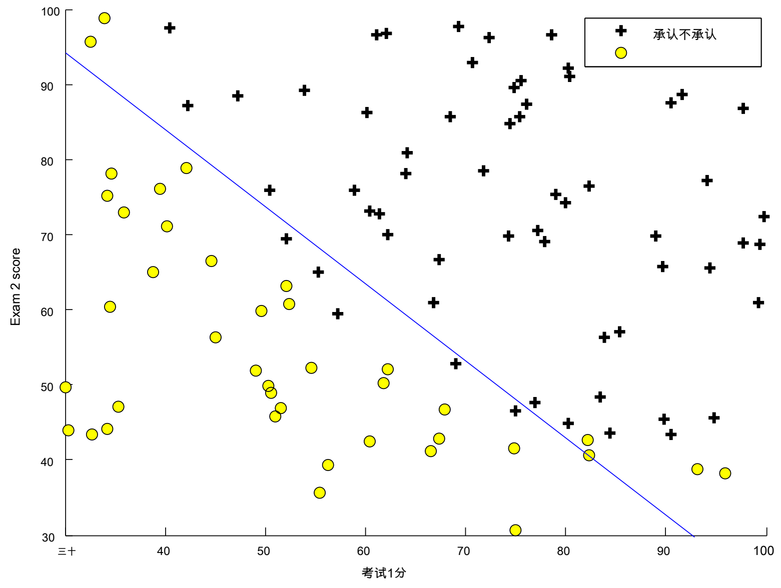


图2：培养具有决策边界数据

1.2.4评估Logistic回归

学习的参数后，您可以使用该模型来预测某个学生是否被录取。对于一个考试1分45和考试2分的85个学生，你应该会看到0.776的录取概率。

评估我们已经找到了参数的质量的另一种方法是看学习模型的预测效果上我们的训练集。在这一部分，**你的任务是完成代码 predict.m。 该 预测 函数将产生"1"或"0"的预测给定的数据集和一个学习参数矢量 θ。**

当你已经完成了代码 predict.m， 该 ex2.m 脚本将继续通过计算实例百分比它得到正确地报告您的 CLASSI网络ER的训练精度。

现在，您应该提交的预测功能的回归。

2正则化回归

在这部分练习中，您将实现正规化回归预测从制造工厂微芯片是否通过质量保证（QA）。QA过程中，每个微芯片经过各种测试，以确保其正常运行。

假设你是工厂的产品经理，你有两个双FF erent测试一些微芯片的测试结果。从这两个测试，你想，以确定是否微芯片应该接受或拒绝。为了帮助你做出决定，你对过去的微芯片的测试结果，从中你可以建立一个逻辑回归模型的数据集。

你会使用其他脚本， EX2 reg.m 完成练习的这一部分。

2.1可视化数据

与此类似演习的前面部分， plotData 被用来产生一个音响古尔像图 3 ，其中所述轴是两个测试分数，和正（ $Y=1$，接受）和负（ $Y=0$，拒绝）的例子被示出与二FF erent标记。

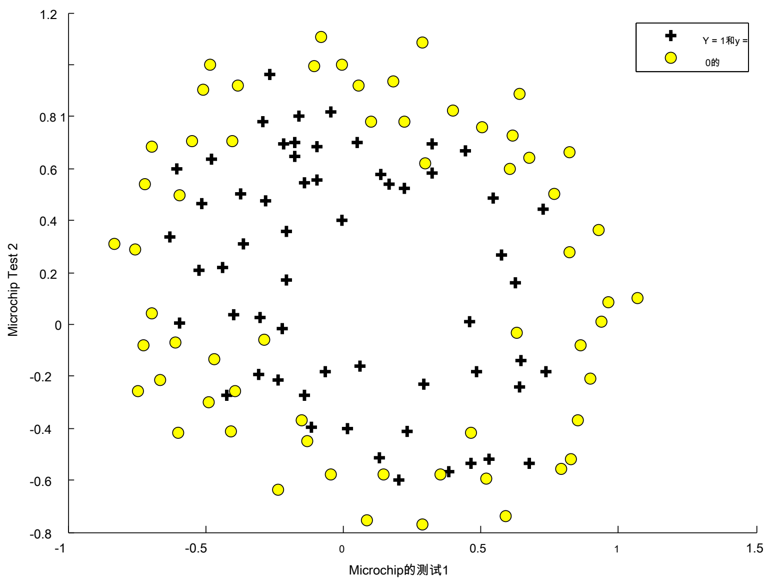


图3：训练数据的叠加

数字 3 可见，我们的数据不能被分为积极的和

通过经由图的直线负例子。因此，逻辑回归的直接应用将无法在本集表现良好，因为回归将只能到第二科幻线性决策边界。

2.2特征映射

到Fi牛逼更好的数据的一种方法是从每个数据点创建更多的功能。在所提供的功能 mapFeature.m， 我们将功能映射到所有多项式项 X_1 和 X_2 一直到第六功率。

$$\text{mapFeature}(X) = \begin{bmatrix} 1 \\ X_1 \\ X_2 \\ X_{21} \\ X_1 X_2 \\ X_{22} \\ X_{31} \\ \vdots \\ X_1 X_{52} \\ X_{62} \end{bmatrix}$$

作为该映射的结果，我们的两个特征（在两个QA测试的分数）载体已被改造成一个28维向量。培训了这个高维特征向量逻辑回归CLASSI网络呃将有更复杂的决策边界，并在我们的2维图表绘制时会出现非线性的。

虽然特征映射使我们能够建立一个更富有表现力CLASSI网络呃，它也更容易受到过度拟合。在练习的下一个部分，你将实现正规化回归到Fi t时的数据，也看到自己正如何帮助打击过度拟合问题。

2.3成本函数和梯度

现在，您将实现代码来计算正规化回归成本函数和梯度。完成代码 costFunctionReg.m 返回的成本和梯度。

回想一下，在回归正规化成本函数

$$\hat{J}(\theta) = \frac{1}{\text{米}} \sum_{I=1}^{\text{米}} [-Y(\text{一世})\,\text{日志}(H_\theta(X(\text{一世}))) - (1 - Y(\text{一世}))\,\text{日志}(1 - H_\theta(X(\text{一世})))] + \frac{\lambda}{2\,\text{米}} \sum_{J=1}^{\tilde{n}} \theta^2_{\text{学家}}$$

请注意，你不应该正规化参数 θ_0。在 八度， 记得开始的索引从1，因此，你不应该在正规化

THETA（1） 参数（其对应于 θ_0）在代码中。成本函数的梯度是一个矢量，其中 $\hat{J}$日 元件被去音响定义如下：

$$\frac{\partial J(\theta)\partial\theta}{0} = 1\frac{}{米}\sum_{I=1}^{米} \left(H_{\theta}(X(-世)) - Y(-世)\right) X(-世) \qquad 对于 J=0$$

$$\frac{\partial J(\theta)\partial\theta}{\hat{J}} = \left(\frac{1}{米}\sum_{I=1}^{米} \left(H_{\theta}(X(-世)) - Y(-世)\right) X(-世) - \frac{\lambda M\theta\hat{J}}{} \right) \qquad 对于 \hat{J} \geq 1$$

一旦你完成， EX2 reg.m 会打电话给你 costFunctionReg 使用的初始值的功能 θ（ 初始化为全零）。您应该看到的是，成本大约是0.693。

现在，您应该提交的成本函数和梯度的正规化回归。做两个意见，一个是成本函数和一个渐变。

使用2.3.1学习参数 fminunc

类似以前的部分，你将使用 fminunc 学习的最优参数 θ。 如果你已经完成了规则化回归的成本和梯度（ costFunctionReg.m ） 正确的话，你应该能够逐步执行的下一部分 EX2 reg.m 学习参数 θ 运用 fminunc。

_

2.4绘制决策边界

为了帮助您可视这个CLASSI网络呃学习的模型，我们所提供的功能 plotDecisionBoundary.m 图表分开的正和负示例中的（非线性）决策边界。在

plotDecisionBoundary.m， 我们通过计算上的均匀分布的网格CLASSI科幻读者的预测，然后绘制非线性的决策边界，并提请在预测从变化的等高线图 $Y=0$〜 $Y=1$.学习的参数后 θ， 在下一步 前reg.m 将绘制类似于图判定边界 4 。

_

2.5可选（未分级）演习

在这部分练习中，你会得到尝试双FF erent正则化参数的数据集，以了解如何正规化防止过度拟合。

注意在决策边界的变化而改变你 *λ*。 用小
λ，你应该ND科幻的CLASSI网络呃得几乎每一个训练例子是正确的，但却得到了非常复杂的边界，因此过度拟合数据（图 五 ）。这是不是一个好的决策边界：例如，它预测一个点 *X* = (- 0.25， 1.5) 被接受（ *Y* = 1 ），这似乎是给定的训练集不正确的决定。

具有较大 *λ*，你应该看到的曲线，显示了简单的决策边界仍然分开肯定和否定得相当好。但是，如果 *λ* 被设置为过高的值，你不会得到一个良好的网络连接T和决策边界将不会跟随数据这么好，因此在拟合数据（图

6 ）。

你不需要提交任何解决方案，这些可选的（未分级）练习。

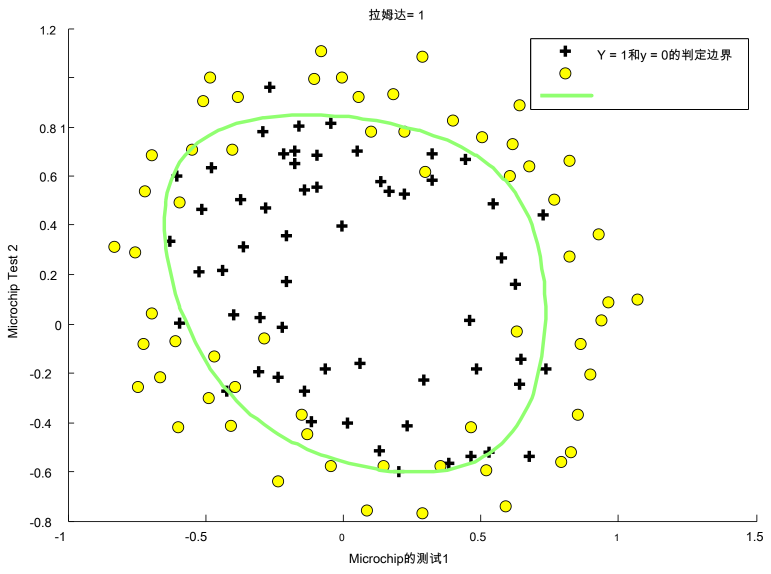


图4：培养具有决策边界数据（ *λ*= 1 ）

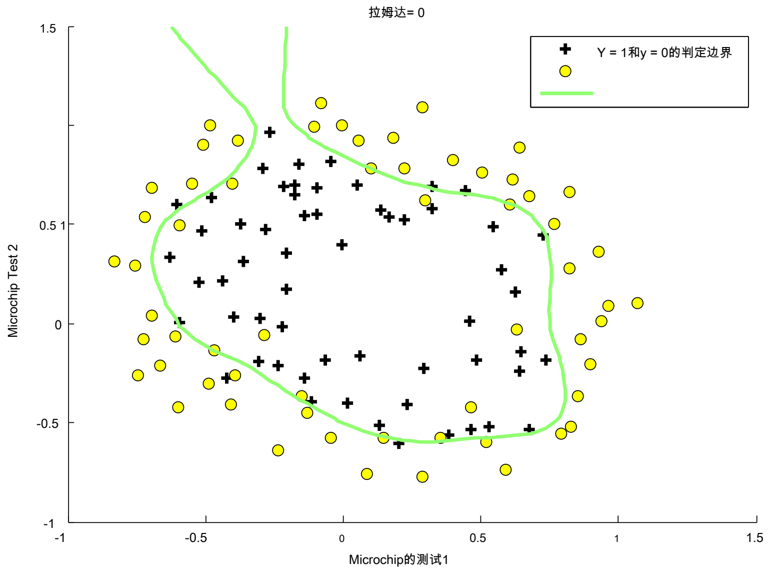


图5：无规则化（过度拟合）（ *λ*=0 ）

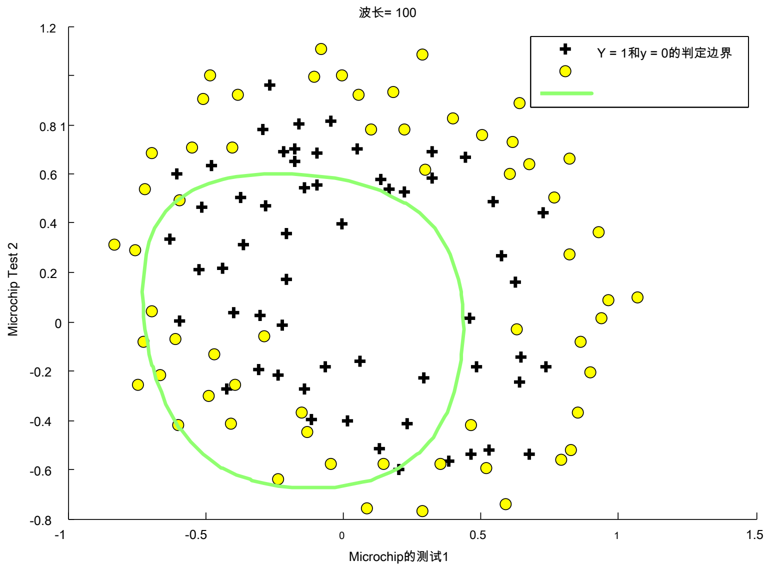


图6：太多正规化（主管拟合）（ *λ*=100 ）

提交和分级

在完成任务的各部分后，请务必使用 提交
功能系统，您的解决方案提交到我们的服务器。以下是这项工作的每个部分是如何拿下了故障。

部分	提交文件	点
双曲线函数	sigmoid.m	5分
logistic回归的计算成本 costFunction.m		30分
梯度回归	costFunction.m	30分
预测功能	predict.m	5分
对于正则LR计算成本	costFunctionReg.m 15分	
梯度正则LR	costFunctionReg.m 15分	
总积分		100点

你被允许多次提交您的解决方案，我们将只需要得分最高的考虑。为了防止速效音响再猜测，该系统强制最小的提交之间5分钟。

编程练习3：多类CLASSI网络阳离子和神经网络

机器学习

介绍

在这个练习中，您将实现一个-VS-所有逻辑回归和神经网络识别手写的数字。在开始编程练习之前，我们强烈建议观看视频讲座，并完成对相关主题的复习题。

要开始使用的演习，下载启动代码并解压缩其内容的目录，你想完成练习。如果需要，可使用 光盘 在八度命令来开始这个练习前切换到该目录。

文件包含在此锻炼

ex3.m - 八音脚本，将帮助您逐步完成部分1
EX3 nn.m - 八音脚本，将帮助您逐步完成第2部分
ex3data1.mat - 训练集的手写数字
ex3weights.mat - 对于神经网络训练的初始权重
submitWeb.m - 另类提交脚本
submit.m - 提交脚本发送您的解决方案，我们的服务器
displayData.m - 功能以帮助可视化数据集
fmincg.m - 功能最小化过程（类似fminunc）
sigmoid.m - 双曲线函数
lrCostFunction.m [？] - Logistic回归成本函数
oneVsAll.m [？] - 培养出一个-VS-所有的多级CLASSI网络呃
predictOneVsAll.m [？] - 预测使用一VS-所有的多级CLASSI网络呃
predict.m [？] - 神经网络预测功能

? 表明科幻LES，你将需要完成

在整个练习中，您将使用脚本 ex3.m 和 EX3 nn.m. _
这些脚本建立数据集的问题，并作出你会写函数的调用。你并不需要修改这些脚本。您只需要修改的功能在其他科幻LES，按照此任务的指示。

从哪里获得帮助

我们还大力鼓励使用在线 Q＆A论坛 讨论与其他学生练习。但是，不看别人写的任何源代码或与他人分享你的源代码。

如果遇到网络错误使用 提交 脚本，你也可以使用在线表单提交您的解决方案。要使用此 *替代* 提交界面，运行 submitWeb 脚本生成提交文件连接（例如，

提交EX2 part1.txt）。 然后，您可以通过编程练习页面网页提交表单提交这个文件（去编程练习页面，**然后选择您要提交的锻炼）。如果您有没有问题，使用通过标准提交系统提交 提交 脚本，你做 *不* 需要**使用这种替代提交界面。

1多类CLASSI网络阳离子

在这个练习中，您将使用逻辑回归和神经网络识别手写的数字（从0到9）。自动手写数字识别是目前广泛使用 - 从认识上的邮件信封邮政编码（邮政编码）来识别写在银行支票金额。这个练习将告诉你如何你学到的方法可以用于此CLASSI网络阳离子任务。

在运动的第一个部分，你将扩展以前的logistic回归的FPGA实现，并将其应用于一个-VS-所有CLASSI网络阳离子。

1.1数据集

你给出的数据集 ex3data1.mat 包含的手写数字5000个训练实例。 [1] **的。 垫 格式，则意味着该数据**

[1] **这是MNIST手写数字数据集的一个子集（ http://yann.lecun.com/ exdb / MNIST / ）。**

已被保存在本机八度/ Matlab的矩阵格式，而不是像科幻csv-文件文本（ASCII）格式。这些矩阵可以通过直接读入程序 加载 命令。加载后，正确的尺寸和值矩阵将出现在你的程序的内存。矩阵就已经被命名，因此你不需要指定名称给他们。

```
从文件%负载保存矩阵
加载（ 'ex3data1.mat' ）;
%的矩阵X和Y，现在会在你的八度环境
```

还有在5000个训练实例 ex3data1.mat， 其中，每个训练实例是20像素由位的20像素的灰度图像。每个像素由表示该位置处的灰度强度的FL浮点数字表示。像素的20乘20栅格“展开”到一个400维向量。每一种训练例子成为我们的数据矩阵X.单行这给我们带来了5000×400矩阵 X 其中，每一行是一个手写数字图像的训练例子。

$$X = \begin{bmatrix} -\ (\ X(1))\ \check{T}- \\ -\ (\ X(2))\ \check{T}- \\ \vdots \\ -\ (\ X(\text{公吨}- \end{bmatrix}$$

训练集的第二部分是一个5000维向量 ÿ 包含用于训练集标签。为了让事情有八度/ Matlab的索引，那里是没有零指数更兼容，我们已制订了数字零值十位。因此，“0”位被标记为“10”，而数字“1”到“9”以它们的自然顺序标记为“1”至“9”。

1.2可视化数据

您将通过可视化训练集的一个子集开始。在第1部分 ex3.m，
代码中随机选择从选择100行 X 并将这些行到 displayData 功能。此函数的每一行映射到一个20像素乘20像素的灰度级图像，并显示图像一起。我们提供的 displayData 功能，并且我们鼓励您检查代码，看看它是如何工作的。运行此步骤后，你应该看到像图的图像

1。

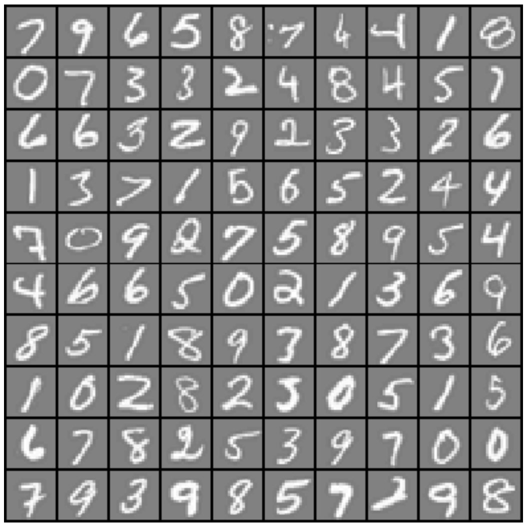


图1：从数据集实例

1.3向量化Logistic回归

您将使用多个单VS-所有逻辑回归模型建立多级CLASSI网络呃。由于有10个班，你需要训练10个独立的逻辑回归CLASSI网络ERS。为了使本次培训éFFI cient，重要的是要确保你的代码以及量化。在本节中，您将实现回归的向量化版本不采用任何 对于 循环。您可以在一个练习作为起点，这个练习使用你的代码。

1.3.1矢量化成本函数

我们将通过编写成本函数的向量化版本开始。回想一下，在（非正规）logistic回归分析，成本函数是

$$\hat{J}(\theta)=1 \frac{}{米} \sum_{I=1}^{米} [-Y(-世)\text{日志}(H_{\theta}(X(-世)))-(1-Y(-世))\text{日志}(1-H_{\theta}(X(-世)))]。$$

为了计算每个元素的总和，我们必须计算 $H_{\theta}(X(-世))$

每一个实例 一世，哪里 $H_{\theta}(X(i))=克(\theta^{\check{T}}X(-世))$ 和 $克(Z)=1 \overline{1+\ddot{E}_{-\check{z}}}$是个双曲线函数。事实证明，我们可以快速计算出这个通过使用矩阵乘法我们所有的例子。让我们去科幻N E X和 θ如

$$X = \begin{bmatrix} -(X(_{1}))\,\check{T}- \\ -(X(_{2}))\,\check{T}- \\ \vdots \\ -(X(\text{公吨}- \end{bmatrix} \quad 和\ \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{\tilde{n}} \end{bmatrix} 。$$

然后，通过计算矩阵乘积 *Xθ*， 我们有

$$X\theta = \begin{bmatrix} -(X(_{1}))\,\check{T}\theta- \\ -(X(_{2}))\,\check{T}\theta- \\ \vdots \\ -(X(\text{公吨}\,\theta- \end{bmatrix} = \begin{bmatrix} -\ \ \theta_{T(}\,X(_{1}))- \\ -\ \ \theta_{T(}\,X(_{2}))- \\ \vdots \\ -\ \ \theta_{T(}\,X(_{\text{米}}))- \end{bmatrix} 。$$

在过去的平等，我们使用的事实， 一个 $\check{T}B=\check{T}$ **一个 如果 一个 和** *b* **是矢量。这使我们能够计算产品** $\theta\,\check{T}\,X($ $_{-\text{世}})$ **我们所有的例子 一世 在一个代码行。**

你的任务是写在网络连接文件的非正规成本函数 lrCostFunction.m
你的实现应该使用的策略，我们上面介绍计算 $\theta\,\check{T}\,X($ $_{-\text{世}})$**。你也应该用量化的方法为成本函数的其余部分。的全矢量版本** lrCostFunction.m **不应包含任何循环。**

（提示：你可能想使用逐元素乘法运算（*）和求和操作。 和 写此功能时）

1.3.2向量化梯度

回想一下，（非正规）逻辑回归成本的梯度是一个矢量，其中 $\hat{J}_{\text{日}}$ **元件被去音响定义为**

$$\frac{\partial J \partial \theta}{\hat{J}} = 1\frac{}{\text{米}} \sum_{I=1}^{\text{米}} \Big(\; (H_{\theta(}\,X(_{-\text{世}}))-Y(_{-\text{世}})\}\,X(_{\tilde{\sigma}\text{世}}) \Big)$$

要通过矢量化数据集此操作，我们开始通过写出来的所有

五

明确的偏导数为所有 $\theta_{ĵ}$,

$$\begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_{ñ}} \end{bmatrix} = \frac{1}{\text{米}} \begin{bmatrix} \sum_{I=1}^{\text{米}} \left((H_{\theta}(X(-\text{世})) - Y(-\text{世})) X(-\text{世})_0 \right) \\ \sum_{I=1}^{\text{米}} \left((H_{\theta}(X(-\text{世})) - Y(-\text{世})) X(-\text{世})_1 \right) \\ \sum_{I=1}^{\text{米}} \left((H_{\theta}(X(-\text{世})) - Y(-\text{世})) X(-\text{世})_2 \right) \\ \vdots \\ \sum_{I=1}^{\text{米}} \left((H_{\theta}(X(-\text{世})) - Y(-\text{世})) X(-\text{世})_{ñ} \right) \end{bmatrix}$$

$$= \frac{1}{\text{米}} \sum_{I=1}^{\text{米}} \left((H_{\theta}(X(-\text{世})) - Y(-\text{世})) X(-\text{世}) \right)$$

$$= \frac{1}{M} X^{T} (H_{\theta}(X) - Y)。 \qquad (1)$$

哪里

$$H_{\theta}(X) - Y = \begin{bmatrix} H_{\theta}(X(1)) - Y(1) \\ H_{\theta}(X(2)) - Y(2) \\ \vdots \\ H_{\theta}(X(1)) - Y(M) \end{bmatrix}。$$

注意 $X(-\text{世})$ 是矢量，而（ $H_{\theta}(X(-\text{世})) - Y(-\text{世})$ 是标量（单数）。为了了解推导的最后一步，让 $\beta_{I} = (H_{\theta}(X(-\text{世})) - Y(-\text{世}))$ 并观察：

$$\sum_{-\text{世}} \beta_{-\text{世}} X(i) = \begin{bmatrix} | & | & & | \\ X(1) & X(2) & \circ\circ\circ & X(M) \\ | & | & & | \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{\text{米}} \end{bmatrix} = X^{T}\beta,$$

其中值 $\beta_{I} = (H_{\theta}(X(-\text{世})) - Y(-\text{世}))$。

上面的表达式允许我们计算所有没有任何环路的偏导数。如果您熟悉线性代数，我们建议您通过矩阵乘法工作上面说服自己，矢量化版本做同样的计算。您现在应该实现方程 1 计算正确的矢量梯度。一旦你完成，完成的功能 lrCostFunction.m 通过实现梯度。

调试提示： 矢量化代码有时可能会非常棘手。调试一个常见的策略是打印出您所使用的工作矩阵的**大小 尺寸 功能。例如，给定一个数据矩阵** X **尺寸100** $\times$ **20（100个例子，20个特征）和** θ**，尺寸为20的矢量** $\times$ **1，你可以观察到，** $X\theta$ **是一个有效的乘法运算，而** θX **不是。此外，如果你有你的代码**的非矢量化版本，你可以比较你的量化代码和非量化代码输出，以确保它们产生相同的输出。

1.3.3矢量化正规化的回归

您已经实现logistic回归的量化之后，你现在将添加正规化的成本函数。回想一下，正规化回归，成本函数是德网络定义为

$$\hat{J}(\theta)=1\frac{}{米}\sum_{I=1}^{米}\left[-Y(-世)\,\text{日志}\,(H_{\theta}(X(-世)))-(1-Y(-世))\,\text{日志}\,(1-H_{\theta}(X(-世)))\right]+\lambda\frac{}{2米}\sum_{J=1}^{\tilde{n}}\theta^{2}_{学家}$$

请注意，您应该 *不* **采用规范** θ_0 **其用于偏置项。**

对于相应地，的偏导数正则化逻辑回归成本 $\theta_{\hat{J}}$**在德网络定义为**

$$\frac{\partial J(\theta)\partial\theta}{0}=1\frac{}{米}\sum_{I=1}^{米}(H_{\theta}(X(-世))-Y(-世))_{\hat{J}}X(-世)\qquad \text{对于 } J=0$$

$$\frac{\partial J(\theta)\partial\theta}{\hat{J}}=\left(\frac{1}{米}\sum_{I=1}^{米}(H_{\theta}(X(-世))-Y(-世))_{\hat{J}}X(-世)\right)+\frac{\lambda M\theta_{\hat{J}}}{}\qquad \text{对于 } \hat{J}\geq 1$$

现在修改代码， lrCostFunction 考虑到正规化。再次，你不应该把任何循环到你的代码。

八度提示： 当实现了正规化回归矢量化，您可能常常只想之和更新的某些元素 θ。 在八度，你可以索引矩阵访问和更新只有某些元素。例如， A(:, 3:5) = B(:, 1:3) will replaces the columns 3 to 5 of A with the columns 1 to 3 from B. One special keyword you can use in indexing is the end keyword in indexing. This allows us to select columns (or rows) until the end of the matrix. For example, A(:, 2:end) will only return elements from the 2 $_{nd}$ to last column of A. Thus, you could use this together with the sum and .^ operations to compute the sum of only the elements you are interested in (e.g., sum(z(2:end).^2)). In the starter code, lrCostFunction.m, we have also provided hints on yet *another* possible method computing the regularized gradient.

You should now submit your vectorized logistic regression cost function.

1.4 One-vs-all Classification

In this part of the exercise, you will implement one-vs-all classification by training multiple regularized logistic regression classifiers, one for each of the K classes in our dataset (Figure 1). In the handwritten digits dataset,
$K = 10$, but your code should work for any value of K.

You should now complete the code in oneVsAll.m to train one classifier for each class. In particular, your code should return all the classifier parameters in a matrix $\Theta \in \mathbb{R}^{K\times(N+1)}$, where each row of Θ corresponds to the learned logistic regression parameters for one class. You can do this with a “for”-loop from 1 to K, training each classifier independently.

Note that the y argument to this function is a vector of labels from 1 to
10, where we have mapped the digit “0” to the label 10 (to avoid confusions with indexing).

When training the classifier for class $k \in \{1, ..., K\}$, you will want a m-dimensional vector of labels y, where $y_j \in 0, 1$ indicates whether the j- th training instance belongs to class k ($y_j = 1$), or if it belongs to a different class ($y_j = 0$). You may find logical arrays helpful for this task.

Octave Tip: Logical arrays in Octave are arrays which contain binary (0 or 1) elements. In Octave, evaluating the expression a == b for a vector a
(of size $m \times 1$) and scalar b will return a vector of the same size as a with ones at positions where the elements of a are equal to b and zeroes where they are different. To see how this works for yourself, try the following code in Octave:

```
    a = 1:10; % Create a and b b = 3; a == b

                         % You should try different values of b here
```

Furthermore, you will be using fmincg for this exercise (instead of fminunc). fmincg works similarly to fminunc, but is more more efficient for dealing with a large number of parameters.

After you have correctly completed the code for oneVsAll.m, the script
ex3.m will continue to use your oneVsAll function to train a multi-class classifier.

You should now submit the training function for one-vs-all classification.

1.4.1 One-vs-all Prediction

After training your one-vs-all classifier, you can now use it to predict the digit contained in a given image. For each input, you should compute the "probability" that it belongs to each class using the trained logistic regression classifiers. Your one-vs-all prediction function will pick the class for which the corresponding logistic regression classifier outputs the highest probability and return the class label (1, 2,..., or *K)* as the prediction for the input example.

You should now complete the code in predictOneVsAll.m to use the one-vs-all classifier to make predictions.

Once you are done, ex3.m will call your predictOneVsAll function using the learned value of Θ.
You should see that the training set accuracy is about
94.9% (i.e., it classifies 94.9% of the examples in the training set correctly).

You should now submit the prediction function for one-vs-all classification.

---

2 Neural Networks

In the previous part of this exercise, you implemented multi-class logistic regression to recognize handwritten digits. However, logistic regression cannot form more complex hypotheses as it is only a linear classifier. [2]

In this part of the exercise, you will implement a neural network to recognize handwritten digits using the same training set as before. The neural network will be able to represent complex models that form non-linear hypotheses. For this week, you will be using parameters from a neural network that we have already trained. Your goal is to implement the feedforward propagation algorithm to use our weights for prediction. In next week's exercise, you will write the backpropagation algorithm for learning the neural network parameters.

The provided script, ex3 nn.m, will help you step through this exercise.

2.1 Model representation

Our neural network is shown in Figure 2 . It has 3 layers – an input layer, a hidden layer and an output layer. Recall that our inputs are pixel values of digit images. Since the images are of size 20 $\times$ 20, this gives us 400 input layer units (excluding the extra bias unit which always outputs +1). As before, the training data will be loaded into the variables X and y.

You have been provided with a set of network parameters ($\Theta^{(1)}, \Theta^{(2)}$) already trained by us. These are stored in ex3weights.mat and will be loaded by ex3 nn.m into Theta1 and Theta2 The parameters have dimensions that are sized for a neural network with 25 units in the second layer and 10 output units (corresponding to the 10 digit classes).

```
% Load saved matrices from file
load( 'ex3weights.mat' );

% The matrices Theta1 and Theta2 will now be in your Octave % environment

% Theta1 has size 25 x 401 % Theta2 has size
10 x 26
```

[2] You could add more features (such as polynomial features) to logistic regression, but that can be very expensive to train.

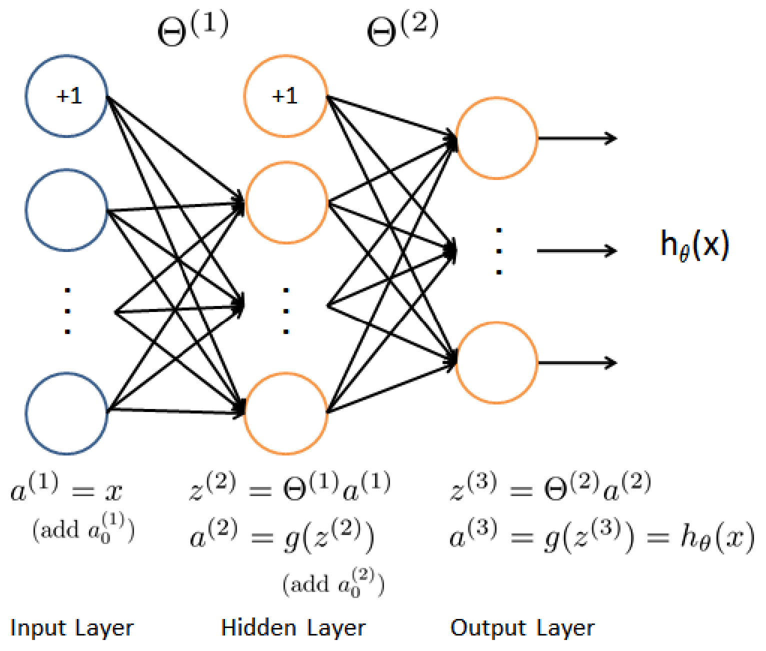


Figure 2: Neural network model.

2.2 Feedforward Propagation and Prediction

Now you will implement feedforward propagation for the neural network. You will need to complete the code in predict.m to return the neural network's prediction.

You should implement the feedforward computation that computes $h_\theta(x^{(i)})$ for every example i and returns the associated predictions. Similar to the one-vs-all classification strategy, the prediction from the neural network will be the label that has the largest output $(h_\theta(x))_k$.

Implementation Note: The matrix X contains the examples in rows. When you complete the code in predict.m, you will need to add the column of 1's to the matrix. The matrices Theta1 and Theta2 contain the parameters for each unit in rows. Specifically, the first row of Theta1

corresponds to the first hidden unit in the second layer. In Octave, when you compute $z^{(2)} = \Theta^{(1)} a^{(1)}$, be sure that you index (and if necessary, transpose) X correctly so that you get $a^{(l)}$ as a column vector.

Once you are done, ex3 nn.m will call your predict function using the loaded set of parameters for Theta1 and Theta2. You should see that the accuracy is about 97.5%. After that, an interactive sequence will launch displaying images from the training set one at a time, while the console prints out the predicted label for the displayed image. To stop the image sequence, press Ctrl-C.

You should now submit the neural network prediction function.

Submission and Grading

After completing this assignment, be sure to use the submit function to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	Submitted File	Points
Regularized Logisic Regression	lrCostFunction.m	30 points
One-vs-all classifier training	oneVsAll.m	20 points
One-vs-all classifier prediction	predictOneVsAll.m 20 points	
Neural Network Prediction Function predict.m		30 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.

Programming Exercise 4: Neural Networks Learning

Machine Learning

Introduction

In this exercise, you will implement the backpropagation algorithm for neural networks and apply it to the task of hand-written digit recognition. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the cd command in Octave to change to this directory before starting this exercise.

Files included in this exercise

ex4.m - Octave script that will help step you through the exercise
ex4data1.mat - Training set of hand-written digits
ex4weights.mat - Neural network parameters for exercise 4
submit.m - Submission script that sends your solutions to our servers
submitWeb.m - Alternative submission script
displayData.m - Function to help visualize the dataset
fmincg.m - Function minimization routine (similar to fminunc)
sigmoid.m - Sigmoid function
computeNumericalGradient.m - Numerically compute gradients
checkNNGradients.m - Function to help check your gradients
debugInitializeWeights.m - Function for initializing weights
predict.m - Neural network prediction function
[?] sigmoidGradient.m - Compute the gradient of the sigmoid function
[?] randInitializeWeights.m - Randomly initialize weights
[?] nnCostFunction.m - Neural network cost function

? indicates files you will need to complete

Throughout the exercise, you will be using the script ex4.m. These scripts set up the dataset for the problems and make calls to functions that you will write. You do not need to modify the script. You are only required to modify functions in other files, by following the instructions in this assignment.

Where to get help

We also strongly encourage using the online Q&A Forum to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

If you run into network errors using the submit script, you can also use an online form for submitting your solutions. To use this *alternative* submission interface, run the submitWeb script to generate a submission file (e.g.,
submit ex2 part1.txt). You can then submit this file through the web submission form in the programming exercises page (go to the programming exercises page, then select the exercise you are submitting for). If you are having no problems submitting through the standard submission system using the submit script, you do *not* need to use this alternative submission interface.

---

1 Neural Networks

In the previous exercise, you implemented feedforward propagation for neural networks and used it to predict handwritten digits with the weights we provided. In this exercise, you will implement the backpropagation algorithm to *learn* the parameters for the neural network.

The provided script, ex4.m, will help you step through this exercise.

1.1 Visualizing the data

In the first part of ex4.m, the code will load the data and display it on a 2-dimensional plot (Figure 1) by calling the function displayData.

This is the same dataset that you used in the previous exercise. There are 5000 training examples in ex3data1.mat, where each training example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by

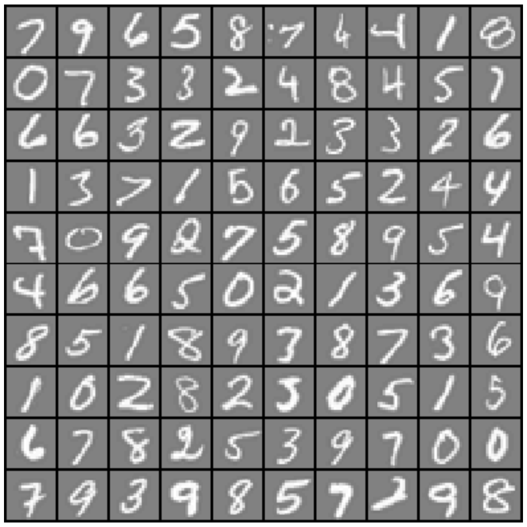


Figure 1: Examples from the dataset

a floating point number indicating the grayscale intensity at that location. The 20 by 20 grid of pixels is "unrolled" into a 400-dimensional vector. Each of these training examples becomes a single row in our data matrix X. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image.

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

The second part of the training set is a 5000-dimensional vector y that contains labels for the training set. To make things more compatible with Octave/Matlab indexing, where there is no zero index, we have mapped the digit zero to the value ten. Therefore, a "0" digit is labeled as "10", while the digits "1" to "9" are labeled as "1" to "9" in their natural order.

1.2 Model representation

Our neural network is shown in Figure 2 . It has 3 layers – an input layer, a hidden layer and an output layer. Recall that our inputs are pixel values of digit images. Since the images are of size 20 $\times$ 20, this gives us 400 input layer units (not counting the extra bias unit which always outputs +1). The training data will be loaded into the variables X and y by the ex4.m script.

You have been provided with a set of network parameters ($\Theta^{(1)}, \Theta^{(2)}$) already trained by us. These are stored in ex4weights.mat and will be loaded by ex4.m into Theta1 and Theta2. The parameters have dimensions that are sized for a neural network with 25 units in the second layer and 10 output units (corresponding to the 10 digit classes).

```
% Load saved matrices from file
load( 'ex4weights.mat' );

% The matrices Theta1 and Theta2 will now be in your workspace % Theta1 has size 25 x 401 % Theta2 has size 10
x 26
```

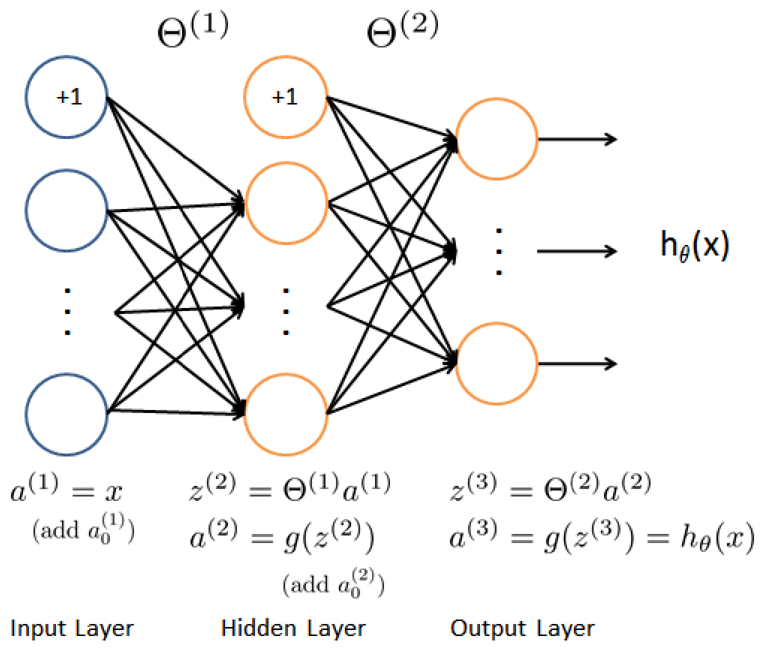


Figure 2: Neural network model.

1.3 Feedforward and cost function

Now you will implement the cost function and gradient for the neural network. First, complete the code in nnCostFunction.m to return the cost.

Recall that the cost function for the neural network (without regularization) is

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right],$$

where $h_\theta(x^{(i)})$ is computed as shown in the Figure 2 and $K = 10$ is the total number of possible labels. Note that $h_\theta(x^{(i)})_k = a_k^{(3)}$ is the activation (output value) of the k- th output unit. Also, recall that whereas the original labels (in the variable y) were 1, 2, ..., 10, for the purpose of training a neural network, we need to recode the labels as vectors containing only values 0 or
1, so that

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots \text{ or } \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

For example, if $x^{(i)}$ is an image of the digit 5, then the corresponding $y^{(i)}$ (that you should use with the cost function) should be a 10-dimensional vector with $y_5 = 1$, and the other elements equal to 0. You should implement the feedforward computation that computes $h_\theta(x^{(i)})$ for every example i and sum the cost over all examples. Your code should also work for a dataset of any size, with any number of labels (you can assume that there are always at least $K \geq 3$ labels).

Implementation Note: The matrix X contains the examples in rows (i.e., X(i,:)' is the i-th training example $x^{(i)}$, expressed as a $n \times 1$ vector.) When you complete the code in nnCostFunction.m, you will need to add the column of 1's to the X matrix. The parameters for each unit in the neural network is represented in Theta1 and Theta2 as one row. Specifically, the first row of Theta1 corresponds to the first hidden unit in the second layer. You can use a for- loop over the examples to compute the cost.

Once you are done, ex4.m will call your nnCostFunction using the loaded set of parameters for Theta1 and Theta2. You should see that the cost is about 0.287629.

You should now submit the neural network cost function (feedforward).

1.4 Regularized cost function

The cost function for neural networks with regularization is given by

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

You can assume that the neural network will only have 3 layers – an input layer, a hidden layer and an output layer. However, your code should work for any number of input units, hidden units and outputs units. While we have explicitly listed the indices above for $\Theta^{(1)}$ and $\Theta^{(2)}$ for clarity, do note that your code should in general work with $\Theta^{(1)}$ and $\Theta^{(2)}$ of any size.

Note that you should not be regularizing the terms that correspond to the bias. For the matrices Theta1 and Theta2, this corresponds to the first column of each matrix. You should now add regularization to your cost function. Notice that you can first compute the unregularized cost function

J using your existing nnCostFunction.m and then later add the cost for the regularization terms.

Once you are done, ex4.m will call your nnCostFunction using the loaded set of parameters for Theta1 and Theta2, and $\lambda = 1$. You should see that the cost is about 0.383770.

You should now submit the regularized neural network cost function (feedforward).

2 Backpropagation

In this part of the exercise, you will implement the backpropagation algorithm to compute the gradient for the neural network cost function. You will need to complete the nnCostFunction.m so that it returns an appropriate value for grad. Once you have computed the gradient, you will be able to train the neural network by minimizing the cost function $J(\Theta)$ using an advanced optimizer such as fmincg.

You will first implement the backpropagation algorithm to compute the gradients for the parameters for the (unregularized) neural network. After you have verified that your gradient computation for the unregularized case is correct, you will implement the gradient for the regularized neural network.

2.1 Sigmoid gradient

To help you get started with this part of the exercise, you will first implement the sigmoid gradient function. The gradient for the sigmoid function can be computed as

$$g'(z) = \frac{d}{dz} g(z) = g(z)(1 - g(z))$$

where

$$\text{sigmoid}(z) = g(z) = \frac{1}{1 + e^{-z}}.$$

When you are done, try testing a few values by calling sigmoidGradient(z) at the Octave command line. For large values (both positive and negative) of z, the gradient should be close to 0. When z = 0, the gradient should be exactly 0.25. Your code should also work with vectors and matrices. For a matrix, your function should perform the sigmoid gradient function on every element.

You should now submit the sigmoid gradient function.

2.2 Random initialization

When training neural networks, it is important to randomly initialize the parameters for symmetry breaking. One effective strategy for random initialization is to randomly select values for $\Theta^{(l)}$ uniformly in the range $[-_{init}, {}_{init}]$.

You should use $_{init} = 0.12$.[1] This range of values ensures that the parameters are kept small and makes the learning more efficient.

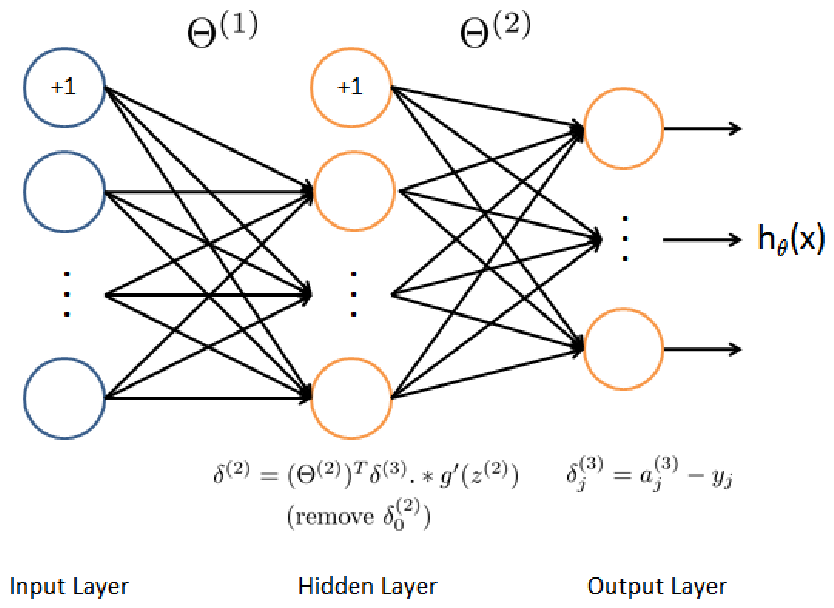
Your job is to complete randInitializeWeights.m to initialize the weights for Θ; modify the file and fill in the following code:

[1] One effective strategy for choosing $_{init}$ is to base it on the number of units in the network. A good choice of $_{init}$ is $_{init} = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}}$, where $L_{in} = s_l$ and $L_{out} = s_{l+1}$ are the number of units in the layers adjacent to $\Theta^{(l)}$.

```
% Randomly initialize the weights to small values
epsilon init = 0.12;
W = rand(L out, 1 + L in) * 2 * epsilon init − epsilon init;
```

You do not need to submit any code for this part of the exercise.

2.3 Backpropagation



$\Theta^{(1)}$
$\Theta^{(2)}$
+1
+1
$h_\theta(x)$
$\delta^{(2)} = (\Theta^{(2)})^T\delta^{(3)} .* g'(z^{(2)})$
(remove $\delta_0^{(2)}$)
$\delta_j^{(3)} = a_j^{(3)} - y_j$
Input Layer
Hidden Layer
Output Layer

Figure 3: Backpropagation Updates.

Now, you will implement the backpropagation algorithm. Recall that the intuition behind the backpropagation algorithm is as follows. Given a training example ($x^{(t)}, y^{(t)}$), we will first run a "forward pass" to compute all the activations throughout the network, including the output value of the hypothesis $h_\Theta(x)$. Then, for each node j in layer l, we would like to compute an "error term" $\delta^{(l)}_j$ that measures how much that node was "responsible" for any errors in our output.

For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to define $\delta^{(3)}_j$ (since layer 3 is the output layer). For the hidden units, you will compute $\delta^{(l)}_j$ based on a weighted average of the error terms of the nodes in layer (l + 1).

In detail, here is the backpropagation algorithm (also depicted in Figure 3). You should implement steps 1 to 4 in a loop that processes one example at a time. Concretely, you should implement a for-loop for t = 1:m and place steps 1-4 below inside the for-loop, with the t_{th} iteration performing the calculation on the t_{th} training example ($x^{(t)}, y^{(t)}$). Step 5 will divide the accumulated gradients by m to obtain the gradients for the neural network cost function.

1. Set the input layer's values ($a^{(1)}$) to the t- th training example $x^{(t)}$. Perform a feedforward pass (Figure 2), computing the activations ($z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)}$) for layers 2 and 3. Note that you need to add a + 1 term to ensure that the vectors of activations for layers $a^{(1)}$ and $a^{(2)}$ also include the bias unit. In Octave, if a 1 is a column vector, adding one corresponds to
a 1 = [1 ; a 1].

2. For each output unit k in layer 3 (the output layer), set

$$\delta^{(3)}_k = (a^{(3)}_k - y_k),$$

where $y_k \in \{0, 1\}$ indicates whether the current training example belongs to class k ($y_k = 1$), or if it belongs to a different class ($y_k = 0$). You may find logical arrays helpful for this task (explained in the previous programming exercise).

3. For the hidden layer $l = 2$, set

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

4. Accumulate the gradient from this example using the following formula. Note that you should skip or remove $\delta^{(2)}_0$. In Octave, removing $\delta^{(2)}_0$ corresponds to delta 2 = delta 2(2:end).

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

5. Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $\frac{1}{m}$:

$$\frac{\partial}{\partial \Theta^{(l)}_{ij}} J(\Theta) = D^{(l)}_{ij} = \frac{1}{m} \Delta^{(l)}_{ij}$$

Octave Tip: You should implement the backpropagation algorithm only after you have successfully completed the feedforward and cost functions. While implementing the backpropagation algorithm, it is often useful to use the size function to print out the sizes of the variables you are working with if you run into dimension mismatch errors (" nonconformant arguments " errors in Octave).

After you have implemented the backpropagation algorithm, the script
ex4.m will proceed to run gradient checking on your implementation. The gradient check will allow you to increase your confidence that your code is computing the gradients correctly.

2.4 Gradient checking

In your neural network, you are minimizing the cost function $J(\Theta)$. To perform gradient checking on your parameters, you can imagine "unrolling" the parameters $\Theta^{(1)}, \Theta^{(2)}$ into a long vector θ. By doing so, you can think of the cost function being $J(\theta)$ instead and use the following gradient checking procedure.

Suppose you have a function $f_i(\theta)$ that purportedly computes $\frac{\partial}{\partial\theta_i} J(\theta)$; you'd like to check if f_i is outputting correct derivative values.

$$\text{Let } \theta^{(i+)} = \theta + \begin{bmatrix} 0 \\ 0\ldots \\ \vdots \\ 0 \end{bmatrix} \quad \text{and } \theta^{(i-)} = \theta - \begin{bmatrix} 0 \\ 0\ldots \\ \vdots \\ 0 \end{bmatrix}$$

So, $\theta^{(i+)}$ is the same as θ, except its i- th element has been incremented by
. Similarly, $\theta^{(i-)}$ is the corresponding vector with the i- th element decreased by . You can now numerically verify $f_i(\theta)$'s correctness by checking, for each
i, that:

$$f_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2}.$$

The degree to which these two values should approximate each other will depend on the details of J. But assuming $= 10^{-4}$, you'll usually find that the left- and right-hand sides of the above will agree to at least 4 significant digits (and often many more).

We have implemented the function to compute the numerical gradient for you in computeNumericalGradient.m. While you are not required to modify the file, we highly encourage you to take a look at the code to understand how it works.

In the next step of ex4.m, it will run the provided function checkNNGradients.m which will create a small neural network and dataset that will be used for checking your gradients. If your backpropagation implementation is correct, you should see a relative difference that is less than 1e-9.

Practical Tip: When performing gradient checking, it is much more efficient to use a small neural network with a relatively small number of input units and hidden units, thus having a relatively small number of parameters. Each dimension of θ requires two evaluations of the cost function and this can be expensive. In the function checkNNGradients,

our code creates a small random model and dataset which is used with computeNumericalGradient for gradient checking. Furthermore, after you are confident that your gradient computations are correct, you should turn off gradient checking before running your learning algorithm.

Practical Tip: Gradient checking works for any function where you are computing the cost and the gradient. Concretely, you can use the same computeNumericalGradient.m function to check if your gradient implementations for the other exercises are correct too (e.g., logistic regression's cost function).

Once your cost function passes the gradient check for the (unregularized) neural network cost function, you should submit the neural network gradient function (backpropagation).

2.5 Regularized Neural Networks

After you have successfully implemeted the backpropagation algorithm, you will add regularization to the gradient. To account for regularization, it turns out that you can add this as an additional term *after* computing the gradients using backpropagation.

Specifically, after you have computed $\Delta^{(l)}_{ij}$ using backpropagation, you should add regularization using

$$\frac{\partial}{\partial \Theta^{(l)}_{ij}} J(\Theta) = D^{(l)}_{ij} = \frac{1}{m}\Delta^{(l)}_{ij} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial \Theta^{(l)}_{ij}} J(\Theta) = D^{(l)}_{ij} = \frac{1}{m}\Delta^{(l)}_{ij} + \frac{\lambda}{m}\Theta^{(l)}_{ij} \quad \text{for } j \geq 1$$

Note that you should *not* be regularizing the first column of $\Theta^{(l)}$ which is used for the bias term. Furthermore, in the parameters $\Theta^{(l)}_{ij}$, i is indexed starting from 1, and j is indexed starting from 0. Thus,

$$\Theta^{(l)} = \begin{bmatrix} \Theta^{(l)}_{1,0} & \Theta^{(l)}_{1,1} & \dots \\ \Theta^{(l)}_{2,0} & \Theta^{(l)}_{2,1} \\ \vdots & & \ddots \end{bmatrix}.$$

Somewhat confusingly, indexing in Octave starts from 1 (for both i and j), thus Theta1(2, 1) actually corresponds to $\Theta^{(l)}_{2,0}$ (i.e., the entry in the second row, first column of the matrix $\Theta^{(1)}$ shown above)

Nowmodify your code that computes grad in nnCostFunction to account for regularization. After you are done, the ex4.m script will proceed to run gradient checking on your implementation. If your code is correct, you should expect to see a relative difference that is less than 1e-9.

You should now submit your regularized neural network gradient.

2.6 Learning parameters using fmincg

After you have successfully implemented the neural network cost function and gradient computation, the next step of the ex4.m script will use fmincg to learn a good set parameters.

After the training completes, the ex4.m script will proceed to report the training accuracy of your classifier by computing the percentage of examples it got correct. If your implementation is correct, you should see a reported training accuracy of about 95.3% (this may vary by about 1% due to the random initialization). It is possible to get higher training accuracies by training the neural network for more iterations. We encourage you to try training the neural network for more iterations (e.g., set MaxIter to 400) and also vary the regularization parameter λ. With the right learning settings, it is possible to get the neural network to perfectly fit the training set.

3 Visualizing the hidden layer

One way to understand what your neural network is learning is to visualize what the representations captured by the hidden units. Informally, given a particular hidden unit, one way to visualize what it computes is to find an input x that will cause it to activate (that is, to have an activation value ($a^{(l)}_i$) close to 1). For the neural network you trained, notice that the i^{th} row of $\Theta^{(1)}$ is a 401-dimensional vector that represents the parameter for the i^{th} hidden unit. If we discard the bias term, we get a 400 dimensional vector that represents the weights from each input pixel to the hidden unit.

Thus, one way to visualize the "representation" captured by the hidden unit is to reshape this 400 dimensional vector into a 20 × 20 image and display it.[2] The next step of ex4.m does this by using the displayData function and it will show you an image (similar to Figure 4) with 25 units, each corresponding to one hidden unit in the network.

In your trained network, you should find that the hidden units corresponds roughly to detectors that look for strokes and other patterns in the input.

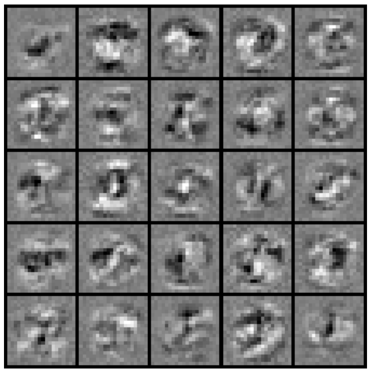


Figure 4: Visualization of Hidden Units.

3.1 Optional (ungraded) exercise

In this part of the exercise, you will get to try out different learning settings for the neural network to see how the performance of the neural network varies with the regularization parameter λ and number of training steps (the MaxIter option when using fmincg).

Neural networks are very powerful models that can form highly complex decision boundaries. Without regularization, it is possible for a neural network to "overfit" a training set so that it obtains close to 100% accuracy on

[2] It turns out that this is equivalent to finding the input that gives the highest activation for the hidden unit, given a "norm" constraint on the input (i.e., $\|x\|_2 \leq 1$).

the training set but does not as well on new examples that it has not seen before. You can set the regularization λ to a smaller value and the MaxIter
parameter to a higher number of iterations to see this for youself.

You will also be able to see for yourself the changes in the visualizations of the hidden units when you change the learning parameters λ and MaxIter.

You do not need to submit any solutions for this optional (ungraded) exercise.

Submission and Grading

After completing various parts of the assignment, be sure to use the submit
function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	Submitted File	Points
Feedforward and Cost Function	nnCostFunction.m	30 points
Regularized Cost Function	nnCostFunction.m	15 points
Sigmoid Gradient	sigmoidGradient.m	5 points
Neural Net Gradient Function (Backpropagation)	nnCostFunction.m	40 points
Regularized Gradient	nnCostFunction.m	10 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.

Programming Exercise 5: Regularized Linear Regression and Bias v.s. Variance

Machine Learning

Introduction

In this exercise, you will implement regularized linear regression and use it to study models with different bias-variance properties. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the cd command in Octave to change to this directory before starting this exercise.

Files included in this exercise

ex5.m - Octave script that will help step you through the exercise
ex5data1.mat - Dataset
submit.m - Submission script that sends your solutions to our servers
submitWeb.m - Alternative submission script
featureNormalize.m - Feature normalization function
fmincg.m - Function minimization routine (similar to fminunc)
plotFit.m - Plot a polynomial fit
trainLinearReg.m - Trains linear regression using your cost function
[?] linearRegCostFunction.m - Regularized linear regression cost function

[?] learningCurve.m - Generates a learning curve
[?] polyFeatures.m - Maps data into polynomial feature space
[?] validationCurve.m - Generates a cross validation curve

? indicates files you will need to complete

Throughout the exercise, you will be using the script ex5.m. These scripts set up the dataset for the problems and make calls to functions that you will write. You are only required to modify functions in other files, by following the instructions in this assignment.

Where to get help

We also strongly encourage using the online Q&A Forum to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

If you run into network errors using the submit script, you can also use an online form for submitting your solutions. To use this *alternative* submission interface, run the submitWeb script to generate a submission file (e.g.,
submit ex5 part2.txt). You can then submit this file through the web submission form in the programming exercises page (go to the programming exercises page, then select the exercise you are submitting for). If you are having no problems submitting through the standard submission system using the submit script, you do *not* need to use this alternative submission interface.

---

1 Regularized Linear Regression

In the first half of the exercise, you will implement regularized linear regression to predict the amount of water flowing out of a dam using the change of water level in a reservoir. In the next half, you will go through some diagnostics of debugging learning algorithms and examine the effects of bias v.s. variance.

The provided script, ex5.m, will help you step through this exercise.

1.1 Visualizing the dataset

We will begin by visualizing the dataset containing historical records on the change in the water level, *x,* and the amount of water flowing out of the dam,
y.

This dataset is divided into three parts:

- A training set that your model will learn on: X, y

- A cross validation set for determining the regularization parameter: Xval, yval
- A test set for evaluating performance. These are "unseen" examples which your model did not see during training: Xtest, ytest

The next step of ex5.m will plot the training data (Figure 1). In the following parts, you will implement linear regression and use that to fit a straight line to the data and plot learning curves. Following that, you will implement polynomial regression to find a better fit to the data.

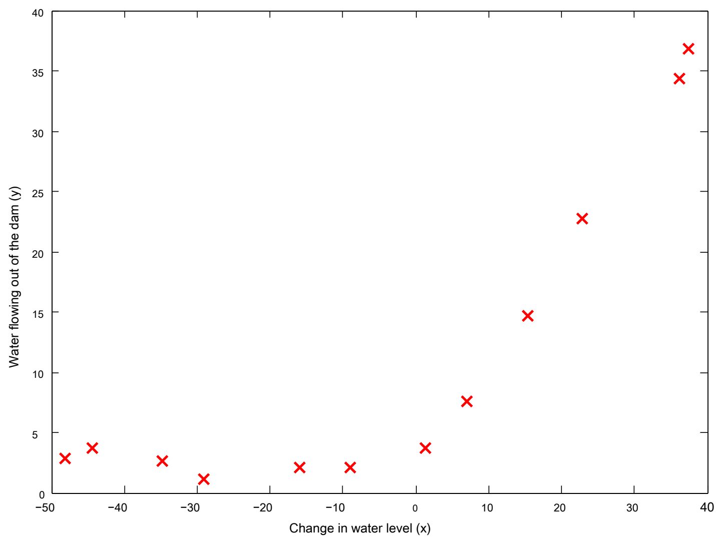


Figure 1: Data

1.2 Regularized linear regression cost function

Recall that regularized linear regression has the following cost function:

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2\right) + \frac{\lambda}{2m}\left(\sum_{j=1}^{n}\theta_j^2\right),$$

where λ is a regularization parameter which controls the degree of regularization (thus, help preventing overfitting). The regularization term puts a penalty on the overal cost J. As the magnitudes of the model parameters θ_j increase, the penalty increases as well. Note that you should not regularize the θ_0 term. (In Octave, the θ_0 term is represented as theta(1) since indexing in Octave starts from 1).

You should now complete the code in the file linearRegCostFunction.m.
Your task is to write a function to calculate the regularized linear regression cost function. If possible, try to vectorize your code and avoid writing loops. When you are finished, the next part of ex5.m will run your cost function using theta initialized at [1; 1]. You should expect to see an output of

303.993.

You should now submit your regularized linear regression cost function.

1.3 Regularized linear regression gradient

Correspondingly, the partial derivative of regularized linear regression's cost for θ_j is defined as

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1$$

In linearRegCostFunction.m, add code to calculate the gradient, returning it in the variable grad. When you are finished, the next part of
ex5.m will run your gradient function using theta initialized at [1; 1].
You should expect to see a gradient of [- 15.30; 598.250].

You should now submit your regularized linear regression gradient function.

1.4 Fitting linear regression

Once your cost function and gradient are working correctly, the next part of
ex5.m will run the code in trainLinearReg.m to compute the optimal values of θ. This training function uses fmincg to optimize the cost function.

In this part, we set regularization parameter λ to zero. Because our current implementation of linear regression is trying to fit a 2-dimensional θ,
regularization will not be incredibly helpful for a θ of such low dimension. In the later parts of the exercise, you will be using polynomial regression with regularization.

Finally, the ex5.m script should also plot the best fit line, resulting in an image similar to Figure 2
. The best fit line tells us that the model is

not a good fit to the data because the data has a non-linear pattern. While visualizing the best fit as shown is one possible way to debug your learning algorithm, it is not always easy to visualize the data and model. In the next section, you will implement a function to generate learning curves that can help you debug your learning algorithm even if it is not easy to visualize the data.

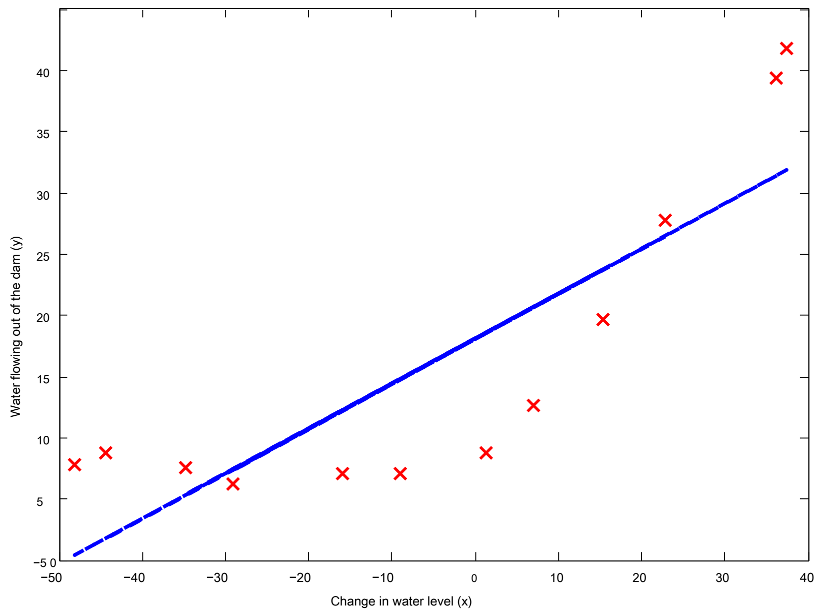


Figure 2: Linear Fit

2 Bias-variance

An important concept in machine learning is the bias-variance tradeoff. Models with high bias are not complex enough for the data and tend to underfit, while models with high variance overfit to the training data.

In this part of the exercise, you will plot training and test errors on a learning curve to diagnose bias-variance problems.

2.1 Learning curves

You will now implement code to generate the learning curves that will be useful in debugging learning algorithms. Recall that a learning curve plots training and cross validation error as a function of training set size. Your job is to fill in learningCurve.m so that it returns a vector of errors for the training set and cross validation set.

To plot the learning curve, we need a training and cross validation set error for different *training* set sizes. To obtain different training set sizes, you should use different subsets of the original training set X. Specifically, for a training set size of i, you should use the first i examples (i.e., X(1:i,:)

and y(1:i)).

You can use the trainLinearReg function to find the θ parameters. Note that the lambda is passed as a parameter to the learningCurve function. After learning the θ parameters, you should compute the error on the training and cross validation sets. Recall that the training error for a dataset is defined as

$$J_{\text{train}}(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2\right].$$

In particular, note that the training error does not include the regularization term. One way to compute the training error is to use your existing cost function and set λ to 0 *only* when using it to compute the training error and cross validation error. When you are computing the training set error, make sure you compute it on the training subset (i.e., X(1:n,:) and y(1:n))

(instead of the entire training set). However, for the cross validation error, you should compute it over the *entire* cross validation set. You should store the computed errors in the vectors error_train and error_val.

When you are finished, ex5.m wil print the learning curves and produce a plot similar to Figure 3 .

You should now submit your learning curve function.

In Figure 3 , you can observe that *both* the train error and cross validation error are high when the number of training examples is increased. This reflects a high bias problem in the model – the linear regression model is too simple and is unable to fit our dataset well. In the next section, you will implement polynomial regression to fit a better model for this dataset.

3 Polynomial regression

The problem with our linear model was that it was too simple for the data and resulted in underfitting (high bias). In this part of the exercise, you will

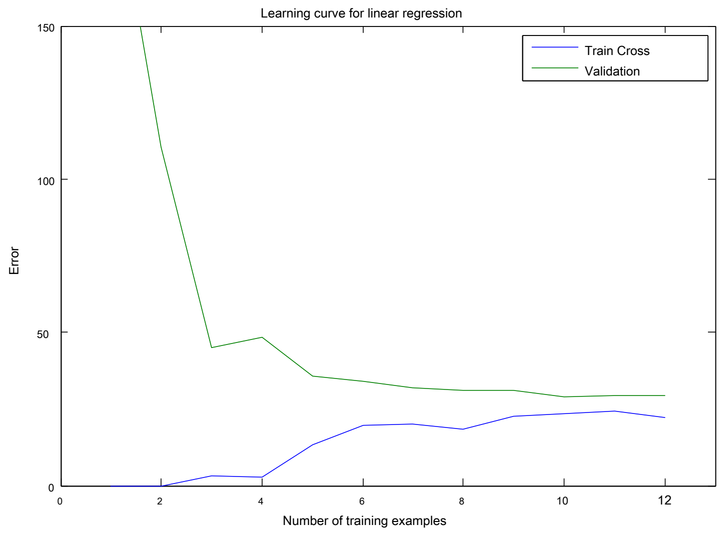


Figure 3: Linear regression learning curve

address this problem by adding more features.

For use polynomial regression, our hypothesis has the form:

$$h_\theta(x) = \theta_0 + \theta_1 * (\text{waterLevel}) + \theta_2 * (\text{waterLevel})^2 + \cdots + \theta_p * (\text{waterLevel})^p$$
$$= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p.$$

Notice that by defining $x_1 = (\text{waterLevel})$, $x_2 = (\text{waterLevel})^2, \ldots, x_p = (\text{waterLevel})^p$, we obtain a linear regression model where the features are the various powers of the original value (waterLevel).

Now, you will add more features using the higher powers of the existing feature x in the dataset. Your task in this part is to complete the code in polyFeatures.m so that the function maps the original training set X of size $m \times 1$ into its higher powers. Specifically, when a training set X of size $m \times 1$ is passed into the function, the function should return a $m \times p$ matrix X poly, where column 1 holds the original values of X, column 2 holds the values of X.^2, column 3 holds the values of X.^3, and so on. Note that you don't have to account for the zero-eth power in this function.

Now you have a function that will map features to a higher dimension, and Part 6 of ex5.m will apply it to the training set, the test set, and the cross validation set (which you haven't used yet).

You should now submit your polynomial feature mapping function.

3.1 Learning Polynomial Regression

After you have completed polyFeatures.m, the ex5.m script will proceed to train polynomial regression using your linear regression cost function.

Keep in mind that even though we have polynomial terms in our feature vector, we are still solving a linear regression optimization problem. The polynomial terms have simply turned into features that we can use for linear regression. We are using the same cost function and gradient that you wrote for the earlier part of this exercise.

For this part of the exercise, you will be using a polynomial of degree 8. It turns out that if we run the training directly on the projected data, will not work well as the features would be badly scaled (e.g., an example with
$x = 40$ will now have a feature $x_8 = 40^8 = 6.5 \times 10^{12}$). Therefore, you will need to use feature normalization.

Before learning the parameters θ for the polynomial regression, ex5.m will first call featureNormalize and normalize the features of the training set, storing the mu, sigma parameters separately. We have already implemented this function for you and it is the same function from the first exercise.

After learning the parameters θ, you should see two plots (Figure 4 , 5) generated for polynomial regression with $\lambda = 0$.

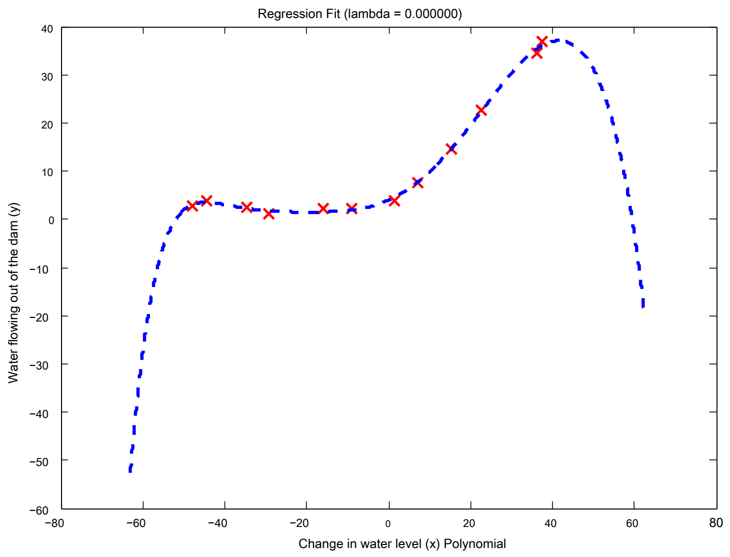


Figure 4: Polynomial fit, $\lambda = 0$

From Figure 4 , you should see that the polynomial fit is able to follow the datapoints very well - thus, obtaining a low training error. However, the

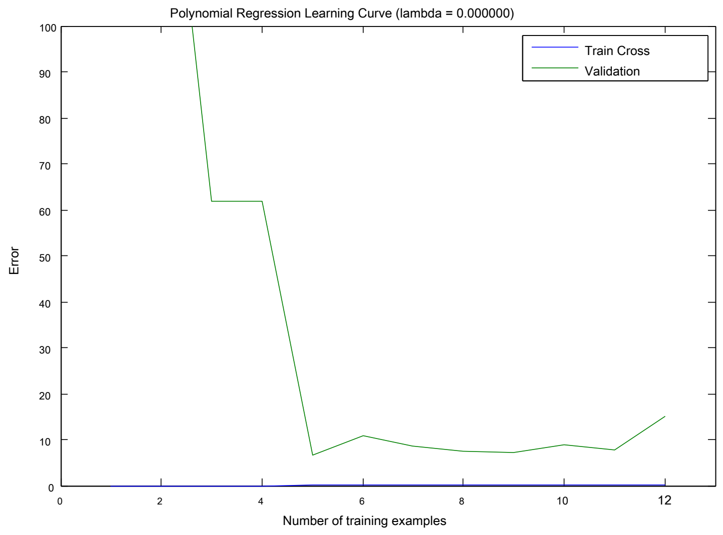


Figure 5: Polynomial learning curve, $\lambda = 0$

polynomial fit is very complex and even drops off at the extremes. This is an indicator that the polynomial regression model is overfitting the training data and will not generalize well.

To better understand the problems with the unregularized ($\lambda = 0$) model, you can see that the learning curve (Figure 5) shows the same effect where the low training error is low, but the cross validation error is high. There is a gap between the training and cross validation errors, indicating a high variance problem.

One way to combat the overfitting (high-variance) problem is to add regularization to the model. In the next section, you will get to try different λ parameters to see how regularization can lead to a better model.

3.2 Optional (ungraded) exercise: Adjusting the regularization parameter

In this section, you will get to observe how the regularization parameter affects the bias-variance of regularized polynomial regression. You should now modify the the lambda parameter in the ex5.m and try $\lambda = 1, 100$. For each of these values, the script should generate a polynomial fit to the data and also a learning curve.

For $\lambda = 1$, you should see a polynomial fit that follows the data trend well (Figure 6) and a learning curve (Figure 7) showing that both the cross validation and training error converge to a relatively low value. This shows the $\lambda = 1$ regularized polynomial regression model does not have the highbias or high-variance problems. In effect, it achieves a good trade-off between bias and variance.

For $\lambda = 100$, you should see a polynomial fit (Figure 8) that does not follow the data well. In this case, there is too much regularization and the model is unable to fit the training data.

You do not need to submit any solutions for this optional (ungraded) exercise.

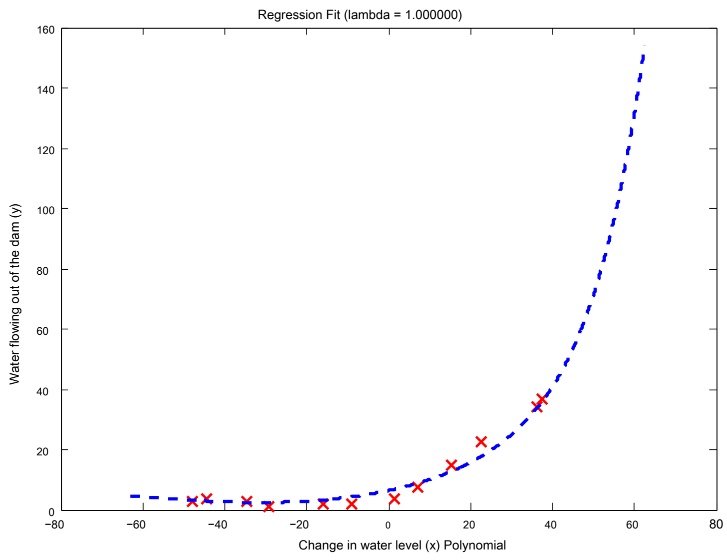


Figure 6: Polynomial fit, $\lambda = 1$

3.3 Selecting λ using a cross validation set

From the previous parts of the exercise, you observed that the value of λ can significantly affect the results of regularized polynomial regression on the training and cross validation set. In particular, a model without regularization ($\lambda = 0$) fits the training set well, but does not generalize. Conversely, a model with too much regularization ($\lambda = 100$) does not fit the training set and testing set well. A good choice of λ (e.g., $\lambda = 1$) can provide a good fit to the data.

In this section, you will implement an automated method to select the λ parameter. Concretely, you will use a cross validation set to evaluate

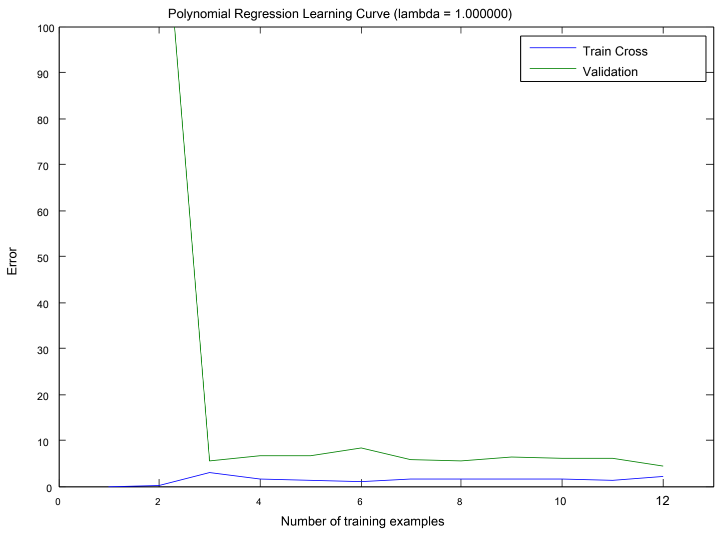


Figure 7: Polynomial learning curve, $\lambda = 1$

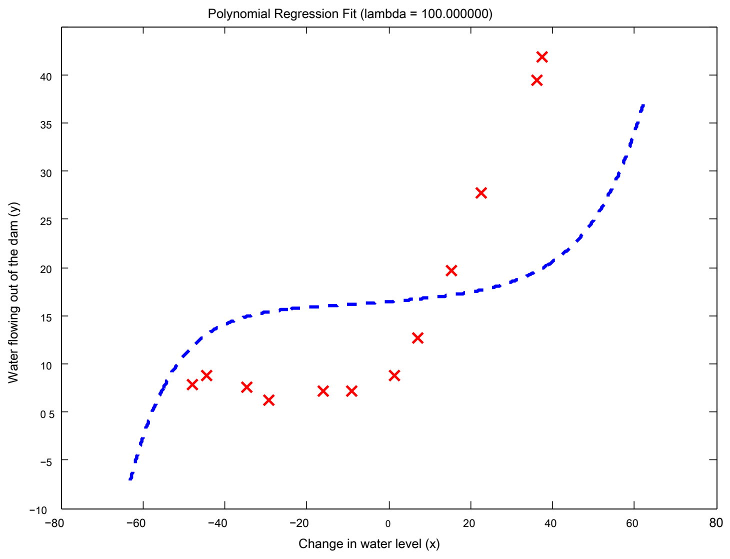


Figure 8: Polynomial fit, $\lambda = 100$

how good each λ value is. After selecting the best λ value using the cross validation set, we can then evaluate the model on the test set to estimate how well the model will perform on actual unseen data.

Your task is to complete the code in validationCurve.m. Specifically, you should should use the trainLinearReg function to train the model using different values of λ and compute the training error and cross validation error. You should try λ in the following range: {0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10}.

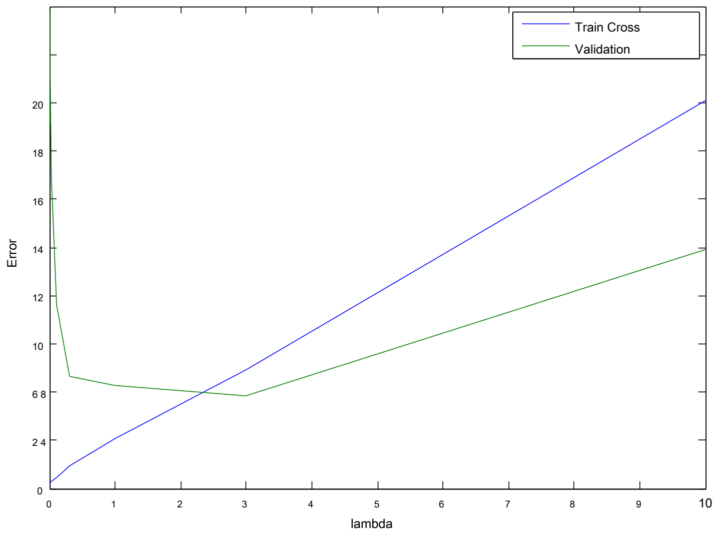


Figure 9: Selecting λ using a cross validation set

After you have completed the code, the next part of ex5.m will run your function can plot a cross validation curve of error v.s. λ that allows you select which λ parameter to use. You should see a plot similar to Figure 9 . In this figure, we can see that the best value of λ is around 3. Due to randomness in the training and validation splits of the dataset, the cross validation error can sometimes be lower than the training error.

You should now submit your validation curve function.

3.4 Optional (ungraded) exercise: Computing test set error

In the previous part of the exercise, you implemented code to compute the cross validation error for various values of the regularization parameter λ.

However, to get a better indication of the model's performance in the real world, it is important to evaluate the "final" model on a test set that was not used in any part of training (that is, it was neither used to select the λ

parameters, nor to learn the model parameters θ).

For this optional (ungraded) exercise, you should compute the test error using the best value of λ you found. In our cross validation, we obtained a test error of 3.8599 for $\lambda = 3$.

You do not need to submit any solutions for this optional (ungraded) exercise.

3.5 Optional (ungraded) exercise: Plotting learning curves with randomly selected examples

In practice, especially for small training sets, when you plot learning curves to debug your algorithms, it is often helpful to average across multiple sets of randomly selected examples to determine the training error and cross validation error.

Concretely, to determine the training error and cross validation error for i examples, you should first randomly select i examples from the training set and i examples from the cross validation set. You will then learn the parameters θ using the randomly chosen training set and evaluate the parameters θ on the randomly chosen training set and cross validation set. The above steps should then be repeated multiple times (say 50) and the averaged error should be used to determine the training error and cross validation error for i examples.

For this optional (ungraded) exercise, you should implement the above strategy for computing the learning curves. For reference, figure 10 shows the learning curve we obtained for polynomial regression with $\lambda = 0.01$. Your figure may differ slightly due to the random selection of examples.

You do not need to submit any solutions for this optional (ungraded) exercise.

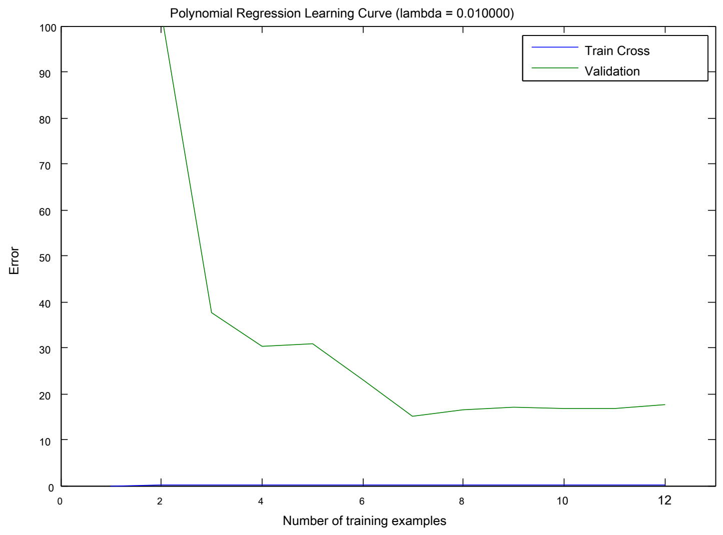


Figure 10: Optional (ungraded) exercise: Learning curve with randomly selected examples

Submission and Grading

After completing various parts of the assignment, be sure to use the submit
function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	Submitted File	Points
Regularized Linear Regression Cost Function	linearRegCostFunction.m 25 points	
Regularized Linear Regression Gradient	linearRegCostFunction.m 25 points	
Learning Curve	learningCurve.m	20 points
Polynomial Feature Mapping	polyFeatures.m	10 points
Cross Validation Curve	validationCurve.m	20 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.

Programming Exercise 6:
Support Vector Machines

Machine Learning

Introduction

In this exercise, you will be using support vector machines (SVMs) to build a spam classifier. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the cd command in Octave to change to this directory before starting this exercise.

Files included in this exercise

ex6.m - Octave script for the first half of the exercise
ex6data1.mat - Example Dataset 1
ex6data2.mat - Example Dataset 2
ex6data3.mat - Example Dataset 3
svmTrain.m - SVM rraining function
svmPredict.m - SVM prediction function
plotData.m - Plot 2D data
visualizeBoundaryLinear.m - Plot linear boundary
visualizeBoundary.m - Plot non-linear boundary
linearKernel.m - Linear kernel for SVM
[?] gaussianKernel.m - Gaussian kernel for SVM
[?] dataset3Params.m - Parameters to use for Dataset 3

ex6 spam.m - Octave script for the second half of the exercise
spamTrain.mat - Spam training set

spamTest.mat - Spam test set
emailSample1.txt - Sample email 1
emailSample2.txt - Sample email 2
spamSample1.txt - Sample spam 1
spamSample2.txt - Sample spam 2
vocab.txt - Vocabulary list
getVocabList.m - Load vocabulary list
porterStemmer.m - Stemming function
readFile.m - Reads a file into a character string
submit.m - Submission script that sends your solutions to our servers
submitWeb.m - Alternative submission script
[?] processEmail.m - Email preprocessing
[?] emailFeatures.m - Feature extraction from emails

? indicates files you will need to complete

Throughout the exercise, you will be using the script ex6.m. These scripts set up the dataset for the problems and make calls to functions that you will write. You are only required to modify functions in other files, by following the instructions in this assignment.

Where to get help

We also strongly encourage using the online Q&A Forum to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

If you run into network errors using the submit script, you can also use an online form for submitting your solutions. To use this *alternative* submission interface, run the submitWeb script to generate a submission file (e.g.,
submit ex6 part2.txt). You can then submit this file through the web submission form in the programming exercises page (go to the programming exercises page, then select the exercise you are submitting for). If you are having no problems submitting through the standard submission system using the submit script, you do *not* need to use this alternative submission interface.

1 Support Vector Machines

In the first half of this exercise, you will be using support vector machines (SVMs) with various example 2D datasets. Experimenting with these datasets will help you gain an intuition of how SVMs work and how to use a Gaussian kernel with SVMs. In the next half of the exercise, you will be using support vector machines to build a spam classifier.

The provided script, ex6.m, will help you step through the first half of the exercise.

1.1 Example Dataset 1

We will begin by with a 2D example dataset which can be separated by a linear boundary. The script ex6.m will plot the training data (Figure 1). In this dataset, the positions of the positive examples (indicated with +) and the negative examples (indicated with *o)* suggest a natural separation indicated by the gap. However, notice that there is an outlier positive example + on the far left at about (0.1, 4.1). As part of this exercise, you will also see how this outlier affects the SVM decision boundary.

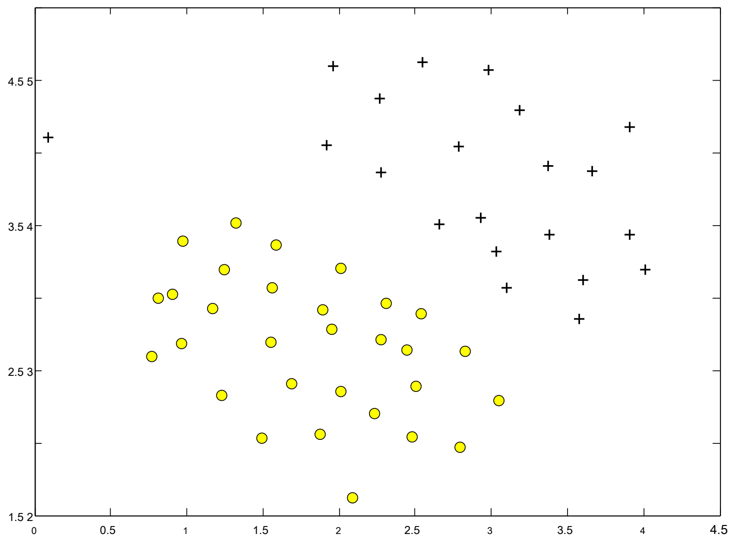


Figure 1: Example Dataset 1

In this part of the exercise, you will try using different values of the *C* parameter with SVMs. Informally, the *C* parameter is a positive value that controls the penalty for misclassified training examples. A large *C* parameter tells the SVM to try to classify all the examples correctly. *C* plays a role similar to $\frac{1}{\lambda}$, where *λ* is the regularization parameter that we were using previously for logistic regression.

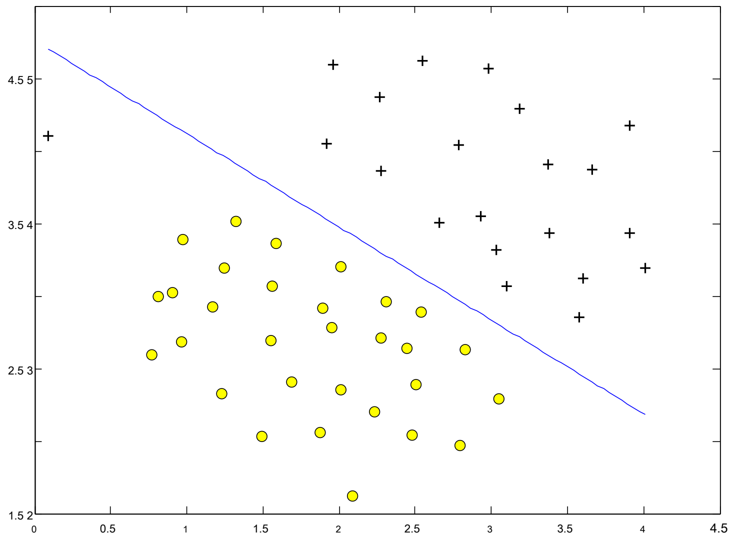


Figure 2: SVM Decision Boundary with *C* = 1 (Example Dataset 1)

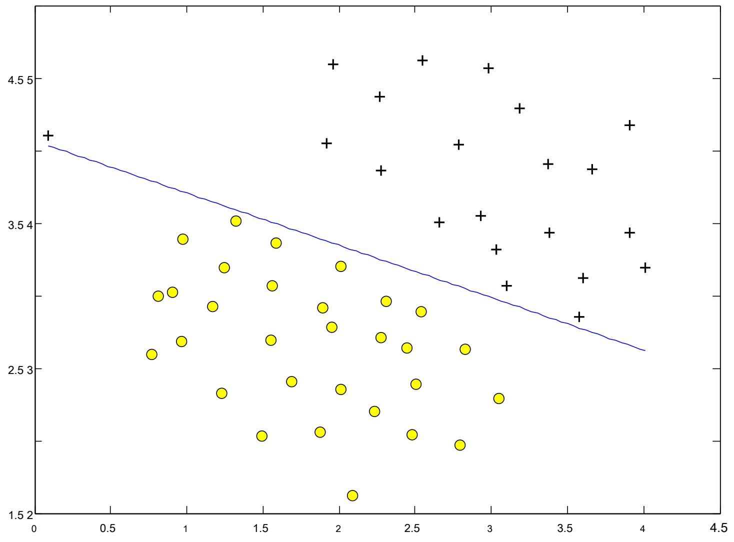


Figure 3: SVM Decision Boundary with *C* = 100 (Example Dataset 1)

The next part in ex6.m will run the SVM training (with *C* = 1) using

SVM software that we have included with the starter code, svmTrain.m. [1]
When $C = 1$, you should find that the SVM puts the decision boundary in the gap between the two datasets and *misclassifies* the data point on the far left (Figure 2).

Implementation Note: Most SVM software packages (including svmTrain.m) automatically add the extra feature $x_0 = 1$ for you and automatically take care of learning the intercept term θ_0. So when passing your training data to the SVM software, there is no need to add this extra feature $x_0 = 1$ yourself. In particular, in Octave your code should be working with training examples $x \in <_{n}$ (rather than $x \in <_{n+1}$); for example, in the first example dataset $x \in <_2$.

Your task is to try different values of C on this dataset. Specifically, you should change the value of C in the script to $C = 100$ and run the SVM training again. When $C = 100$, you should find that the SVM now classifies every single example correctly, but has a decision boundary that does not appear to be a natural fit for the data (Figure 3).

1.2 SVM with Gaussian Kernels

In this part of the exercise, you will be using SVMs to do non-linear classification. In particular, you will be using SVMs with Gaussian kernels on datasets that are not linearly separable.

1.2.1 Gaussian Kernel

To find non-linear decision boundaries with the SVM, we need to first implement a Gaussian kernel. You can think of the Gaussian kernel as a similarity function that measures the “distance” between a pair of examples, ($x^{(i)}, x^{(j)}$). The Gaussian kernel is also parameterized by a bandwidth parameter, σ, which determines how fast the similarity metric decreases (to 0) as the examples are further apart.

You should now complete the code in gaussianKernel.m to compute the Gaussian kernel between two examples, ($x^{(i)}, x^{(j)}$). The Gaussian kernel

[1] In order to ensure compatibility with Octave, we have included this implementation of an SVM learning algorithm. However, this particular implementation was chosen to maximize compatibility, and is not very efficient. If you are training an SVM on a real problem, especially if you need to scale to a larger dataset, we strongly recommend instead using a highly optimized SVM toolbox such as LIBSVM .

function is defined as:

$$K_{gaussian}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right).$$

Once you've completed the function gaussianKernel.m, the script ex6.m
will test your kernel function on two provided examples and you should expect to see a value of 0.324652.

You should now submit your function that computes the Gaussian kernel.

1.2.2 Example Dataset 2

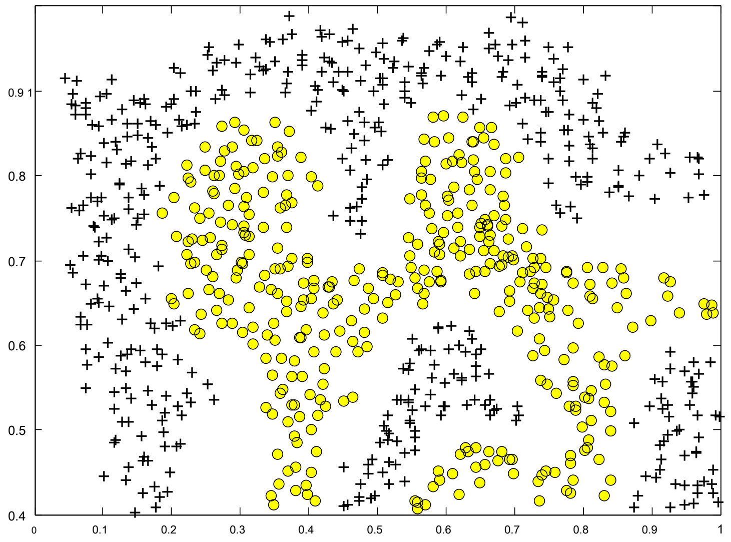


Figure 4: Example Dataset 2

The next part in ex6.m will load and plot dataset 2 (Figure 4). From
the figure, you can obserse that there is no linear decision boundary that separates the positive and negative examples for this dataset. However, by using the Gaussian kernel with the SVM, you will be able to learn a non-linear decision boundary that can perform reasonably well for the dataset.

If you have correctly implemented the Gaussian kernel function, ex6.m
will proceed to train the SVM with the Gaussian kernel on this dataset.

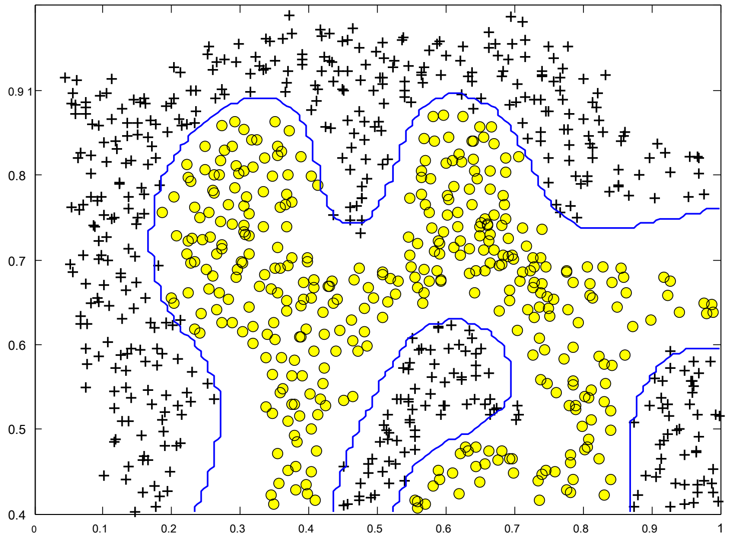


Figure 5: SVM (Gaussian Kernel) Decision Boundary (Example Dataset 2)

Figure 5 shows the decision boundary found by the SVM with a Gaussian kernel. The decision boundary is able to separate most of the positive and negative examples correctly and follows the contours of the dataset well.

1.2.3 Example Dataset 3

In this part of the exercise, you will gain more practical skills on how to use a SVM with a Gaussian kernel. The next part of ex6.m will load and display a third dataset (Figure 6). You will be using the SVM with the Gaussian kernel with this dataset.

In the provided dataset, ex6data3.mat, you are given the variables X, y, Xval, yval. The provided code in ex6.m trains the SVM classifier using the training set (X, y) using parameters loaded from dataset3Params.m.

Your task is to use the cross validation set Xval, yval to determine the best C and σ parameter to use. You should write any additional code necessary to help you search over the parameters C and σ. For *both* C and σ, we suggest trying values in multiplicative steps (e.g., 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30). Note that you should try all possible pairs of values for C and σ (e.g., $C = 0.3$ and $\sigma = 0.1$). For example, if you try each of the 8 values listed above for C

and for σ^2, you would end up training and evaluating (on the cross validation set) a total of $8^2 = 64$ different models.

After you have determined the best C and σ parameters to use, you should modify the code in dataset3Params.m, filling in the best parameters

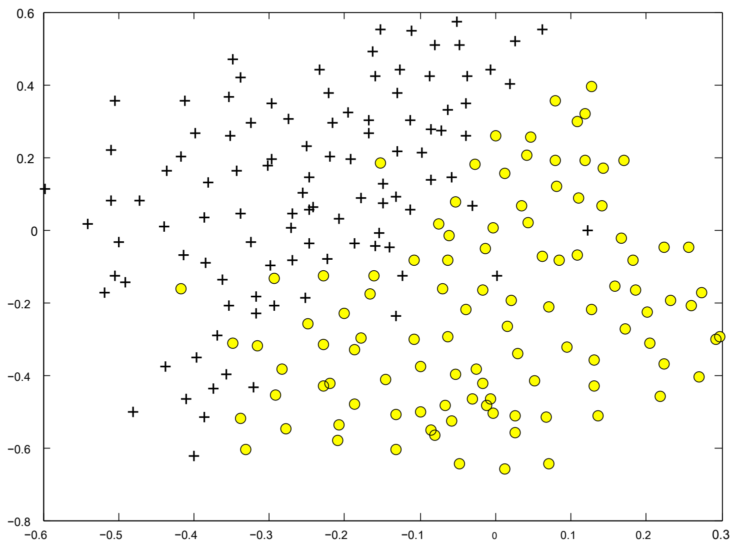


Figure 6: Example Dataset 3

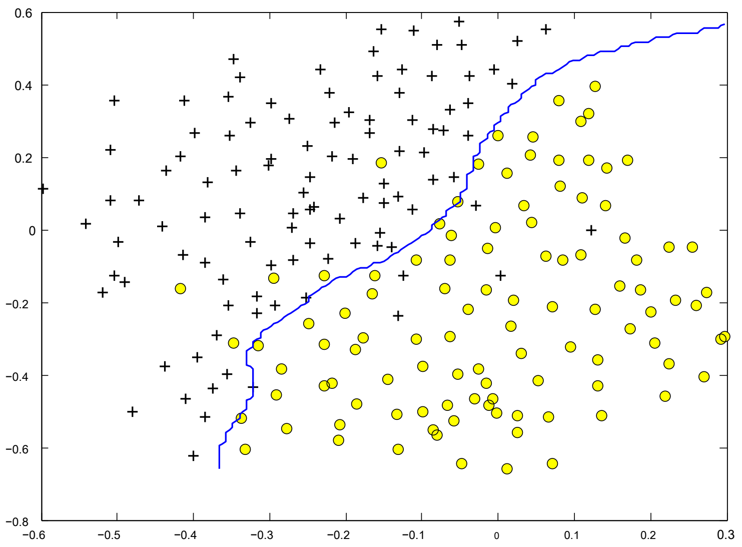


Figure 7: SVM (Gaussian Kernel) Decision Boundary (Example Dataset 3)

you found. For our best parameters, the SVM returned a decision boundary shown in Figure 7 .

Implementation Tip: When implementing cross validation to select the best C and σ parameter to use, you need to evaluate the error on the cross validation set. Recall that for classification, the error is defined as the fraction of the cross validation examples that were classified incorrectly. In Octave, you can compute this error using mean(double(predictions ~= yval)), where predictions is a vector containing all the predictions from the SVM, and yval are the true labels from the cross validation set. You can use the svmPredict function to generate the predictions for the cross validation set.

You should now submit your best C and σ values.

2 Spam Classification

Many email services today provide spam filters that are able to classify emails into spam and non-spam email with high accuracy. In this part of the exercise, you will use SVMs to build your own spam filter.

You will be training a classifier to classify whether a given email, x, is spam ($y = 1$) or non-spam ($y = 0$). In particular, you need to convert each email into a feature vector $x \in \mathbb{R}^n$. The following parts of the exercise will walk you through how such a feature vector can be constructed from an email.

Throughout the rest of this exercise, you will be using the the script ex6 spam.m. The dataset included for this exercise is based on a a subset of the SpamAssassin Public Corpus.[2] For the purpose of this exercise, you will only be using the body of the email (excluding the email headers).

2.1 Preprocessing Emails

```
> Anyone knows how much it costs to host a web portal ?
>
Well, it depends on how many visitors youre expecting. This can be anywhere from less than 10 bucks a
month to a couple of $100. You should checkout http://www.rackspace.com/ or perhaps Amazon EC2 if youre
running something big..


To unsubscribe yourself from this mailing list, send an email to: groupname-unsubscribe@egroups.com
```

Figure 8: Sample Email

Before starting on a machine learning task, it is usually insightful to take a look at examples from the dataset. Figure 8 shows a sample email that contains a URL, an email address (at the end), numbers, and dollar amounts. While many emails would contain similar types of entities (e.g., numbers, other URLs, or other email addresses), the specific entities (e.g., the specific URL or specific dollar amount) will be different in almost every email. Therefore, one method often employed in processing emails is to "normalize" these values, so that all URLs are treated the same, all numbers are treated the same, etc. For example, we could replace each URL in the email with the unique string "httpaddr" to indicate that a URL was present.

[2] http://spamassassin.apache.org/publiccorpus/

This has the effect of letting the spam classifier make a classification decision based on whether *any* URL was present, rather than whether a specific URL was present. This typically improves the performance of a spam classifier, since spammers often randomize the URLs, and thus the odds of seeing any particular URL again in a new piece of spam is very small.

In processEmail.m, we have implemented the following email preprocessing and normalization steps:

- Lower-casing: The entire email is converted into lower case, so that captialization is ignored (e.g., IndIcaTE is treated the same as Indicate).
- Stripping HTML: All HTML tags are removed from the emails. Many emails often come with HTML formatting; we remove all the HTML tags, so that only the content remains.
- Normalizing URLs: All URLs are replaced with the text " httpaddr ".
- Normalizing Email Addresses: All email addresses are replaced with the text " emailaddr ".
- Normalizing Numbers: All numbers are replaced with the text " number ".
- Normalizing Dollars: All dollar signs ($) are replaced with the text " dollar ".
- Word Stemming: Words are reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with " discount ". Sometimes, the Stemmer actually strips off additional characters from the end, so "include", "includes", "included", and "including" are all replaced with " includ ".
- Removal of non-words: Non-words and punctuation have been removed. All white spaces (tabs, newlines, spaces) have all been trimmed to a single space character.

The result of these preprocessing steps is shown in Figure 9 . While pre-processing has left word fragments and non-words, this form turns out to be much easier to work with for performing feature extraction.

```
anyon know how much it cost to host a web portal well it depend on how mani visitor your expect thi can be anywher
from less than number buck a month to a coupl of dollarnumb you should checkout httpaddr or perhap amazon
ecnumb if your run someth big to unsubscrib yourself from thi mail list send an email to emailaddr
```

Figure 9: Preprocessed Sample Email

```
1 aa 2 ab
3 abil

. . .
86 anyon
. . .
916 know
. . .
1898 zero
1899 zip
```

Figure 10: Vocabulary List

```
86 916 794 1077 883 370 1699
790 1822 1831 883 431 1171
794 1002 1893 1364 592 1676
238 162 89 688 945 1663 1120
1062 1699 375 1162 479 1893
1510 799 1182 1237 810 1895
1440 1547 181 1699 1758
1896 688 1676 992 961 1477
71 530 1699 531
```

Figure 11: Word Indices for Sample Email

2.1.1 Vocabulary List

After preprocessing the emails, we have a list of words (e.g., Figure 9) for each email. The next step is to choose which words we would like to use in our classifier and which we would want to leave out.

For this exercise, we have chosen only the most frequently occuring words as our set of words considered (the vocabulary list). Since words that occur rarely in the training set are only in a few emails, they might cause the model to overfit our training set. The complete vocabulary list is in the file
vocab.txt and also shown in Figure 10 . Our vocabulary list was selected by choosing all words which occur at least a 100 times in the spam corpus, resulting in a list of 1899 words. In practice, a vocabulary list with about
10,000 to 50,000 words is often used.

Given the vocabulary list, we can now map each word in the preprocessed emails (e.g., Figure 9) into a list of word indices that contains the index of the word in the vocabulary list. Figure 11 shows the mapping for the sample email. Specifically, in the sample email, the word "anyone" was first normalized to "anyon" and then mapped onto the index 86 in the vocabulary list.

Your task now is to complete the code in processEmail.m to perform this mapping. In the code, you are given a string str which is a single word from the processed email. You should look up the word in the vocabulary list vocabList and find if the word exists in the vocabulary list. If the word exists, you should add the index of the word into the word indices variable. If the word does not exist, and is therefore not in the vocabulary, you can skip the word.

Once you have implemented processEmail.m, the script ex6 spam.m will run your code on the email sample and you should see an output similar to Figures 9 & 11 .

Octave Tip: In Octave, you can compare two strings with the strcmp function. For example, strcmp(str1, str2) will return 1 only when both strings are equal. In the provided starter code, vocabList is a “cellarray” containing the words in the vocabulary. In Octave, a cell-array is just like a normal array (i.e., a vector), except that its elements can also be strings (which they can’t in a normal Octave matrix/vector), and you index into them using curly braces instead of square brackets. Specifically, to get the word at index i, you can use vocabList{i}. You can also use

length(vocabList) to get the number of words in the vocabulary.

You should now submit the email preprocessing function.

2.2 Extracting Features from Emails

You will now implement the feature extraction that converts each email into a vector in R^n. For this exercise, you will be using $n = \#$ words in vocabulary list. Specifically, the feature $x_i \in \{0, 1\}$ for an email corresponds to whether the i-th word in the dictionary occurs in the email. That is, $x_i = 1$ if the i-th word is in the email and $x_i = 0$ if the i-th word is not present in the email.

Thus, for a typical email, this feature would look like:

$$x = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n.$$

You should now complete the code in emailFeatures.m to generate a feature vector for an email, given the word indices.

Once you have implemented emailFeatures.m, the next part of ex6 spam.m will run your code on the email sample. You should see that the feature vector had length 1899 and 45 non-zero entries.

You should now submit the email feature extraction function.

2.3 Training SVM for Spam Classification

After you have completed the feature extraction functions, the next step of ex6 spam.m will load a preprocessed training dataset that will be used to train a SVM classifier. spamTrain.mat contains 4000 training examples of spam and non-spam email, while spamTest.mat contains 1000 test examples. Each original email was processed using the processEmail and emailFeatures functions and converted into a vector $x^{(i)} \in \mathbb{R}^{1899}$.

After loading the dataset, ex6 spam.m will proceed to train a SVM to classify between spam ($y = 1$) and non-spam ($y = 0$) emails. Once the training completes, you should see that the classifier gets a training accuracy of about 99.8% and a test accuracy of about 98.5%.

2.4 Top Predictors for Spam

```
our click remov guarante visit basenumb dollar will price pleas nbsp most lo ga dollarnumb
```

Figure 12: Top predictors for spam email

To better understand how the spam classifier works, we can inspect the parameters to see which words the classifier thinks are the most predictive of spam. The next step of ex6 spam.m finds the parameters with the largest positive values in the classifier and displays the corresponding words (Figure 12). Thus, if an email contains words such as "guarantee", "remove", "dollar", and "price" (the top predictors shown in Figure 12), it is likely to be classified as spam.

2.5 Optional (ungraded) exercise: Try your own emails

Now that you have trained a spam classifier, you can start trying it out on your own emails. In the starter code, we have included two email examples (emailSample1.txt and emailSample2.txt) and two spam examples (spamSample1.txt and spamSample2.txt). The last part of ex6 spam.m

runs the spam classifier over the first spam example and classifies it using the learned SVM. You should now try the other examples we have provided and see if the classifier gets them right. You can also try your own emails by replacing the examples (plain text files) with your own emails.

You do not need to submit any solutions for this optional (ungraded) exercise.

2.6 Optional (ungraded) exercise: Build your own dataset

In this exercise, we provided a preprocessed training set and test set. These datasets were created using the same functions (processEmail.m and emailFeatures.m) that you now have completed. For this optional (ungraded) exercise, you will build your own dataset using the original emails from the SpamAssassin Public Corpus .

Your task in this optional (ungraded) exercise is to download the original files from the public corpus and extract them. After extracting them, you should run the processEmail[3] and emailFeatures functions on each email to extract a feature vector from each email. This will allow you to build a dataset X, y of examples. You should then randomly divide up the dataset into a training set, a cross validation set and a test set.

While you are building your own dataset, we also encourage you to try building your own vocabulary list (by selecting the high frequency words

[3] The original emails will have email headers that you might wish to leave out. We have included code in processEmail that will help you remove these headers.

that occur in the dataset) and adding any additional features that you think might be useful.

Finally, we also suggest trying to use highly optimized SVM toolboxes such as LIBSVM .

You do not need to submit any solutions for this optional (ungraded) exercise.

Submission and Grading

After completing various parts of the assignment, be sure to use the submit function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	Submitted File	Points
Gaussian Kernel	gaussianKernel.m	25 points
Parameters (*C, σ)* for Dataset 3	dataset3Params.m	25 points
Email Preprocessing	processEmail.m	25 points
Email Feature Extraction	emailFeatures.m	25 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.

Programming Exercise 7:
K- means Clustering and Principal Component Analysis

Machine Learning

Introduction

In this exercise, you will implement the *K*- means clustering algorithm and apply it to compress an image. In the second part, you will use principal component analysis to find a low-dimensional representation of face images. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the cd command in Octave to change to this directory before starting this exercise.

Files included in this exercise

ex7.m - Octave/Matlab script for the first exercise on *K*- means
ex7 pca.m - Octave/Matlab script for the second exercise on PCA
ex7data1.mat - Example Dataset for PCA
ex7data2.mat - Example Dataset for *K*- means
ex7faces.mat - Faces Dataset
bird small.png - Example Image
displayData.m - Displays 2D data stored in a matrix
drawLine.m - Draws a line over an exsiting figure
plotDataPoints.m - Initialization for *K*- means centroids
plotProgresskMeans.m - Plots each step of *K*- means as it proceeds
runkMeans.m - Runs the *K*- means algorithm
[?] pca.m - Perform principal component analysis

[?] projectData.m - Projects a data set into a lower dimensional space
[?] recoverData.m - Recovers the original data from the projection
[?] findClosestCentroids.m - Find closest centroids (used in *K*- means)
[?] computeCentroids.m - Compute centroid means (used in *K*- means)
[?] kMeansInitCentroids.m - Initialization for *K*- means centroids

? indicates files you will need to complete

Throughout the first part of the exercise, you will be using the script
ex7.m, for the second part you will use ex7 pca.m. These scripts set up the dataset for the problems and make calls to functions that you will write. You are only required to modify functions in other files, by following the instructions in this assignment.

Where to get help

We also strongly encourage using the online Q&A Forum to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

If you run into network errors using the submit script, you can also use an online form for submitting your solutions. To use this *alternative* submission interface, run the submitWeb script to generate a submission file (e.g.,
submit ex7 part2.txt). You can then submit this file through the web submission form in the programming exercises page (go to the programming exercises page, then select the exercise you are submitting for). If you are having no problems submitting through the standard submission system using the submit script, you do *not* need to use this alternative submission interface.

---

1 *K*- means Clustering

In this this exercise, you will implement the *K*- means algorithm and use it for image compression. You will first start on an example 2D dataset that will help you gain an intuition of how the *K*- means algorithm works. After that, you wil use the *K*- means algorithm for image compression by reducing the number of colors that occur in an image to only those that are most common in that image. You will be using ex7.m for this part of the exercise.

1.1 Implementing *K*- means

The *K*- means algorithm is a method to automatically cluster similar data examples together. Concretely, you are given a training set $\{ x^{(1)}, ..., x^{(m)} \}$ (where $x^{(i)} \in \mathbb{R}^n$), and want to group the data into a few cohesive "clusters". The intuition behind *K*- means is an iterative procedure that starts by guessing the initial centroids, and then refines this guess by repeatedly assigning examples to their closest centroids and then recomputing the centroids based on the assignments.

The *K*- means algorithm is as follows:

```
% Initialize centroids
centroids = kMeansInitCentroids(X, K);
for iter = 1:iterations
      % Cluster assignment step: Assign each data point to the % closest centroid. idx(i) corresponds to cˆ(i), the
      index % of the centroid assigned to example i

      idx = findClosestCentroids(X, centroids);

      % Move centroid step: Compute means based on centroid % assignments

      centroids = computeMeans(X, idx, K);
end
```

The inner-loop of the algorithm repeatedly carries out two steps: (i) Assigning each training example $x^{(i)}$ to its closest centroid, and (ii) Recomputing the mean of each centroid using the points assigned to it. The *K*- means algorithm will always converge to some final set of means for the centroids. Note that the converged solution may not always be ideal and depends on the initial setting of the centroids. Therefore, in practice the *K*- means algorithm is usually run a few times with different random initializations. One way to choose between these different solutions from different random initializations is to choose the one with the lowest cost function value (distortion).

You will implement the two phases of the *K*- means algorithm separately in the next sections.

1.1.1 Finding closest centroids

In the "cluster assignment" phase of the *K*- means algorithm, the algorithm assigns every training example $x^{(i)}$ to its closest centroid, given the current positions of centroids. Specifically, for every example *i* we set

$$c^{(i)} := j \text{ that minimizes } || x^{(i)} - \mu_j ||^2,$$

3

where $c^{(i)}$ is the index of the centroid that is closest to $x^{(i)}$, and μ_j is the position (value) of the j'th centroid. Note that $c^{(i)}$ corresponds to idx(i) in the starter code.

Your task is to complete the code in findClosestCentroids.m. This function takes the data matrix X and the locations of all centroids inside
centroids and should output a one-dimensional array idx that holds the index (a value in $\{1, ..., K\}$, where K is total number of centroids) of the closest centroid to every training example.

You can implement this using a loop over every training example and every centroid.

Once you have completed the code in findClosestCentroids.m, the script ex7.m will run your code and you should see the output [1 3 2]
corresponding to the centroid assignments for the first 3 examples.

You should now submit your "finding closest centroids" function.

1.1.2 Computing centroid means

Given assignments of every point to a centroid, the second phase of the algorithm recomputes, for each centroid, the mean of the points that were assigned to it. Specifically, for every centroid k we set

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

where C_k is the set of examples that are assigned to centroid k. Concretely, if two examples say $x^{(3)}$ and $x^{(5)}$ are assigned to centroid $k = 2$, then you should update $\mu_2 = \frac{1}{2}(x^{(3)} + x^{(5)})$.

You should now complete the code in computeCentroids.m. You can implement this function using a loop over the centroids. You can also use a loop over the examples; but if you can use a vectorized implementation that does not use such a loop, your code may run faster.

Once you have completed the code in computeCentroids.m, the script ex7.m will run your code and output the centroids after the first step of K-means.

You should now submit your compute centroids function.

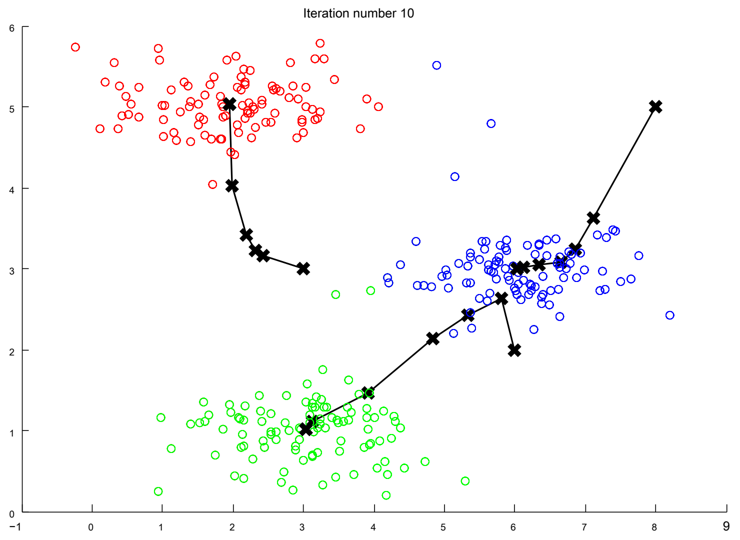


Figure 1: The expected output.

1.2 *K*- means on example dataset

After you have completed the two functions (findClosestCentroids and computeCentroids), the next step in ex7.m will run the *K*- means algorithm on a toy 2D dataset to help you understand how *K*- means works. Your functions are called from inside the runKmeans.m script. We encourage you to take a look at the function to understand how it works. Notice that the code calls the two functions you implemented in a loop.

When you run the next step, the *K*- means code will produce a visualization that steps you through the progress of the algorithm at each iteration. Press *enter* multiple times to see how each step of the *K*- means algorithm changes the centroids and cluster assignments. At the end, your figure should look as the one displayed in Figure 1 .

1.3 Random initialization

The initial assignments of centroids for the example dataset in ex7.m were designed so that you will see the same figure as in Figure 1 . In practice, a good strategy for initializing the centroids is to select random examples from the training set.

In this part of the exercise, you should complete the function kMeansInitCentroids.m with the following code:

```
% Initialize the centroids to be random examples

% Randomly reorder the indices of examples
randidx = randperm(size(X, 1));
% Take the first K examples as centroids
centroids = X(randidx(1:K), :);
```

The code above first randomly permutes the indices of the examples (using randperm). Then, it selects the first K examples based on the random permutation of the indices. This allows the examples to be selected at random without the risk of selecting the same example twice.

You do not need to make any submissions for this part of the exercise.

1.4 Image compression with K- means

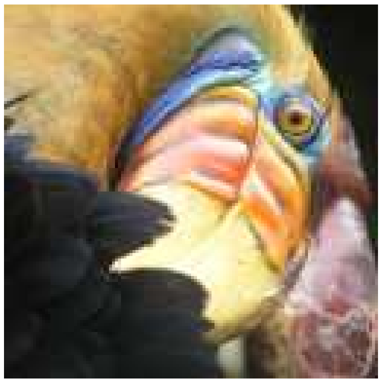


Figure 2: The original 128x128 image.

In this exercise, you will apply K- means to image compression. In a straightforward 24-bit color representation of an image, [1] each pixel is represented as three 8-bit unsigned integers (ranging from 0 to 255) that specify the red, green and blue intensity values. This encoding is often refered to as the RGB encoding. Our image contains thousands of colors, and in this part of the exercise, you will reduce the number of colors to 16 colors.

By making this reduction, it is possible to represent (compress) the photo in an efficient way. Specifically, you only need to store the RGB values of

[1] The provided photo used in this exercise belongs to Frank Wouters and is used with his permission.

the 16 selected colors, and for each pixel in the image you now need to only store the index of the color at that location (where only 4 bits are necessary to represent 16 possibilities).

In this exercise, you will use the *K*- means algorithm to select the 16 colors that will be used to represent the compressed image. Concretely, you will treat every pixel in the original image as a data example and use the *K*- means algorithm to find the 16 colors that best group (cluster) the pixels in the 3dimensional RGB space. Once you have computed the cluster centroids on the image, you will then use the 16 colors to replace the pixels in the original image.

1.4.1 *K*- means on pixels

In Matlab and Octave, images can be read in as follows:

```
% Load 128x128 color image (bird small.png)                    _
A = imread( 'bird small.png' ); _

% You will need to have installed the image package to used % imread. If you do not have the image package
installed, you % should instead change the following line to % %



        load('bird small.mat'); % Loads the image into the variable A
```

This creates a three-dimensional matrix A whose first two indices identify a pixel position and whose last index represents red, green, or blue. For example, A(50, 33, 3) gives the blue intensity of the pixel at row 50 and column 33.

The code inside ex7.m first loads the image, and then reshapes it to create an $m \times 3$ matrix of pixel colors (where $m = 16384 = 128 \times 128$), and calls your *K*- means function on it.

After finding the top $K = 16$ colors to represent the image, you can now assign each pixel position to its closest centroid using the findClosestCentroids
function. This allows you to represent the original image using the centroid assignments of each pixel. Notice that you have significantly reduced the number of bits that are required to describe the image. The original image required 24 bits for each one of the 128×128 pixel locations, resulting in total size of $128 \times 128 \times 24 = 393,216$ bits. The new representation requires some overhead storage in form of a dictionary of 16 colors, each of which require 24 bits, but the image itself then only requires 4 bits per pixel location. The final number of bits used is therefore $16 \times 24 + 128 \times 128 \times 4 = 65,920$ bits, which corresponds to compressing the original image by about a factor of 6.

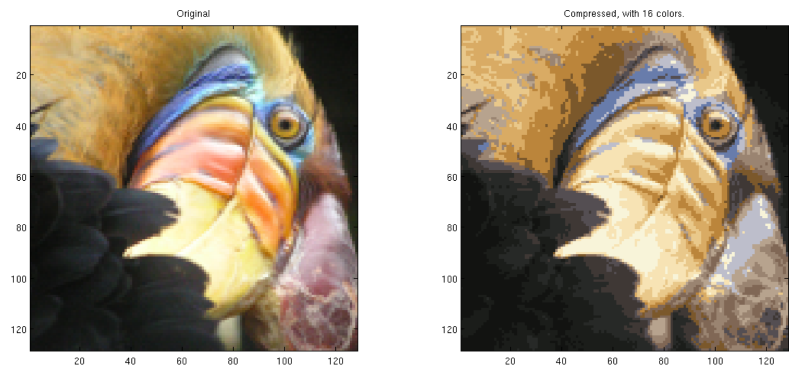


Figure 3: Original and reconstructed image (when using *K*- means to compress the image).

Finally, you can view the effects of the compression by reconstructing the image based only on the centroid assignments. Specifically, you can replace each pixel location with the mean of the centroid assigned to it. Figure 3
shows the reconstruction we obtained. Even though the resulting image retains most of the characteristics of the original, we also see some compression artifacts.

You do not need to make any submissions for this part of the exercise.

1.5 Optional (ungraded) exercise: Use your own image

In this exercise, modify the code we have supplied to run on one of your own images. Note that if your image is very large, then *K*- means can take a long time to run. Therefore, we recommend that you resize your images to managable sizes before running the code. You can also try to vary *K* to see the effects on the compression.

2 Principal Component Analysis

In this exercise, you will use principal component analysis (PCA) to perform dimensionality reduction. You will first experiment with an example 2D dataset to get intuition on how PCA works, and then use it on a bigger dataset of 5000 face image dataset.

The provided script, ex7 pca.m, will help you step through the first half of the exercise.

2.1 Example Dataset

To help you understand how PCA works, you will first start with a 2D dataset which has one direction of large variation and one of smaller variation. The script ex7 pca.m will plot the training data (Figure 4). In this part of the exercise, you will visualize what happens when you use PCA to reduce the data from 2D to 1D. In practice, you might want to reduce data from 256 to 50 dimensions, say; but using lower dimensional data in this example allows us to visualize the algorithms better.

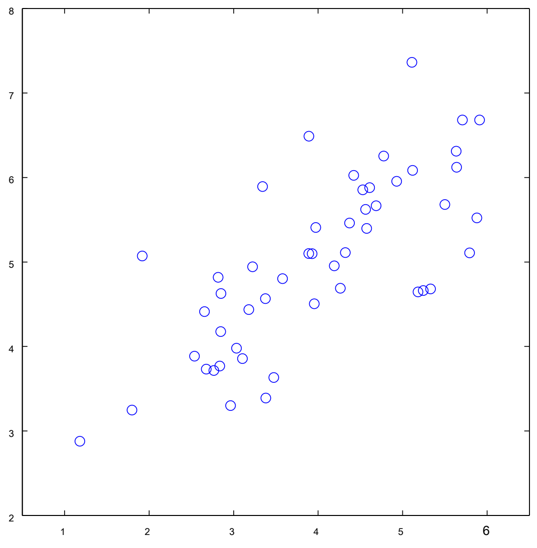


Figure 4: Example Dataset 1

2.2 Implementing PCA

In this part of the exercise, you will implement PCA. PCA consists of two computational steps: First, you compute the covariance matrix of the

data. Then, you use Octave's SVD function to compute the eigenvectors
$U_1, U_2, \ldots, U_n$. These will correspond to the principal components of variation in the data.

Before using PCA, it is important to first normalize the data by subtracting the mean value of each feature from the dataset, and scaling each dimension so that they are in the same range. In the provided script ex7 pca.m,
this normalization has been performed for you using the featureNormalize
function.

After normalizing the data, you can run PCA to compute the principal components. You task is to complete the code in pca.m to compute the principal components of the dataset. First, you should compute the covariance matrix of the data, which is given by:

$$\Sigma = \frac{1}{m} X^T X$$

where X is the data matrix with examples in rows, and m is the number of examples. Note that Σ is a $n \times n$ matrix and not the summation operator.

After computing the covariance matrix, you can run SVD on it to compute the principal components. In Octave, you can run SVD with the following command: [U, S, V] = svd(Sigma), where U will contain the principal components and S will contain a diagonal matrix.

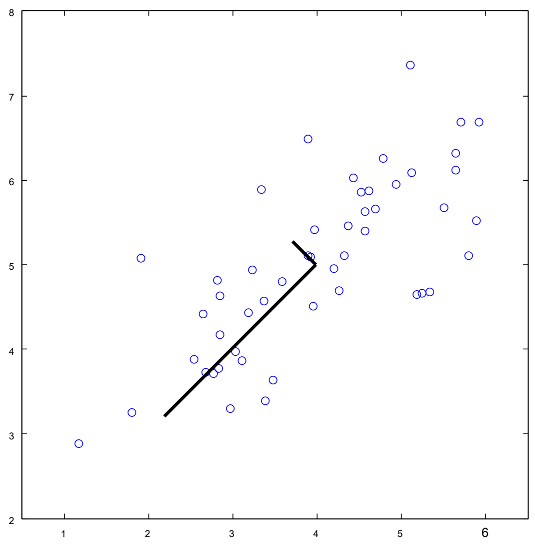


Figure 5: Computed eigenvectors of the dataset

Once you have completed pca.m, the ex7 pca.m script will run PCA on the example dataset and plot the corresponding principal components found

(Figure 5). The script will also output the top principal component (eigenvector) found, and you should expect to see an output of about [- 0.707
- 0.707]. (It is possible that Octave may instead output the negative of this, since U_1 and $-U_1$ are equally valid choices for the first principal component.)

You should now submit your PCA function.

2.3 Dimensionality Reduction with PCA

After computing the principal components, you can use them to reduce the feature dimension of your dataset by projecting each example onto a lower dimensional space, $x^{(i)} \rightarrow z^{(i)}$ (e.g., projecting the data from 2D to 1D). In this part of the exercise, you will use the eigenvectors returned by PCA and project the example dataset into a 1-dimensional space.

In practice, if you were using a learning algorithm such as linear regression or perhaps neural networks, you could now use the projected data instead of the original data. By using the projected data, you can train your model faster as there are less dimensions in the input.

2.3.1 Projecting the data onto the principal components

You should now complete the code in projectData.m. Specifically, you are given a dataset X, the principal components U, and the desired number of dimensions to reduce to K. You should project each example in X onto the top K components in U. Note that the top K components in U are given by the first K columns of U, that is U reduce = U(:, 1:K).

Once you have completed the code in projectData.m, ex7 pca.m will project the first example onto the first dimension and you should see a value of about 1.481 (or possibly - 1.481, if you got $-U_1$ instead of U_1).

You should now submit the project data function.

2.3.2 Reconstructing an approximation of the data

After projecting the data onto the lower dimensional space, you can approximately recover the data by projecting them back onto the original high dimensional space. Your task is to complete recoverData.m to project each example in Z back onto the original space and return the recovered approximation in X rec.

Once you have completed the code in projectData.m, ex7 pca.m will recover an approximation of the first example and you should see a value of about [- 1.047 -1.047].

You should now submit the recover data function.

2.3.3 Visualizing the projections

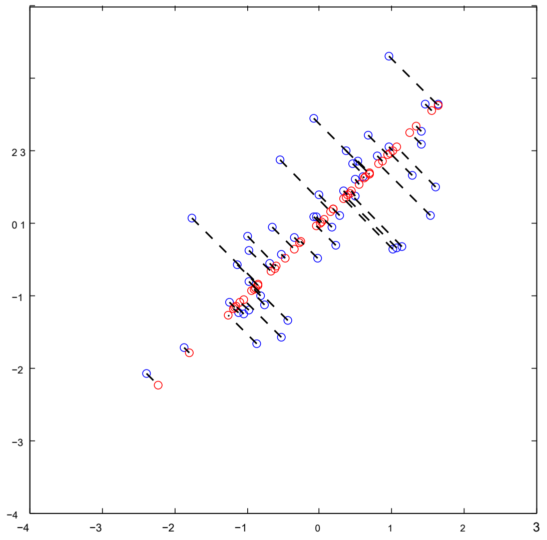


Figure 6: The normalized and projected data after PCA.

After completing both projectData and recoverData, ex7 pca.m will now perform both the projection and approximate reconstruction to show how the projection affects the data. In Figure 6 , the original data points are indicated with the blue circles, while the projected data points are indicated with the red circles. The projection effectively only retains the information in the direction given by U_1.

2.4 Face Image Dataset

In this part of the exercise, you will run PCA on face images to see how it can be used in practice for dimension reduction. The dataset ex7faces.mat
contains a dataset[2] X of face images, each 32 × 32 in grayscale. Each row of X corresponds to one face image (a row vector of length 1024). The next

[2] This dataset was based on a cropped version of the labeled faces in the wild dataset.

step in ex7 pca.m will load and visualize the first 100 of these face images (Figure 7).

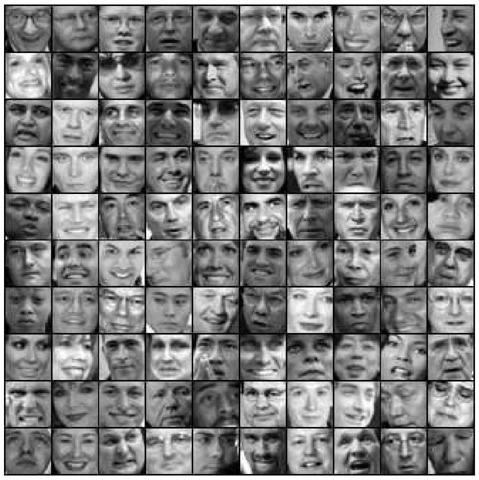


Figure 7: Faces dataset

2.4.1 PCA on Faces

To run PCA on the face dataset, we first normalize the dataset by subtracting the mean of each feature from the data matrix X. The script ex7 pca.m will do this for you and then run your PCA code. After running PCA, you will obtain the principal components of the dataset. Notice that each principal component in U (each row) is a vector of length n (where for the face dataset,

$n = 1024$). It turns out that we can visualize these principal components by reshaping each of them into a 32×32 matrix that corresponds to the pixels in the original dataset. The script ex7 pca.m displays the first 36 principal components that describe the largest variations (Figure 8). If you want, you can also change the code to display more principal components to see how they capture more and more details.

2.4.2 Dimensionality Reduction

Now that you have computed the principal components for the face dataset, you can use it to reduce the dimension of the face dataset. This allows you to use your learning algorithm with a smaller input size (e.g., 100 dimensions) instead of the original 1024 dimensions. This can help speed up your learning algorithm.

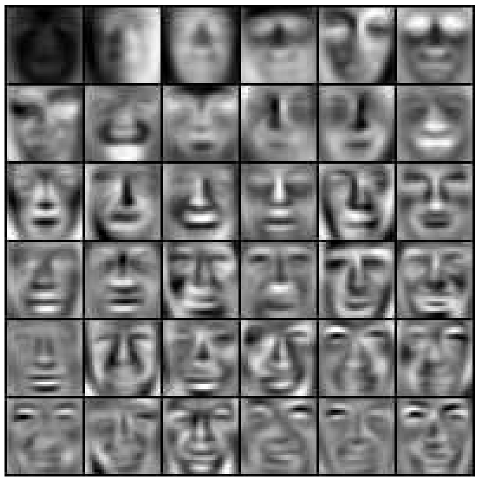


Figure 8: Principal components on the face dataset

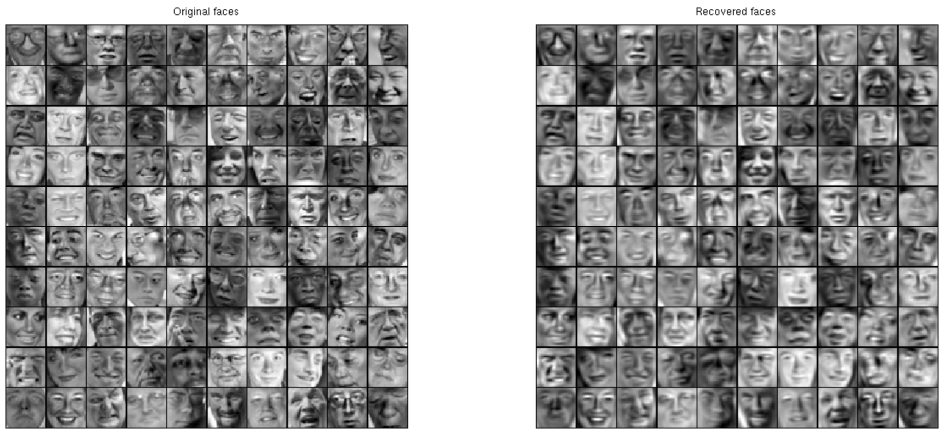


Figure 9: Original images of faces and ones reconstructed from only the top 100 principal components.

The next part in ex7 pca.m will project the face dataset onto only the first 100 principal components. Concretely, each face image is now described by a vector $z^{(i)} \in \mathbb{R}^{100}$.

To understand what is lost in the dimension reduction, you can recover the data using only the projected dataset. In ex7 pca.m, an approximate recovery of the data is performed and the original and projected face images are displayed side by side (Figure 9). From the reconstruction, you can observe that the general structure and appearance of the face are kept while the fine details are lost. This is a remarkable reduction (more than 10 $\times$) in the dataset size that can help speed up your learning algorithm significantly. For example, if you were training a neural network to perform person recognition (gven a face image, predict the identitfy of the person), you can use the dimension reduced input of only a 100 dimensions instead of the original pixels.

2.5 Optional (ungraded) exercise: PCA for visualization

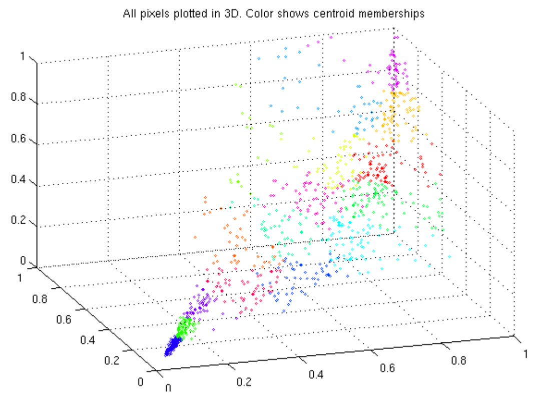


Figure 10: Original data in 3D

In the earlier *K*- means image compression exercise, you used the *K*- means algorithm in the 3-dimensional RGB space. In the last part of the ex7 pca.m
_
script, we have provided code to visualize the final pixel assignments in this 3D space using the scatter3 function. Each data point is colored according to the cluster it has been assigned to. You can drag your mouse on the figure to rotate and inspect this data in 3 dimensions.

It turns out that visualizing datasets in 3 dimensions or greater can be cumbersome. Therefore, it is often desirable to only display the data in 2D even at the cost of losing some information. In practice, PCA is often used to reduce the dimensionality of data for visualization purposes. In the next part of ex7 pca.m, the script will apply your implementation of PCA to the 3dimensional data to reduce it to 2 dimensions and visualize the result in a 2D scatter plot. The PCA projection can be thought of as a rotation that selects the view that maximizes the spread of the data, which often corresponds to the “best” view.

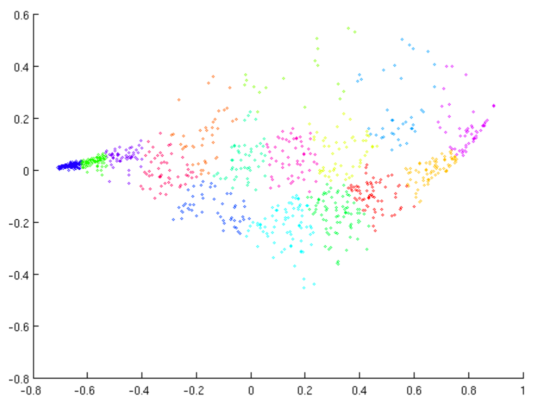


Figure 11: 2D visualization produced using PCA

Submission and Grading

After completing various parts of the assignment, be sure to use the submit
function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	Submitted File	Points
Find Closest Centroids	findClosestCentroids.m 30 points	
Compute Centroid Means	computeCentroids.m	30 points
PCA	pca.m	20 points
Project Data	projectData.m	10 points
Recover Data	recoverData.m	10 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.

Programming Exercise 8: Anomaly Detection and Recommender Systems

Machine Learning

Introduction

In this exercise, you will implement the anomaly detection algorithm and apply it to detect failing servers on a network. In the second part, you will use collaborative filtering to build a recommender system for movies. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the cd command in Octave to change to this directory before starting this exercise.

Files included in this exercise

ex8.m - Octave/Matlab script for first part of exercise
ex8 cofi.m - Octave/Matlab script for second part of exercise
ex8data1.mat - First example Dataset for anomaly detection
ex8data2.mat - Second example Dataset for anomaly detection
ex8 movies.mat - Movie Review Dataset
ex8 movieParams.mat - Parameters provided for debugging
multivariateGaussian.m - Computes the probability density function for a Gaussian distribution

visualizeFit.m - 2D plot of a Gaussian distribution and a dataset
checkCostFunction.m - Gradient checking for collaborative filtering
computeNumericalGradient.m - Numerically compute gradients
fmincg.m - Function minimization routine (similar to fminunc)
loadMovieList.m - Loads the list of movies into a cell-array

movie ids.txt - List of movies
normalizeRatings.m - Mean normalization for collaborative filtering
[?] estimateGaussian.m - Estimate the parameters of a Gaussian distribution with a diagonal covariance matrix
[?] selectThreshold.m - Find a threshold for anomaly detection
[?] cofiCostFunc.m - Implement the cost function for collaborative filtering

? indicates files you will need to complete

Throughout the first part of the exercise (anomaly detection) you will be using the script ex8.m. For the second part of collaborative filtering, you will use ex8 cofi.m. These scripts set up the dataset for the problems and make calls to functions that you will write. You are only required to modify functions in other files, by following the instructions in this assignment.

Where to get help

We also strongly encourage using the online Q&A Forum to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

If you run into network errors using the submit script, you can also use an online form for submitting your solutions. To use this *alternative* submission interface, run the submitWeb script to generate a submission file (e.g.,
submit ex8 part2.txt). You can then submit this file through the web submission form in the programming exercises page (go to the programming exercises page, then select the exercise you are submitting for). If you are having no problems submitting through the standard submission system using the submit script, you do *not* need to use this alternative submission interface.

---

1 Anomaly detection

In this exercise, you will implement an anomaly detection algorithm to detect anomalous behavior in server computers. The features measure the throughput (mb/s) and latency (ms) of response of each server. While your servers were operating, you collected $m = 307$ examples of how they were behaving, and thus have an unlabeled dataset $\{x^{(1)}, \ldots, x^{(m)}\}$. You suspect that the vast majority of these examples are "normal" (non-anomalous) examples of the servers operating normally, but there might also be some examples of servers acting anomalously within this dataset.

You will use a Gaussian model to detect anomalous examples in your dataset. You will first start on a 2D dataset that will allow you to visualize what the algorithm is doing. On that dataset you will fit a Gaussian distribution and then find values that have very low probability and hence can be considered anomalies. After that, you will apply the anomaly detection algorithm to a larger dataset with many dimensions. You will be using ex8.m

for this part of the exercise.

The first part of ex8.m will visualize the dataset as shown in Figure 1 .

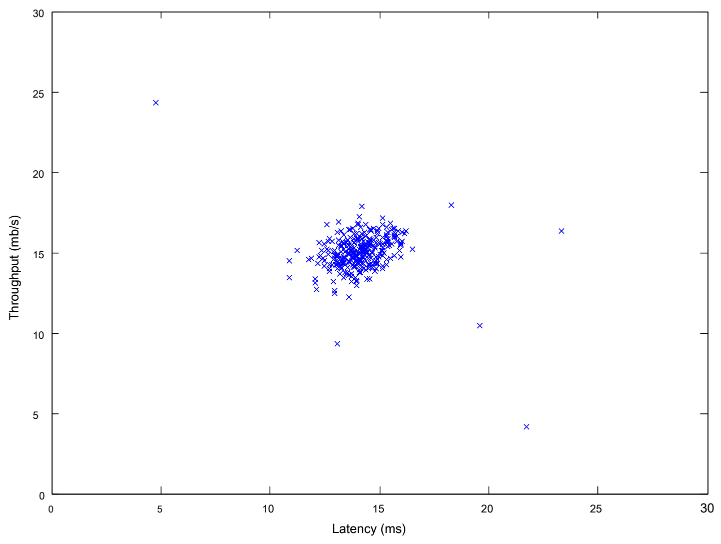


Figure 1: The first dataset.

1.1 Gaussian distribution

To perform anomaly detection, you will first need to fit a model to the data's distribution.

Given a training set $\{x^{(1)}, \ldots, x^{(m)}\}$ (where $x^{(i)} \in \mathbb{R}^n$), you want to estimate the Gaussian distribution for each of the features x_i. For each feature
$i = 1 \ldots n$, you need to find parameters μ_i and σ^2_i that fit the data in the
i-th dimension $\{x^{(1)}_i, \ldots, x^{(m)}_i\}$ (the i-th dimension of each example).

The Gaussian distribution is given by

$$p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ^2 controls the variance.

1.2 Estimating parameters for a Gaussian

You can estimate the parameters, (μ_i, σ_i^2), of the i-th feature by using the following equations. To estimate the mean, you will use:

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_i^{(j)} \qquad (1)$$

and for the variance you will use:

$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m} (x_i^{(j)} - \mu_i)^2. \qquad (2)$$

Your task is to complete the code in estimateGaussian.m. This function takes as input the data matrix X and should output an n-dimension vector mu that holds the mean of all the n features and another n-dimension vector sigma2 that holds the variances of all the features. You can implement this using a for-loop over every feature and every training example (though a vectorized implementation might be more efficient; feel free to use a vectorized implementation if you prefer). Note that in Octave, the var function will (by default) use $\frac{1}{m-1}$, instead of $\frac{1}{m}$, when computing σ_i^2.

Once you have completed the code in estimateGaussian.m, the next part of ex8.m will visualize the contours of the fitted Gaussian distribution. You should get a plot similar to Figure 2 . From your plot, you can see that most of the examples are in the region with the highest probability, while the anomalous examples are in the regions with lower probabilities.

You should now submit your estimate Gaussian parameters function.

1.3 Selecting the threshold, ε

Now that you have estimated the Gaussian parameters, you can investigate which examples have a very high probability given this distribution and which examples have a very low probability. The low probability examples are

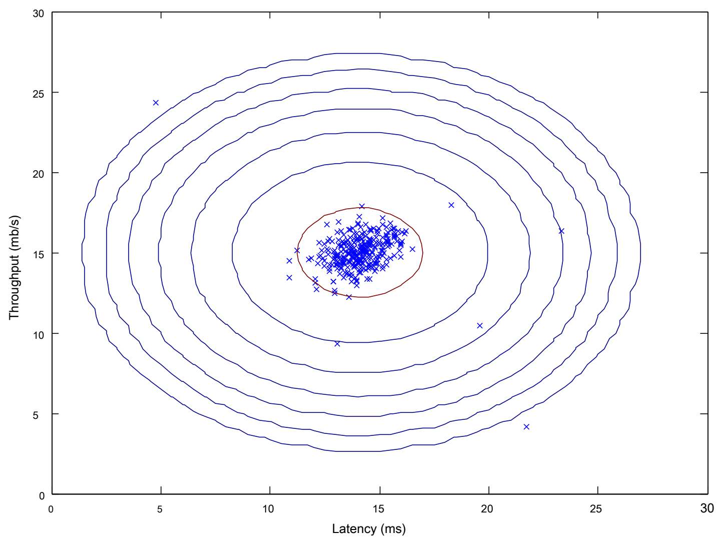


Figure 2: The Gaussian distribution contours of the distribution fit to the dataset.

more likely to be the anomalies in our dataset. One way to determine which examples are anomalies is to select a threshold based on a cross validation set. In this part of the exercise, you will implement an algorithm to select the threshold ε using the F_1 score on a cross validation set.

You should now complete the code in selectThreshold.m. For this, we will use a cross validation set $\{(x^{(1)}_{cv}, y^{(1)}_{cv}), \ldots, (x^{(m_{cv})}_{cv}, y^{(m_{cv})}_{cv})\}$, where the label $y = 1$ corresponds to an anomalous example, and $y = 0$ corresponds to a normal example. For each cross validation example, we will compute $p(x^{(i)}_{cv})$. The vector of all of these probabilities $p(x^{(1)}_{cv}), \ldots, p(x^{(m_{cv})}_{cv})$ is passed to selectThreshold.m in the vector pval. The corresponding labels $y^{(1)}_{cv}, \ldots, y^{(m_{cv})}_{cv}$ is passed to the same function in the vector yval.

The function selectThreshold.m should return two values; the first is the selected threshold ε. If an example x has a low probability $p(x) < \varepsilon$, then it is considered to be an anomaly. The function should also return the F_1 score, which tells you how well you're doing on finding the ground truth anomalies given a certain threshold. For many different values of ε, you will compute the resulting F_1 score by computing how many examples the current threshold classifies correctly and incorrectly.

The F_1 score is computed using precision ($prec$) and recall (rec):

$$F_1 = 2 \cdot \frac{prec \cdot rec}{prec + rec}, \qquad (3)$$

You compute precision and recall by:

$$prec = \frac{tp}{tp + fp} \tag{4}$$

$$rec = \frac{tp}{tp + fn}, \tag{5}$$

where

- *tp* is the number of true positives: the ground truth label says it's an anomaly and our algorithm correctly classified it as an anomaly.
- *fp* is the number of false positives: the ground truth label says it's not an anomaly, but our algorithm incorrectly classified it as an anomaly.
- *fn* is the number of false negatives: the ground truth label says it's an anomaly, but our algorithm incorrectly classified it as not being anomalous.

In the provided code selectThreshold.m, there is already a loop that will try many different values of ε and select the best ε based on the F_1 score.

You should now complete the code in selectThreshold.m. You can implement the computation of the F1 score using a for-loop over all the cross validation examples (to compute the values *tp, fp, fn). You should see a value for epsilon of about 8.99e-05.

Implementation Note: In order to compute *tp, fp* and *fn,* you may be able to use a vectorized implementation rather than loop over all the examples. This can be implemented by Octave's equality test between a vector and a single number. If you have several binary values in an *n*-dimensional binary vector $v \in \{0, 1\}^n$, you can find out how many values in this vector are 0 by using: sum(*v* == 0). You can also apply a logical and operator to such binary vectors. For instance, let cvPredictions be a binary vector of the size of your number of cross validation set, where the *i*-th element is 1 if your algorithm considers $x^{(i)}_{cv}$ an anomaly, and 0 otherwise. You can then, for example, compute the number of false positives using: fp = sum((cvPredictions == 1) & (yval == 0)).

Once you have completed the code in selectThreshold.m, the next step in ex8.m will run your anomaly detection code and circle the anomalies in the plot (Figure 3).

You should now submit your select threshold function.

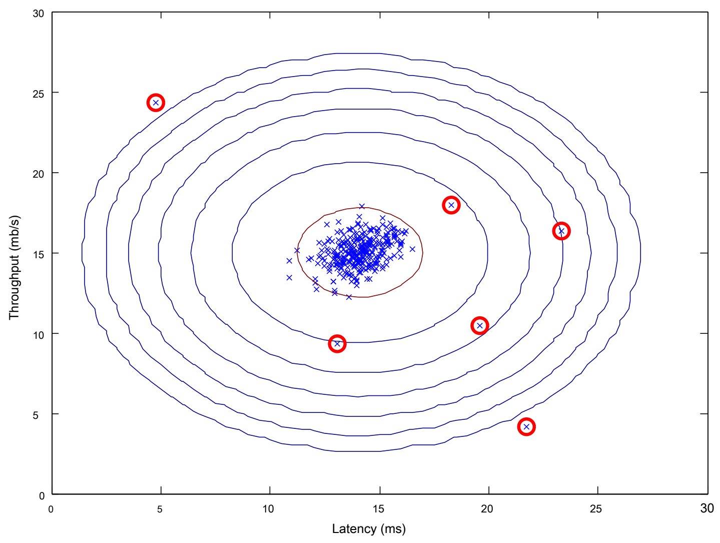


Figure 3: The classified anomalies.

1.4 High dimensional dataset

The last part of the script ex8.m will run the anomaly detection algorithm you implemented on a more realistic and much harder dataset. In this dataset, each example is described by 11 features, capturing many more properties of your compute servers.

The script will use your code to estimate the Gaussian parameters (μ_i and σ_i^2), evaluate the probabilities for both the training data X from which you estimated the Gaussian parameters, and do so for the the cross-validation set Xval. Finally, it will use selectThreshold to find the best threshold ε.

You should see a value epsilon of about 1.38e-18, and 117 anomalies found.

---

2 Recommender Systems

In this part of the exercise, you will implement the collaborative filtering learning algorithm and apply it to a dataset of movie ratings. [1] This dataset consists of ratings on a scale of 1 to 5. The dataset has $n_u = 943$ users, and
$n_m = 1682$ movies. For this part of the exercise, you will be working with the script ex8 cofi.m.

[1] MovieLens 100k Dataset from GroupLens Research.

In the next parts of this exercise, you will implement the function cofiCostFunc.m that computes the collaborative fitlering objective function and gradient. After implementing the cost function and gradient, you will use fmincg.m to learn the parameters for collaborative filtering.

2.1 Movie ratings dataset

The first part of the script ex8 cofi.m will load the dataset ex8 movies.mat, providing the variables Y and R in your Octave environment.

The matrix Y (a num movies $\times$ num users matrix) stores the ratings $y^{(i,j)}$ (from 1 to 5). The matrix R is an binary-valued indicator matrix, where $R(i, j) = 1$ if user j gave a rating to movie i, and $R(i, j) = 0$ otherwise. The objective of collaborative filtering is to predict movie ratings for the movies that users have not yet rated, that is, the entries with $R(i, j) = 0$. This will allow us to recommend the movies with the highest predicted ratings to the user.

To help you understand the matrix Y, the script ex8 cofi.m will compute the average movie rating for the first movie (Toy Story) and output the average rating to the screen.

Throughout this part of the exercise, you will also be working with the matrices, X and Theta:

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(n_m)})^T — \end{bmatrix}, \quad \text{Theta} = \begin{bmatrix} — (\theta^{(1)})^T — \\ — (\theta^{(2)})^T — \\ \vdots \\ — (\theta^{(n_u)})^T — \end{bmatrix}.$$

The i-th row of X corresponds to the feature vector $x^{(i)}$ for the i-th movie, and the j-th row of Theta corresponds to one parameter vector $\theta^{(j)}$, for the j-th user. Both $x^{(i)}$ and $\theta^{(j)}$ are n-dimensional vectors. For the purposes of this exercise, you will use $n = 100$, and therefore, $x^{(i)} \in \mathbb{R}^{100}$ and $\theta^{(j)} \in \mathbb{R}^{100}$.
Correspondingly, X is a $n_m \times 100$ matrix and Theta is a $n_u \times 100$ matrix.

2.2 Collaborative filtering learning algorithm

Now, you will start implementing the collaborative filtering learning algorithm. You will start by implementing the cost function (without regularization).

The collaborative filtering algorithm in the setting of movie recommendations considers a set of n-dimensional parameter vectors $x^{(1)}, \ldots, x^{(n_m)}$ and

$\theta^{(1)}, \ldots, \theta^{(n_u)}$, where the model predicts the rating for movie *i* by user *j* as $y^{(i,j)} = (\theta^{(j)})^T x^{(i)}$. Given a dataset that consists of a set of ratings produced by some users on some movies, you wish to learn the parameter vectors $x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ that produce the best fit (minimizes the squared error).

You will complete the code in cofiCostFunc.m to compute the cost function and gradient for collaborative filtering. Note that the parameters to the function (i.e., the values that you are trying to learn) are X and Theta. In order to use an off-the-shelf minimizer such as fmincg, the cost function has been set up to unroll the parameters into a single vector params. You had previously used the same vector unrolling method in the neural networks programming exercise.

2.2.1 Collaborative filtering cost function

The collaborative filtering cost function (without regularization) is given by

$$J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2.$$

You should now modify cofiCostFunc.m to return this cost in the variable J. Note that you should be accumulating the cost for user *j* and movie *i* only if $R(i, j) = 1$.

After you have completed the function, the script ex8 cofi.m will run your cost function. You should expect to see an output of 22.22.

You should now submit your cost function.

Implementation Note: We strongly encourage you to use a vectorized implementation to compute *J,* since it will later by called many times by the optimization package fmincg. As usual, it might be easiest to first write a non-vectorized implementation (to make sure you have the right answer), and the modify it to become a vectorized implementation (checking that the vectorization steps don't change your algorithm's output). To come up with a vectorized implementation, the following tip might be helpful: You can use the R matrix to set selected entries to 0. For example, R .* M will do an element-wise multiplication between M

and R; since R only has elements with values either 0 or 1, this has the effect of setting the elements of M to 0 only when the corresponding value in R is 0. Hence, sum(sum(R.*M)) is the sum of all the elements of M for which the corresponding element in R equals 1.

2.2.2 Collaborative filtering gradient

Now, you should implement the gradient (without regularization). Specifically, you should complete the code in cofiCostFunc.m to return the variables X grad and Theta grad. Note that X grad should be a matrix of the same size as X and similarly, Theta grad is a matrix of the same size as

Theta. The gradients of the cost function is given by:

$$\frac{\partial J}{\partial x_k^{(i)}} = \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)}$$

$$\frac{\partial J}{\partial \theta_k^{(j)}} = \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} .$$

Note that the function returns the gradient for both sets of variables by unrolling them into a single vector. After you have completed the code to compute the gradients, the script ex8 cofi.m will run a gradient check (checkCostFunction) to numerically check the implementation of your gradients. 2 If your implementation is correct, you should find that the analytical and numerical gradients match up closely.

You should now submit your collaborative filtering gradient function.

2 This is similar to the numerical check that you used in the neural networks exercise.

Implementation Note: You can get full credit for this assignment without using a vectorized implementation, but your code will run much more slowly (a small number of hours), and so we recommend that you try to vectorize your implementation.

To get started, you can implement the gradient with a for-loop over movies (for computing $\frac{\partial J}{\partial x^{(i)}_k}$) and a for-loop over users (for computing $\frac{\partial J}{\partial \theta^{(j)}_k}$). When you first implement the gradient, you might start with an unvectorized version, by implementing another inner for-loop that computes each element in the summation. After you have completed the gradient computation this way, you should try to vectorize your implementation (vectorize the inner for-loops), so that you're left with only two for-loops (one for looping over movies to compute $\frac{\partial J}{\partial x^{(i)}_k}$ for each movie, and one for looping over users to compute $\frac{\partial J}{\partial \theta^{(j)}_k}$ for each user).

Implementation Tip: To perform the vectorization, you might find this helpful: You should come up with a way to compute all the derivatives associated with $x^{(i)}_1, x^{(i)}_2, \ldots, x^{(i)}_n$ i.e., the derivative terms associated with the feature vector $x^{(i)}$ at the same time. Let us define the derivatives for the feature vector of the i-th movie as:

$$(X_{grad}(i,:))^T = \begin{bmatrix} \frac{\partial J}{\partial x^{(i)}_1} \\ \frac{\partial J}{\partial x^{(i)}_2} \\ \vdots \\ \frac{\partial J}{\partial x^{(i)}_n} \end{bmatrix} = \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})\theta^{(j)}$$

To vectorize the above expression, you can start by indexing into Theta and Y to select only the elements of interests (that is, those with $r(i, j) = 1$). Intuitively, when you consider the features for the i-th movie, you only need to be concern about the users who had given ratings to the movie, and this allows you to remove all the other users from Theta and Y.

Concretely, you can set idx = find(R(i, :)==1) to be a list of all the users that have rated movie i. This will allow you to create the temporary matrices Theta_{temp} = Theta(idx, :) and Y_{temp} = Y(i, idx) that index into Theta and Y to give you only the set of users which have rated the i-th movie. This will allow you to write the derivatives as:

$$X_{grad}(i,:) = (X(i,:) * \text{Theta}^T_{temp} - Y_{temp}) * \text{Theta}_{temp}.$$

(Note: The vectorized computation above returns a row-vector instead.)

After you have vectorized the computations of the derivatives with respect to $x^{(i)}$, you should use a similar method to vectorize the derivatives with respect to $\theta^{(j)}$ as well.

2.2.3 Regularized cost function

The cost function for collaborative filtering with regularization is given by

$$J(x^{(1)},\ldots,x^{(n_m)},\theta^{(1)},\ldots,\theta^{(n_u)}) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \left(\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2\right) + \left(\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2\right).$$

You should now add regularization to your original computations of the cost function, *J.* After you are done, the script ex8 cofi.m will run your regularized cost function, and you should expect to see a cost of about 31.34.

You should now submit your regularized cost function.

2.2.4 Regularized gradient

Now that you have implemented the regularized cost function, you should proceed to implement regularization for the gradient. You should add to your implementation in cofiCostFunc.m to return the regularized gradient by adding the contributions from the regularization terms. Note that the gradients for the regularized cost function is given by:

$$\frac{\partial J}{\partial x_k^{(i)}} = \sum_{j:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)\theta_k^{(j)} + \lambda x_k^{(i)}$$

$$\frac{\partial J}{\partial \theta_k^{(j)}} = \sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)x_k^{(i)} + \lambda \theta_k^{(j)}.$$

This means that you just need to add $\lambda x^{(i)}$ to the X grad(i,:) variable described earlier, and add $\lambda\theta^{(j)}$ to the Theta grad(j,:) variable described earlier.

After you have completed the code to compute the gradients, the script ex8 cofi.m will run another gradient check (checkCostFunction) to numerically check the implementation of your gradients.

You should now submit the regularized gradient function.

2.3 Learning movie recommendations

After you have finished implementing the collaborative filtering cost function and gradient, you can now start training your algorithm to make movie recommendations for yourself. In the next part of the ex8 cofi.m script, you can enter your own movie preferences, so that later when the algorithm runs, you can get your own movie recommendations! We have filled out some values according to our own preferences, but you should change this according to your own tastes. The list of all movies and their number in the dataset can be found listed in the file movie idx.txt.

2.3.1 Recommendations

```
Top recommendations for you:
Predicting rating 9.0 for movie Titanic (1997) Predicting rating
8.9 for movie Star Wars (1977)
Predicting rating 8.8 for movie Shawshank Redemption, The (1994) Predicting rating 8.5
for movie As Good As It Gets (1997) Predicting rating 8.5 for movie Good Will Hunting
(1997) Predicting rating 8.5 for movie Usual Suspects, The (1995) Predicting rating 8.5 for
movie Schindler’s List (1993) Predicting rating 8.4 for movie Raiders of the Lost Ark
(1981) Predicting rating 8.4 for movie Empire Strikes Back, The (1980) Predicting rating
8.4 for movie Braveheart (1995)



Original ratings provided: Rated 4 for Toy Story
(1995) Rated 3 for Twelve Monkeys (1995) Rated 5
for Usual Suspects, The (1995) Rated 4 for
Outbreak (1995)

Rated 5 for Shawshank Redemption, The (1994) Rated 3 for
While You Were Sleeping (1995) Rated 5 for Forrest Gump
(1994) Rated 2 for Silence of the Lambs, The (1991) Rated 4 for
Alien (1979) Rated 5 for Die Hard 2 (1990) Rated 5 for Sphere
(1998)
```

Figure 4: Movie recommendations

After the additional ratings have been added to the dataset, the script will proceed to train the collaborative filtering model. This will learn the parameters X and Theta. To predict the rating of movie *i* for user *j*, you need

to compute $(\theta^{(j)})^T x^{(i)}$. The next part of the script computes the ratings for all the movies and users and displays the movies that it recommends (Figure
4), according to ratings that were entered earlier in the script. Note that you might obtain a different set of the predictions due to different random initializations.

Submission and Grading

After completing various parts of the assignment, be sure to use the submit
function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	Submitted File	Points
Estimate Gaussian Parameters	estimateGuassian.m	15 points
Select Threshold	selectThreshold.m	15 points
Collaborative Filtering Cost	cofiCostFunc.m	20 points
Collaborative Filtering Gradient	cofiCostFunc.m	30 points
Regularized Cost	cofiCostFunc.m	10 points
Gradient with regularization	cofiCostFunc.m	10 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration. To prevent rapid-fire guessing, the system enforces a minimum of 5 minutes between submissions.