Week 1 Lecture Notes

机器学习：引言 - ML:Introduction

什么是机器学习？ - What is Machine Learning?

给出机器学习的两个定义。Arthur Samuel将其描述为："没有直接对问题进行显式编程的情况下，让计算机拥有学习能力的研究领域。"这是一个古老的、非正式的定义。

Tom Mitchell给出了一个更现代的定义："对于某类任务T和性能度量P，如果计算机程序在T上以P衡量的性能随着经验E而自我完善，那么就称这个计算机程序从经验E进行了学习。"

示例：玩跳棋

E = 许多玩跳棋棋的经验

T = 玩跳棋的任务

P = 程序赢得下一场比赛的概率

一般来说，任何机器学习问题都可以被分为两大类之一：

监督学习，或

无监督学习。

监督学习 - supervised learning

在监督学习中，我们会已知一个数据集，并且我们已经知道正确输出应该是什么样子，也会想到输入和输出之间存在某种关系。

监督学习问题分为"回归（regression）"和"分类（classification）"问题。在回归问题中，我们试图预测连续的输出结果，这意味着我们正试图将输入变量映射到一些连续函数。在分类问题中，我们试图得到离散的输出结果。换句话说，我们试图把输入变量映射成离散的类别。下面是关于连续和离散数据的数学描述。

例 1：

给定房屋市场上房屋大小的数据，试着预测它们的价格。价格高低的函数是一个连续的输出，所以这是一个回归问题。

我们可以把这个例子变成一个分类问题，比如，我们预测房屋"卖得比要价多还是少"。

例 2：

（a）回归 - 给定一个男性/女性的图片，我们根据给定的图片来预测他的年龄。

（b）分类 - 给定一个男性/女性的图片，我们预测他/她是高中，大学，研究生，以及年级。另一个分类的例子 - 银行必须根据某人的信用历史来决定是否给他贷款。

无监督学习 - Unsupervised Learning

无监督学习允许我们在几乎不知道结果应该是什么的情况下处理问题。我们可以从数据中得出某种结构，我们不必具体知道变量带来的影响。

我们可以通过基于数据中变量之间的关系来聚类数据来得出这种结构。

无监督学习没有基于预测结果的反馈，也就是说，没有“老师”来纠正你。

例：

聚类：收集1000篇关于美国经济的文章，找出一种方法，自动将这些文章归类为一个个小部分，这些小部分在不同的变量（如词频、句子长度、页数等）在某种程度上相似或相关。

非聚类：鸡尾酒会算法，它可以在混乱的数据中找到结构（比如在鸡尾酒会上从一组声音中识别出单个声音和音乐(https://en.wikipedia.org/wiki/Cocktail_party_.)）。这里有一个Quora答案来增强你的理解： https://www.quora.com/What-is-the-difference-between-supervised-and-unsupervised-learning-algorithms ?

机器学习：单变量线性回归 - linear regression with one variable

模型表示 - Model Representation

回想一下，在回归问题中，我们采用输入变量，并试图将输出拟合到一个代表结果的连续函数上。

单变量线性回归（linear regression with one variable）也称为“univariate linear regression”。

如果你想从**单个输入**x预测**单个输出**y，可以使用单变量线性回归。这里我们要进行**监督学习**，也就是说我们已经知道输入/输出的大致关系应该是什么。

假设函数 - the Hypothesis Function

我们的假设函数具有一般形式：

$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$

注意，这就像直线的方程。我们给出一个x值，然后$h_\theta(x)$根据θ_0 关于 θ_1 值，得到一个估计输出$\hat{y}$。换句话说，我们试图创建一个名为$h_\theta(x)$的函数，它试图将我们的输入数据（x）映射到输出数据（y）。

例子：

假设我们有以下的训练数据集：

input x	output y
0	4
1	7
2	7
3	8

现在随机猜一下假设函数 h_θ ： $\theta_0 = 2$ 和 $\theta_1 = 2$。 那么假设函数就是 $h_\theta(x) = 2 + 2x$。

x输入1，y就是4。 和数据集相差3。注意，我们将试用θ_0 和 θ_1 的各种值，试图通过映射在x-y平面上的数据点，找到最佳的“拟合”或最具代表性的“直线”。

代价函数 - Cost Function

我们可以使用**代价函数**来评估假设函数的精度。对于所有的输入x，取所有预测结果与实际输出y的差的平均值（实际上是一种更独特的平均值）。

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

简单来说，上式就是$\frac{1}{2}\bar{x}$，其中$\bar{x}$是$h_\theta(x_i) - y_i$的平方的平均值。

这个函数称为“平方误差函数”或“均方误差”。为了方便梯度下降的计算，平均值被$\left(\frac{1}{2m}\right)$减半，因为平方函数的导数项会产生2，这将抵消$\frac{1}{2}$项，计算更方便。

现在我们能够根据正确结果来准确测量我们的假设函数的精度，以便我们能够在新样本上预测结果。

如果我们用图像的方式来思考，训练数据集分布在X-Y平面上，现在要作一条直线（即$h_\theta(x)$定义的直线）穿过这个散乱的数据集。目标是得到一条最好的路线。最好的直线是这样的，使得散点到直线的平均距离最小。如果有最好的情况下，直线应该通过我们训练数据集的所有点。即$J(\theta_0, \theta_1)$的值将是0。

机器学习：梯度下降 - ML:Gradient Descent

现在，我们有了假设函数，也有了评估假设函数与数据吻合程度的方法。现在我们需要估算假设函数中的参数。这就是梯度下降的用处。

假设我们将θ_0 和θ_1看成变量，来绘制假设函数（实际上我们正在将代价函数作为参数估计的函数来绘制）。这可能有点混乱，我们正进行进一步的抽象。我们不是画x和y本身，而是假设函数的参数范围，以及选择特定参数集对应产生的代价。

我们用θ_0代表x轴，θ_1代表y轴，代价函数的输出作为垂直于x轴和y轴的z轴。这个图上的点，就是这些具体的θ参数对应的代价。

当代价函数的值位于图的底部时，即当其值最小时，我们就会知道我们已经成功了。

求最小值的方法是取代价函数的导数（即，求切线）。切线的斜率就是这一点上的导数，它将给我们一个移动的方向。我们一步步移动，顺着代价函数下降，每一步的大小由参数α决定，称为学习速率。

梯度下降算法是：

重复此步，直到收敛：

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$， （$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ 是分别对θ_0和θ_1求偏导数）

其中j=0,1 表示θ_0 或 θ_1

更直观一些，可以看成：

重复此步，直到收敛：

$\theta_j := \theta_j - \alpha[\text{切线的斜率}]$

线性回归的梯度下降 - Gradient Descent for Linear Regression

当梯度下降具体应用于线性回归时，可以导出梯度下降方程的新形式。

我们可以用实际的成本函数和实际的假设函数来代入，并将方程式修改为（公式的推导超出了本课程的范围，但是在这里直接给出可用的公式）：

$$\text{repeat until convergence} : \{$$
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$
$$\}$$

公式推导（译者注）

$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ 是分别对θ_0和θ_1求偏导数

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \cdot \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$j = 0$ 时：

$$\begin{aligned}
&\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
=&\frac{\partial}{\partial \theta_0} \cdot \frac{1}{2m} \cdot \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \\
=&\frac{1}{2m} \cdot 2 \cdot \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot \frac{\partial}{\partial \theta_0} (h_\theta(x^{(i)}) - y^{(i)}) \\
=&\frac{1}{m} \cdot \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot \frac{\partial}{\partial \theta_0} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) \\
=&\frac{1}{m} \cdot \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})
\end{aligned}$$

$j = 1$ 时：

$$
\begin{aligned}
&\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)\\
=&\frac{\partial}{\partial\theta_1}\cdot\frac{1}{2m}\cdot\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2\\
=&\frac{1}{2m}\cdot 2\cdot\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})\cdot\frac{\partial}{\partial\theta_1}(h_\theta(x^{(i)})-y^{(i)})\\
=&\frac{1}{m}\cdot\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})\cdot\frac{\partial}{\partial\theta_1}(\theta_0+\theta_1x^{(i)}-y^{(i)})\\
=&\frac{1}{m}\cdot\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})\cdot x^{(i)}
\end{aligned}
$$

带入递归下降算法，得：

$$
\begin{aligned}
\theta_0 &:= \theta_0-\alpha\frac{1}{m}\cdot\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})\\
\theta_1 &:= \theta_1-\alpha\frac{1}{m}\cdot\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})\cdot x^{(i)}
\end{aligned}
$$

其中m是训练集的大小，θ_0与θ_1将同步更新，x_i, y_i是训练集的值（数据）。

注意，我们现在是分别计算θ_0和 θ_1，计算θ_1中，最后乘了一个x_i。

如果我们随意给出一个假设，然后反复运用这些梯度方程，我们的假设将变得越来越精确。

线性回归的梯度下降：视觉化示例 - Gradient Descent for Linear Regression: visual worked example

这个视频 (https://www.youtube.com/watch?v=WnqQrPNYz5Q) 可能有用，它可视化了随着代价函数的减小，假设函数的学习过程。

（译者注：后文的内容都是线性代数的基础，中文的线性代数教程很多，就不翻译了。）

机器学习：线性代数回顾 - ML:Linear Algebra Review

Khan Academy has excellent Linear Algebra Tutorials (https://www.khanacademy.org/#linear-algebra)

矩阵和矢量 - Matrices and Vectors

Matrices are 2-dimensional arrays:

$$
\begin{bmatrix}
a & b & c\\
d & e & f\\
g & h & i\\
j & k & l
\end{bmatrix}
$$

The above matrix has four rows and three columns, so it is a 4 x 3 matrix.

A vector is a matrix with one column and many rows:

$$\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}$$

So vectors are a subset of matrices. The above vector is a 4 x 1 matrix.

Notation and terms:

- A_{ij} refers to the element in the ith row and jth column of matrix A.
- A vector with 'n' rows is referred to as an 'n'-dimensional vector
- v_i refers to the element in the ith row of the vector.
- In general, all our vectors and matrices will be 1-indexed. Note that for some programming languages, the arrays are 0-indexed.
- Matrices are usually denoted by uppercase names while vectors are lowercase.
- "Scalar" means that an object is a single value, not a vector or matrix.
- $\mathbb{R}$ refers to the set of scalar real numbers
- $\mathbb{R}^n$ refers to the set of n-dimensional vectors of real numbers

矩阵加法和标量乘法 - Addition and Scalar Multiplication

Addition and subtraction are **element-wise**, so you simply add or subtract each corresponding element:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} a+w & b+x \\ c+y & d+z \end{bmatrix}$$

To add or subtract two matrices, their dimensions must be **the same**.

In scalar multiplication, we simply multiply every element by the scalar value:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * x = \begin{bmatrix} a*x & b*x \\ c*x & d*x \end{bmatrix}$$

矩阵-矢量乘法 - Matrix-Vector Multiplication

We map the column of the vector onto each row of the matrix, multiplying each element and summing the result.

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a*x & b*y \\ c*x & d*y \\ e*x & f*y \end{bmatrix}$$

The result is a **vector**. The vector must be the **second** term of the multiplication. The number of **columns** of the matrix must equal the number of **rows** of the vector.

An **m x n matrix** multiplied by an **n x 1 vector** results in an **m x 1 vector**.

矩阵-矩阵乘法 - Matrix-Matrix Multiplication

We multiply two matrices by breaking it into several vector multiplications and concatenating the result

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} * \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} a*w+b*y & a*x+b*z \\ c*w+d*y & c*x+d*z \\ e*w+f*y & e*x+f*z \end{bmatrix}$$

An **m x n matrix** multiplied by an **n x o matrix** results in an **m x o** matrix. In the above example, a 3 x 2 matrix times a 2 x 2 matrix resulted in a 3 x 2 matrix.

To multiply two matrices, the number of **columns** of the first matrix must equal the number of **rows** of the second matrix.

矩阵乘法的性质 - Matrix Multiplication Properties

- Not commutative. $A*B \neq B*A$
- Associative. $(A*B)*C=A*(B*C)$

The **identity matrix**, when multiplied by any matrix of the same dimensions, results in the original matrix. It's just like multiplying numbers by 1. The identity matrix simply has 1's on the diagonal (upper left to lower right diagonal) and 0's elsewhere.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When multiplying the identity matrix after some matrix ($A*I$), the square identity matrix should match the other matrix's **columns**. When multiplying the identity matrix before some other matrix ($I*A$), the square identity matrix should match the other matrix's **rows**.

逆矩阵和转置矩阵 - Inverse and Transpose

The **inverse** of a matrix A is denoted A^{-1}. Multiplying by the inverse results in the identity matrix.

A non square matrix does not have an inverse matrix. We can compute inverses of matrices in octave with the pinv(A) function [1] and in matlab with the inv(A) function. Matrices that don't have an inverse are *singular* or *degenerate*.

The **transposition** of a matrix is like rotating the matrix 90° in clockwise direction and then reversing it. We can compute transposition of matrices in matlab with the transpose(A) function or A':

$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$$

$$A^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

In other words:

$$A_{ij} = A^T_{ji}$$