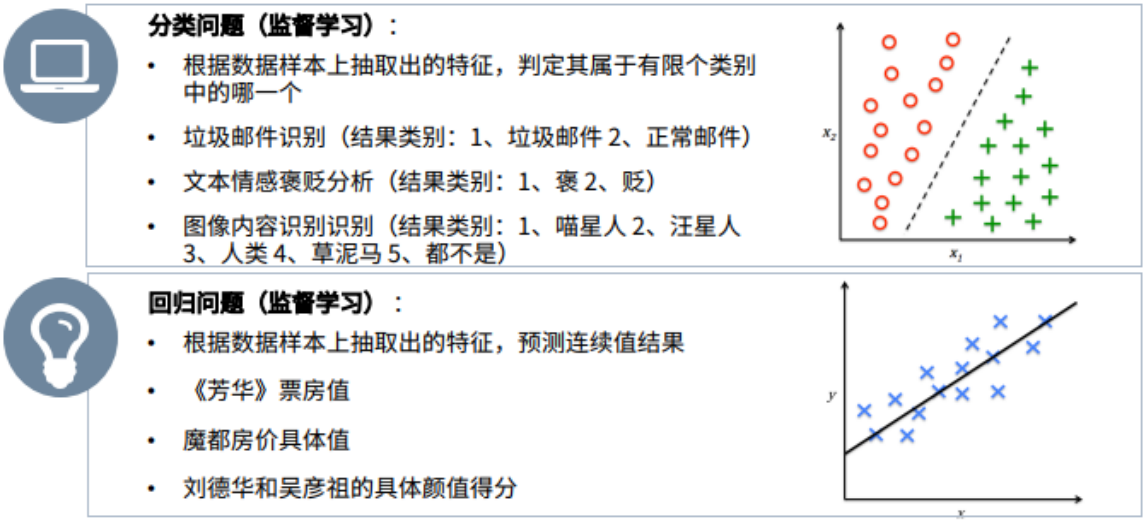
常见算法课程讲义

0. 什么是机器学习

利用大量的数据样本，使得计算机通过不断的学习获得一个模型，用来对新的未知数据做预测。

- 有监督学习(分类、回归)

同时将数据样本和标签输入给模型，模型学习到数据和标签的映射关系，从而对新数据进行预测。



- 无监督学习(聚类)

只有数据，没有标签，模型通过总结规律，从数据中挖掘出信息。

- 强化学习

强化学习会在没有任何标签的情况下，通过先尝试做出一些行为得到一个结果，通过这个结果是对还是错的反馈，调整之前的行为，就这样不断的调整，算法能够学习到在什么样的情况下选择什么样的行为可以得到最好的结果。 就好比你有一只还没有训练好的小狗，每当它把屋子弄乱后，就减少美味食物的数量（惩罚），每次表现不错时，就加倍美味食物的数量（奖励），那么小狗最终会学到一个知识，就是把客厅弄乱是不好的行为。

1. TF-IDF算法入门

1.1. 引言

有一篇很长的文章，我要用计算机提取它的关键词（Automatic Keyphrase extraction），完全不加以人工干预，请问怎样才能正确做到？

这个问题涉及到数据挖掘、文本处理、信息检索等很多计算机前沿领域，但是出乎意料的是，有一个非常简单的经典算法，可以给出令人相当满意的结果。它简单到都不需要高等数学，普通人只用10分钟就可以理解，这就是我今天想要介绍的TF-IDF算法。

1.2. TF-IDF的概述

$$词频(TF)=\frac{某个词在文档中的出现次数}{文章的总词数}$$

$$逆文档频率(IDF)=\log(\frac{语料库的文档总数}{包含该词的文档数+1})$$

$$TF-IDF=词频(TF)\times 逆文档频率(IDF)$$

这些都代表什么意思呢？

让我们从一个实例开始讲起。假定现在有一篇长文《中国的蜜蜂养殖》，我们准备用计算机提取它的关键词。

一个容易想到的思路，就是找到出现次数最多的词。如果某个词很重要，它应该在这篇文章中多次出现。于是，我们进行"**词频**"（**Term Frequency，缩写为TF**）统计。

结果你肯定猜到了，出现次数最多的词是----"的"、"是"、"在"----这一类最常用的词。它们叫做"**停用词**"（**stop words**），**表示对找到结果毫无帮助、必须过滤掉的词。**

假设我们把它们都过滤掉了，只考虑剩下的有实际意义的词。这样又会遇到了另一个问题，我们可能发现"中国"、"蜜蜂"、"养殖"这三个词的出现次数一样多。这是不是意味着，作为关键词，它们的重要性是一样的？

显然不是这样。因为"中国"是很常见的词，相对而言，"蜜蜂"和"养殖"不那么常见。如果这三个词在一篇文章的出现次数一样多，有理由认为，"蜜蜂"和"养殖"的重要程度要大于"中国"，也就是说，在关键词排序上面，"蜜蜂"和"养殖"应该排在"中国"的前面。

所以，我们需要一个重要性调整系数，衡量一个词是不是常见词。如果某个词比较少见，但是它在这篇文章中多次出现，那么它很可能就反映了这篇文章的特性，正是我们所需要的关键词。

用统计学语言表达，就是在词频的基础上，要对每个词分配一个"重要性"权重。最常见的词（"的"、"是"、"在"）给予最小的权重，较常见的词（"中国"）给予较小的权重，较少见的词（"蜜蜂"、"养殖"）给予较大的权重。**这个权重叫做"逆文档频率"（Inverse Document Frequency，缩写为IDF），它的大小与一个词的常见程度成反比。**

知道了"词频"（TF）和"逆文档频率"（IDF）以后，将这两个值相乘，就得到了一个词的**TF-IDF**值。**某个词对文章的重要性越高，它的TF-IDF值就越大。**所以，排在最前面的几个词，就是这篇文章的关键词。

1.3. TF-IDF计算

第一步，计算词频。

$$词频(TF) = 某个词在文章中的出现次数$$

考虑到文章有长短之分，为了便于不同文章的比较，进行"词频"标准化。

$$词频(TF) = \frac{某个词在文章中的出现次数}{文章的总词数}$$

或者

$$词频(TF) = \frac{某个词在文章中的出现次数}{该文出现次数最多的词的出现次数}$$

第二步，计算逆文档频率。

这时，需要一个语料库（corpus），用来模拟语言的使用环境。

$$逆文档频率(IDF) = \log(\frac{语料库的文档总数}{包含该词的文档数 + 1})$$

如果一个词越常见，那么分母就越大，逆文档频率就越小越接近0。分母之所以要加1，是为了避免分母为0（即所有文档都不包含该词）。log表示对得到的值取对数。

第三步，计算TF-IDF。

$$TF - IDF = 词频(TF) \times 逆文档频率（IDF）$$

可以看到，**TF-IDF与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比。**所以，自动提取关键词的算法就很清楚了，就是计算出文档的每个词的TF-IDF值，然后按降序排列，取排在最前面的几个词。

还是以《中国的蜜蜂养殖》为例，假定该文长度为1000个词，"中国"、"蜜蜂"、"养殖"各出现20次，则这三个词的"词频"（TF）都为0.02。然后，搜索Google发现，包含"的"字的网页共有250亿张，假定这就是中文网页总数。包含"中国"的网页共有62.3亿张，包含"蜜蜂"的网页为0.484亿张，包含"养殖"的网页为0.973亿张。则它们的逆文档频率（IDF）和TF-IDF如下：

	包含该词的文档数（亿）	IDF	TF-IDF
中国	62.3	0.603	0.0121
蜜蜂	0.484	2.713	0.0543
养殖	0.973	2.410	0.0482

从上表可见，"蜜蜂"的TF-IDF值最高，"养殖"其次，"中国"最低。（如果还计算"的"字的TF-IDF，那将是一个极其接近0的值。）所以，如果只选择一个词，"蜜蜂"就是这篇文章的关键词。

除了自动提取关键词，TF-IDF算法还可以用于许多别的地方。比如，信息检索时，对于每个文档，都可以分别计算一组搜索词（"中国"、"蜜蜂"、"养殖"）的TF-IDF，将它们相加，就可以得到整个文档的TF-IDF。这个值最高的文档就是与搜索词最相关的文档。

TF-IDF算法的优点是简单快速，结果比较符合实际情况。缺点是，单纯以"词频"衡量一个词的重要性，不够全面，有时重要的词可能出现次数并不多。而且，这种算法无法体现词的位置信息，出现位置靠前的词与出现位置靠后的词，都被视为重要性相同，这是不正确的。（一种解决方法是，对全文的第一段和每一段的第一句话，给予较大的权重。）

2. 余弦相似性

2.1. 引言

上一次，我用TF-IDF算法自动提取关键词。

今天，我们再来研究另一个相关的问题。有些时候，除了找到关键词，我们还希望找到与原文章相似的其他文章。比如，"Google新闻"在主新闻下方，还提供多条相似的新闻。

为了找出相似的文章，需要用到"**余弦相似性**"（cosine similiarity）。下面，我举一个例子来说明，什么是"余弦相似性"。

2.2. 余弦相似度概述

余弦相似性通过测量两个向量的夹角的余弦值来度量它们之间的相似性。0度角的余弦值是1，而其他任何角度的余弦值都不大于1；并且其最小值是-1。从而两个向量之间的角度的余弦值确定两个向量是否大致指向相同的方向。两个向量有相同的指向时，余弦相似度的值为1；两个向量夹角为90°时，余弦相似度的值为0；两个向量指向完全相反的方向时，余弦相似度的值为-1。这结果是与向量的长度无关的，仅仅与向量的指向方向相关。余弦相似度通常用于正空间，因此给出的值为0到1之间。

注意这上下界对任何维度的向量空间中都适用，而且余弦相似性最常用于高维正空间。例如在信息检索中，每个词项被赋予不同的维度，而一个维度由一个向量表示，其各个维度上的值对应于该词项在文档中出现的频率。余弦相似度因此可以给出两篇文档在其主题方面的相似度。

另外，它通常用于文本挖掘中的文件比较。此外，在数据挖掘领域中，会用到它来度量集群内部的凝聚力。

定义

两个向量间的余弦值可以通过使用欧几里得点积公式求出：

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos\theta.$$

给定两个属性向量，*A*和*B*，其余弦相似性*θ*由点积和向量长度给出，如下所示：

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}.$$

这里的

A_i, B_i

分别代表向量A和B的各分量

给出的相似性范围从-1到1：-1意味着两个向量指向的方向正好截然相反，1表示它们的指向是完全相同的，0通常表示它们之间是独立的，而在这之间的值则表示中间的相似性或相异性。

对于文本匹配，属性向量*A*和*B*通常是文档中的词频向量。余弦相似性，可以被看作是在比较过程中把文件长度正规化的方法。

在信息检索的情况下，由于一个词的频率（TF-IDF权）不能为负数，所以这两个文档的余弦相似性范围从0到1。并且，两个词的频率向量之间的角度不能大于90°。

2.3. 余弦相似度计算

为了简单起见，我们先从句子着手。

> 句子A：我喜欢看电视，不喜欢看电影。
>
> 句子B：我不喜欢看电视，也不喜欢看电影。

请问怎样才能计算上面两句话的相似程度？

基本思路是：***如果这两句话的用词越相似，它们的内容就应该越相似。因此，可以从词频入手，计算它们的相似程度。***

第一步，分词。

> 句子A：我/喜欢/看/电视，不/喜欢/看/电影。
>
> 句子B：我/不/喜欢/看/电视，也/不/喜欢/看/电影。

第二步，列出所有的词。

> 我，喜欢，看，电视，电影，不，也。

第三步，计算词频。

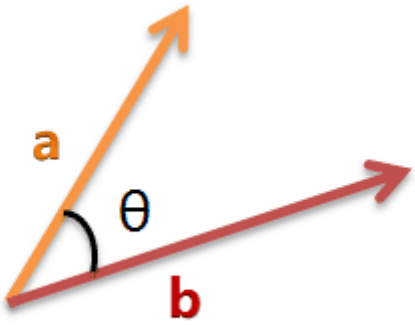
> 句子A：我 1，喜欢 2，看 2，电视 1，电影 1，不 1，也 0。
>
> 句子B：我 1，喜欢 2，看 2，电视 1，电影 1，不 2，也 1。

第四步，写出词频向量。

> 句子A：[1, 2, 2, 1, 1, 1, 0]
>
> 句子B：[1, 2, 2, 1, 1, 2, 1]

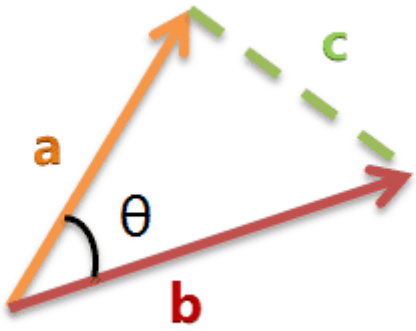
到这里，问题就变成了如何计算这两个向量的相似程度。

我们可以把它们想象成空间中的两条线段，都是从原点（[0, 0, ...]）出发，指向不同的方向。两条线段之间形成一个夹角，如果夹角为0度，意味着方向相同、线段重合；如果夹角为90度，意味着形成直角，方向完全不相似；如果夹角为180度，意味着方向正好相反。**因此，我们可以通过夹角的大小，来判断向量的相似程度。夹角越小，就代表越相似。**



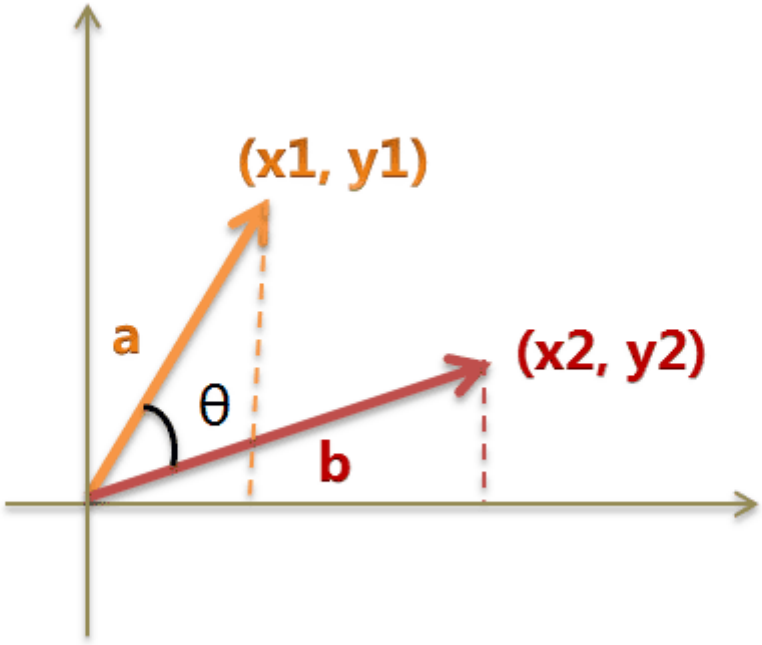
以二维空间为例，上图的a和b是两个向量，我们要计算它们的夹角θ。余弦定理告诉我们，可以用下面的公式求得：

$$\cos\theta = \frac{a^2 + b^2 - c^2}{2ab}$$



假定a向量是[x1, y1]，b向量是[x2, y2]，那么可以将余弦定理改写成下面的形式：

$$\cos\theta = \frac{x_1x_2 + y_1y_2}{\sqrt{x_1^2 + y_1^2} \times \sqrt{x_2^2 + y_2^2}}$$



数学家已经证明，余弦的这种计算方法对n维向量也成立。假定A和B是两个n维向量，A是 [A1, A2, ..., An] ，B是 [B1, B2, ..., Bn] ，则A与B的夹角θ的余弦等于：

$$\cos\theta = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} = \frac{A \cdot B}{|A| \times |B|}$$

使用这个公式，我们就可以得到，句子A与句子B的夹角的余弦。

$$\cos\theta = \frac{1 \times 1 + 2 \times 2 + 2 \times 2 + 1 \times 1 + 1 \times 1 + 1 \times 2 + 0 \times 1}{\sqrt{1^2 + 2^2 + 2^2 + 1^2 + 1^2 + 1^2 + 0^2} \times \sqrt{1^2 + 2^2 + 2^2 + 1^2 + 1^2 + 2^2 + 1^2}}$$

$$= \frac{13}{\sqrt{12} \times \sqrt{16}}$$

$$= 0.938$$

余弦值越接近1，就表明夹角越接近0度，也就是两个向量越相似，这就叫"余弦相似性"。所以，上面的句子A和句子B是很相似的，事实上它们的夹角大约为20.3度。

由此，我们就得到了"找出相似文章"的一种算法：

> （1）使用TF-IDF算法，找出两篇文章的关键词；
>
> （2）每篇文章各取出若干个关键词（比如20个），合并成一个集合，计算每篇文章对于这个集合中的词的词频（为了避免文章长度的差异，可以使用相对词频）；

> （3）生成两篇文章各自的词频向量；
>
> （4）计算两个向量的余弦相似度，值越大就表示越相似。

"余弦相似度"是一种非常有用的算法，只要是计算两个向量的相似程度，都可以采用它。

3.分词方式

3.1. IK Analyzer简介

为什么要分词呢，当大数据处理中要提取语句的特征值，进行向量计算。所有我们要用开源分词工具把语句中的关键词提取出来。

IK Analyzer是什么呢，就是我们需要的这个工具，是基于java开发的轻量级的中文分词工具包。它是以开源项目Luence为主体的，结合词典分词和文法分析算法的中文分词组件。IK有很多版本，在2012版本中，IK实现了简单的分词歧义排除算法。

我们为什么选择IK作为我们的分词工具呢，这里我们简单介绍一下。这里我们采用了网上的一些介绍。

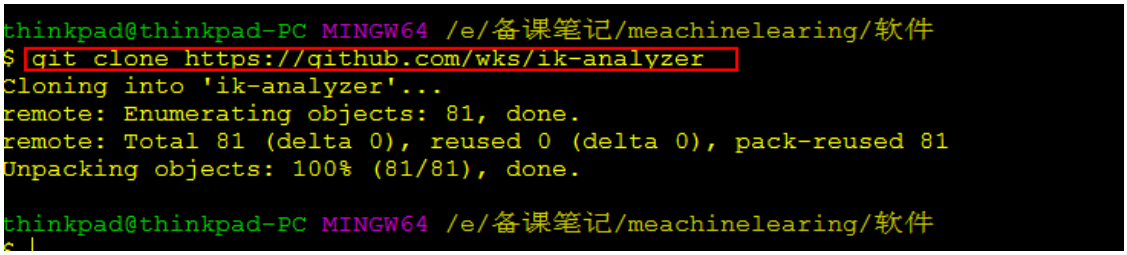
1. IK才用了特有的"正向迭代最细粒度切分算法"，支持细粒度和智能分词两种切分模式。
2. 在系统环境：Core2 i7 3.4G双核，4G内存，window 7 64位， Sun JDK 1.6_29 64位 普通pc环境测试，IK2012具有160万字/秒（3000KB/S）的高速处理能力。
3. 2012版的只能分词模式支持简单的分词排歧义处理和数量词合并输出。
4. 用了多子处理器分析模式，支持 英文字母 数字 中文词汇等
5. 优化词典存储，更小的内存占用。

3.2. IKAnalyzer的引入使用

- 由于maven库里没有ik的坐标。我们需要手动添加到本地的maven仓库中。或则lib引用
- 项目地址：https://github.com/wks/ik-analyzer

1. 首先克隆代码

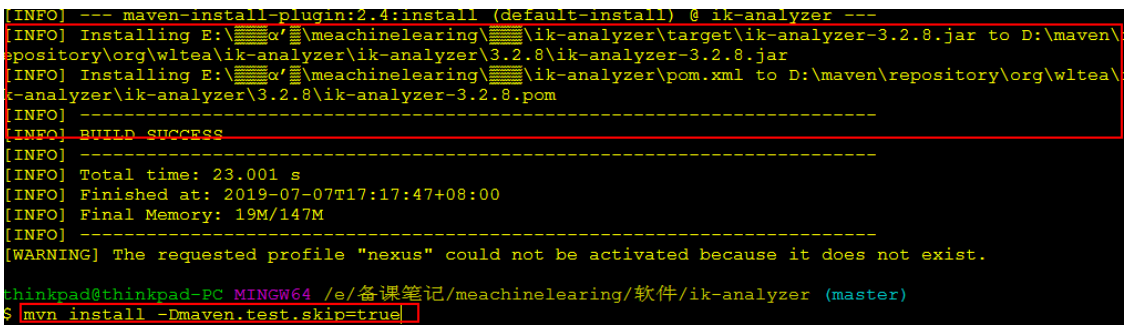
   git clone https://github.com/wks/ik-analyzer



   克隆到本地。

2. 编译并且安装到本地的repository

   mvn install -Dmaven.test.skip=true



(1)编译后也可以将jar上传到自己的maven私有库（如果有maven私有库，那么久直接使用2012版本，直接网上下载，然后上传到maven库即可）。

(2)可以放在本地maven仓库的对应坐标

(3)项目lib下引用

3. 在pom.xml中加入如下配置即可

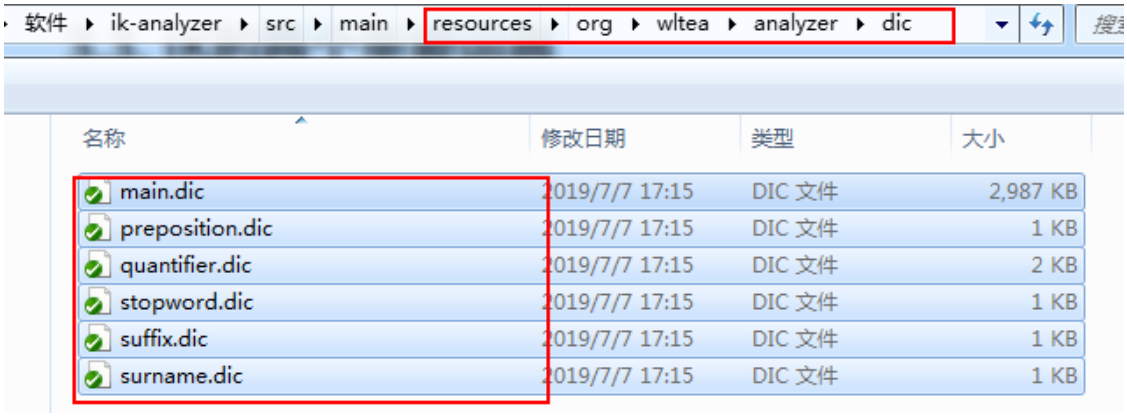
```
<dependency>
   <groupId>org.wltea.ik-analyzer</groupId>
    <artifactId>ik-analyzer</artifactId>
    <version>3.2.8</version>
</dependency>
```

3.3. IK的两个重要词典

- 扩展词典：为的是让需要切分的字符串的词语 根据扩展词典里的词，不要切分开来。

例如：扩展词典中有：中国的台湾 。那么原本会切分成：中国 的 台湾 在 东海 。会切分成：中国的台湾 在 东海

- 停止词典：对比停止词典，直接删掉停止词典中出现的词语



3.4. IK的使用

创建maven项目：ik-analyzer-study。

- pom文件如下：

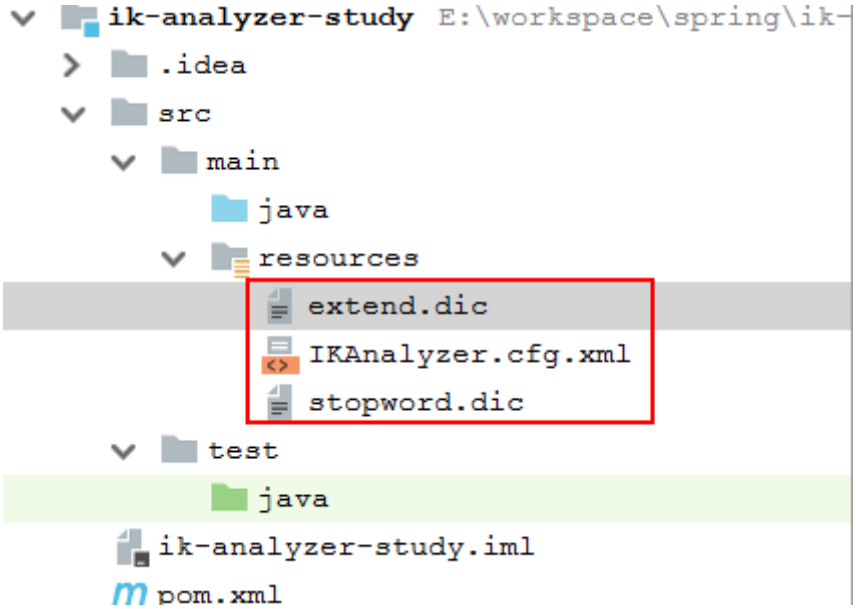
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.desheng.bigdata</groupId>
    <artifactId>ik-analyzer-study</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.wltea.ik-analyzer</groupId>
            <artifactId>ik-analyzer</artifactId>
            <version>3.2.8</version>
```

```
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.lucene/lucene-
analyzers -->
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-analyzers</artifactId>
            <version>3.6.2</version>
        </dependency>
    </dependencies>
</project>
```

- 非常重要的resources目录如下



1. IKAnalyzer.cfg.xml文件

   默认配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典
    <entry key="ext_dict">extend.dic;</entry>
    -->
    <entry key="ext_dict">/mydict.dic;</entry>
    <!--用户可以在这里配置自己的扩展停止词字典
    <entry key="ext_stopwords">/ext_stopword.dic</entry>
    -->
    <entry key="ext_stopwords">stopword.dic</entry>
</properties>
```

   这是ik分词器的核心配置文件，必须放在classpath的根目录下面，用于配置分词器的相关字典信息。比如ext_dict，比如ext_stopwords等等。

2. extend.dic文件 扩展词典

   比如这里配置如下

```
这是一个
巨大的墙
```

3. stopword.dic文件 停用词字典

比如如下配置：

```
一个
一
个
的
```

- 编码测试

```
package com.desheng.bigdata.ik;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;
import org.wltea.analyzer.lucene.IKAnalyzer;

import java.io.IOException;
import java.io.StringReader;

public class IKAnalyzerTest {
    public static void main(String[] args) {
        String line = "这是一个粗糙的栅栏，浪费钱，我想要一堵巨大的墙!”网友Mary说，还附上了“理想”中的边境墙照片";
        String s = beginAnalyzer(line);
        System.out.println(s);
    }
    public static String beginAnalyzer(String line){
        IKAnalyzer analyzer = new IKAnalyzer();
        try {
            return printAnalyzerResult(analyzer, line);
        } catch (IOException e) {
            e.printStackTrace();
        }

        return null;
    }

    public static String printAnalyzerResult(Analyzer analyzer, String keyword)
throws IOException {
        String resultData = "";
        String infoData = "";

        TokenStream tokenStream = analyzer.tokenStream("content",new
StringReader(keyword));
        tokenStream.addAttribute(CharTermAttribute.class);
        while(tokenStream.incrementToken()){
            CharTermAttribute charTermAttribute =
tokenStream.getAttribute(CharTermAttribute.class);
            infoData = infoData+ "\n"+charTermAttribute.toString();

        }
        if(!"".equals(infoData)){
            resultData = resultData + infoData.trim()+"\r\n";
        }else{
            resultData = "";
```

```
        }
        return resultData;
    }
}
```

分词结果：

```
"D:\Program Files\Java\64bit\jdk1.8.0_112\bin\java" ...
这是
一个
粗糙
的
栅栏
浪费
费钱
我
想要
一堵
巨大
的
墙
网友
mary
说
还
附上
上了
理想
中
的
边境
墙
照片
```

4. kmeans算法简介

4.1. 引言

K-means中心思想：事先确定常数K，常数K意味着最终的聚类类别数，首先随机选定初始点为质心，并通过计算每一个样本与质心之间的相似度(这里为欧式距离)，将样本点归到最相似的类中，接着，重新计算每个类的质心(即为类中心)，重复这样的过程，直到质心不再改变，最终就确定了每个样本所属的类别以及每个类的质心。由于每次都要计算所有的样本与每一个质心之间的相似度，故在大规模的数据集上，K-Means算法的收敛速度比较慢。 **聚类算法**：是一种典型的无监督学习算法，主要用于将相似的样本自动归到一个类别中。 聚类算法与分类算法最大的区别是：聚类算法是无监督的学习算法，而分类算法属于监督的学习 算法，分类是知道结果的。 在聚类算法中根据样本之间的相似性，将样本划分到不同的类别中，对于不同的相似度计算方法，会得到不同的聚类结果，常用的相似度计算方法有欧式距离法。

4.2. K-means聚类算法

1 K-means算法的相关描述

聚类是一种无监督的学习，它将相似的对象归到同一簇中。聚类的方法几乎可以应用所有对象，簇内的对象越相似，聚类的效果就越好。K-means算法中的k表示的是聚类为k个簇，means代表取每一个聚类中数据值的均值作为该簇的中心，或者称为**质心**，即**用每一个的类的质心对该簇进行描述。**

聚类和分类最大的不同在于，分类的目标是事先已知的，而聚类则不一样，聚类事先不知道目标变量是什么，类别没有像分类那样被预先定义出来，所以，聚类有时也叫无监督学习。

聚类分析试图将相似的对象归入同一簇，将不相似的对象归为不同簇，那么，显然需要一种合适的相似度计算方法，我们已知的有很多相似度的计算方法，比如欧氏距离，余弦距离，汉明距离等。事实上，我们应该根据具体的应用来选取合适的相似度计算方法。

当然，任何一种算法都有一定的缺陷，没有一种算法时完美的，有的只是人类不断追求完美，不断创新的意志。K-means算法也有它的缺陷，但是我们可以通过一些后处理来得到更好的聚类结果，这些在后面都会一一降到。

K-means算法虽然比较容易实现，但是其可能收敛到局部最优解，且在大规模数据集上收敛速度相对较慢。

2 K-means算法的工作流程

1.选择聚类的个数k（kmeans算法传递超参数的时候，只需设置最大的K值） 2.任意产生k个聚类，然后确定聚类中心，或者直接生成k个中心。 3.对每个点确定其聚类中心点。 4.再计算其聚类新中心。 5.重复以上步骤直到满足收敛要求。（通常就是确定的中心点不再改变。）

首先，随机确定k个初始点的质心；然后将数据集中的每一个点分配到一个簇中，即为每一个点找到距其最近的质心，并将其分配给该质心所对应的簇；该步完成后，每一个簇的质心更新为该簇所有点的平均值。伪代码如下：

```
创建k个点作为起始质心，可以随机选择(位于数据边界内)
    当任意一个点的簇分配结果发生改变时
        对数据集中每一个点
                对每个质心
                    计算质心与数据点之间的距离
                将数据点分配到距其最近的簇
        对每一个簇，计算簇中所有点的均值并将均值作为质心
```

需要说明的是，在算法中，相似度的计算方法默认的是欧氏距离计算，当然也可以使用其他相似度计算函数，比如余弦距离；算法中，k个类的初始化方式为随机初始化，并且初始化的质心必须在整个数据集的边界之内，这可以通过找到数据集每一维的最大值和最小值；然后最小值+取值范围*0到1的随机数，来确保随机点在数据边界之内。

在实际的K-means算法中，采用计算质心-分配-重新计算质心的方式反复迭代，算法停止的条件是，当然数据集所有的点分配的距其最近的簇不在发生变化时，就停止分配，更新所有簇的质心后，返回k个类的质心(一般是向量的形式)组成的质心列表，以及存储各个数据点的分类结果和误差距离的平方的二维矩阵。

上面返回的结果中，之所以存储每个数据点距离其质心误差距离平方，是便于后续的算法预处理。因为K-means算法采取的是随机初始化k个簇的质心的方式，因此聚类效果又可能陷入局部最优解的情况，局部最优解虽然效果不错，但不如全局最优解的聚类效果更好。所以，后续会在算法结束后，采取相应的后处理，使算法跳出局部最优解，达到全局最优解，获得最好的聚类效果。

4.3. 二分K-means算法

二分K-means算法首先将所有点作为一个簇，然后将簇一分为二。之后选择其中一个簇继续进行划分，选择哪一个簇取决于对其进行划分是否能够最大程度的降低SSE的值。上述划分过程不断重复，直至划分的簇的数目达到用户指定的值为止。

二分K-means算法的伪代码如下：

```
将所有点看成一个簇
当簇数目小于k时
对于每一个簇
      计算总误差
      在给定的簇上面进行k-均值聚类（k=2）
      计算将该簇一分为二之后的总误差
选择使得总误差最小的簇进行划分
```

4.4. java实现

Point.java

```
public class Point {
    //点的坐标
    private Double x;
    private Double y;

    //所在类ID
    private int clusterID = -1;

    public Point() {
    }

    public Point(Double x, Double y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return clusterID + "\t" + x + "\t" + y;
    }

    public Double getX() {
        return x;
    }

    public void setX(Double x) {
        this.x = x;
    }

    public Double getY() {
        return y;
    }

    public void setY(Double y) {
        this.y = y;
    }

    public int getClusterID() {
        return clusterID;
    }

    public void setClusterID(int clusterID) {
        this.clusterID = clusterID;
    }
```

```
}
```

EurDistance.java 欧式距离

```
public class EurDistance {
    public static double getEurDistance(Point p1, Point p2) {

        double disX = p1.getX() - p2.getX();
        double disY = p1.getY() - p2.getY();
        return Math.sqrt(Math.pow(disX, 2) + Math.pow(disY, 2));
    }
}
```

KMeansCluster.java 主要的核心类

```
public class KMeansCluster {
    private int k = 5;
    private int maxIter = 50;


    //测试点集
    public List<Point> points;

    //中心点集
    public List<Point> centers;

    public static final double MINDISTANCE = 10000.00;

    public KMeansCluster(int k, int maxIter, List<Point> points) {
        this.k = k;
        this.maxIter = maxIter;
        this.points = points;

        //初始化中心点
        initCenters();
    }
    /*
        初始化聚类中心
        这里的选举策略是，从点集中按需抽取k个作为初始聚类中心
     */
    public void initCenters() {
        centers = new ArrayList<Point>(k);

        for(int i = 0; i < k; i++) {
            Point tmpPoint = points.get(i * 2);
            Point center = new Point(tmpPoint.getX(), tmpPoint.getY());
            center.setClusterID(i + 1);
            centers.add(center);
        }
    }

    /*
    停止条件时满足迭代次数
     */
    public void runKmeans() {
        //已迭代次数
```

```
        int count = 1;
        while (count++ <= maxIter) {
            //遍历每个点，确定其所属簇
            for (Point point : points) {
                assignPoint2Cluster(point);
            }
        }
        //调整中心点
        adjustCenters();
    }

    /*
        调整聚类中心，按照求平衡点的方法获得新的簇心
     */
    public void adjustCenters() {
        double sumx[] = new double[k];
        double sumy[] = new double[k];

        int count[] = new int[k];

        //保存每个簇的横坐标纸盒
        for(int i = 0; i < k; i++) {
            sumx[i] = 0.0;
            sumy[i] = 0.0;
            count[i] = 0;
        }

        //计算每个簇的横坐标总和、记录每个簇的个数
        for(Point point : points) {
            int clusterID = point.getClusterID();

            sumx[clusterID - 1] += point.getX();
            sumy[clusterID - 1] += point.getY();
            count[clusterID - 1]++;
        }

        //更新簇心坐标
        for(int i = 0; i < k; i++) {
            Point tmpPoint = centers.get(i);
            tmpPoint.setX(sumx[i] / count[i]);
            tmpPoint.setY(sumy[i] / count[i]);
            tmpPoint.setClusterID(i + 1);

            centers.set(i, tmpPoint);
        }

    }

    /*
        划分点到某个簇中，欧式距离标准
     * 对传入的每个点，找到与其最近的簇中心点，将此点加入到簇
     */
    public void assignPoint2Cluster(Point point) {
        double minDistance = MINDISTANCE;

        int clusterID = -1;

        for (Point center : centers) {
```

```
            double dis = EurDistance.getEurDistance(point, center);
            if(dis < minDistance) {
                minDistance = dis;
                clusterID = center.getClusterID();
            }
        }
        point.setClusterID(clusterID);
    }
}
```

Kmeans.java 入口类

```
public class Kmeans {

    // 用来聚类的点集
    public List<Point> points;

    // 将聚类结果保存到文件
    FileWriter out = null;

    // 格式化double类型的输出，保留两位小数
    DecimalFormat dFormat = new DecimalFormat("00.00");

    // 具体执行聚类的对象
    public KMeansCluster kMeansCluster;

    // 簇的数量，迭代次数
    public int numCluster = 5;
    public int numIterator = 200;
    // 点集的数量，生成指定数量的点集
    public int numPoints = 50;

    //聚类结果保存路径
    public static final String FILEPATH = "kmeans/res.txt";

    public Kmeans(int numPoints, int numCluster, int numIterator) {
        this.numCluster = numCluster;
        this.numIterator = numIterator;
        this.numPoints = numPoints;
    }

    public static void main(String[] args) {
        //指定点集个数，簇的个数，迭代次数
        Kmeans kmeans = new Kmeans(100, 5, 200);
        //初始化点集、KMeansCluster对象
        kmeans.init();

        //使用KMeansCluster对象进行聚类
        kmeans.runKmeans();

        kmeans.printRes();
        kmeans.saveResToFile(FILEPATH);
    }

    public void saveResToFile(String filepath) {
        try {
            out = new FileWriter(new File(filepath));
```

```
            for (Point point : points) {
                out.write(String.valueOf(point.getClusterID()));
                out.write("  ");

                out.write(dFormat.format(point.getX()));
                out.write("  ");
                out.write(dFormat.format(point.getY()));
                out.write("\r\n");
            }

            out.flush();
            out.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void printRes() {
        System.out.println("==================Centers-I====================");
        for (Point center : kMeansCluster.centers) {
            System.out.println(center.toString());
        }

        System.out.println("==================Points====================");

        for (Point point : points) {
            System.out.println(point.toString());
        }
    }

    public void runKmeans() {
        kMeansCluster.runKmeans();
    }

    public void init() {
        this.initPoints();
        kMeansCluster = new KMeansCluster(numCluster, numIterator, points);
    }

    //初始化点集
    public void initPoints() {
        points = new ArrayList<Point>(numPoints);

        Point tmpPoint;

        for(int i = 0; i < numPoints; i++) {
            tmpPoint = new Point(Math.random() * 150, Math.random() *  100);
            points.add(tmpPoint);
        }
    }
}
```

效果：

```
"D:\Program Files\Java\64bit\jdk1.8.0_112\bin\java" ...
==================Centers-I====================
1    7.923449912894194    65.51080664526522
2    109.06117820385644   48.41045236762421
3    15.072837288752195   11.937420891229225
4    25.277711436320352   77.1051128072667
5    55.95459106654665    28.17958263785448
==================Points=====================
1    7.674603264202057    68.87750732964172
2    105.27343892295089   0.7340943509140607
2    109.2888302885624    69.14788977848364
2    133.46930429454642   81.04718881949461
3    8.143555229230614    17.905416787329166
4    20.84084492486859    60.98901205219155
4    14.323857346365493   70.39742518954638
1    7.685793372947508    75.98699912758383
5    37.3210700849312 36  28.084106233791893
2    144.50048241638243   49.814192560972614
2    86.59807492057085    95.0624812427857
5    49.952992453794835   12.261677213669508
5    70.22334422634377    36.65722359101875
```

4.5. 总结

1 聚类是一种无监督的学习方法。聚类区别于分类，即事先不知道要寻找的内容，没有预先设定好的目标变量。

2 聚类将数据点归到多个簇中，其中相似的数据点归为同一簇，而不相似的点归为不同的簇。相似度的计算方法有很多，具体的应用选择合适的相似度计算方法

3 K-means聚类算法，是一种广泛使用的聚类算法，其中k是需要指定的参数，即需要创建的簇的数目，K-means算法中的k个簇的质心可以通过随机的方式获得，但是这些点需要位于数据范围内。在算法中，计算每个点到质心得距离，选择距离最小的质心对应的簇作为该数据点的划分，然后再基于该分配过程后更新簇的质心。重复上述过程，直至各个簇的质心不再变化为止。

4 K-means算法虽然有效，但是容易受到初始簇质心的情况而影响，有可能陷入局部最优解。为了解决这个问题，可以使用另外一种称为二分K-means的聚类算法。二分K-means算法首先将所有数据点分为一个簇；然后使用K-means（k=2）对其进行划分；下一次迭代时，选择使得SSE下降程度最大的簇进行划分；重复该过程，直至簇的个数达到指定的数目为止。实验表明，二分K-means算法的聚类效果要好于普通的K-means聚类算法。

- 优点： 1、原理简单（靠近中心点），实现容易 2、聚类效果中上（依赖K的选择） 3、空间复杂度o(N)时间复杂度o(IKN) N为样本点个数，K为中心点个数，I为迭代次数
- 缺点： 1、对离群点， 噪声敏感 （中心点易偏移） 2、很难发现大小差别很大的簇及进行增量计算 3、结果不一定是全局最优，只能保证局部最优（与K的个数及初值选取有关）

5. 回归

5.1. 回归分类

5.1.1. 线性回归

利用大量的样本

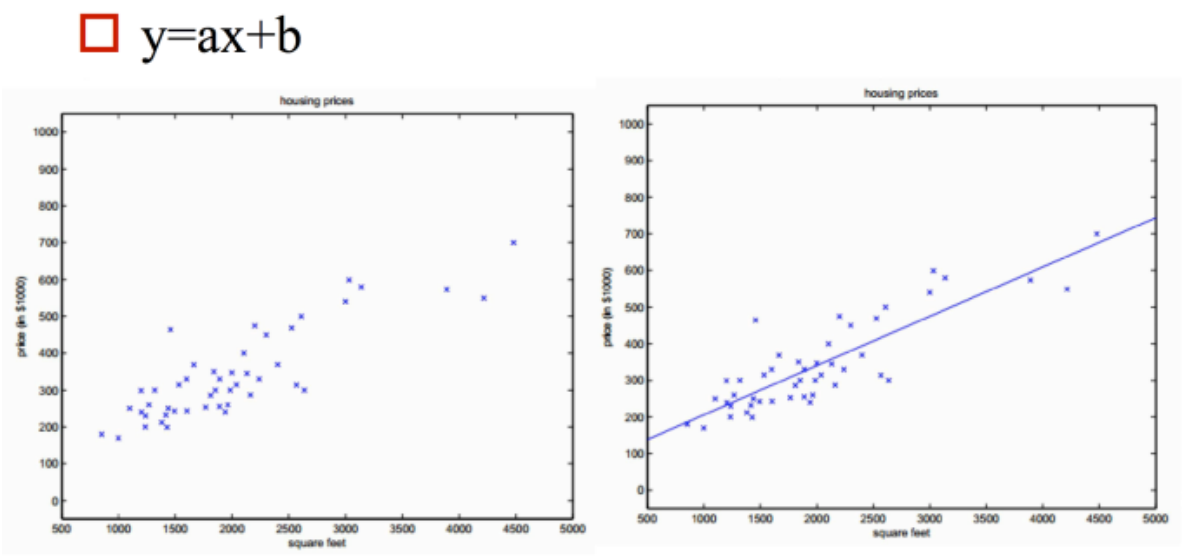
$$D = (x_i, y_i)_{i=1}^N$$

通过有监督的学习，学习到由x到y的映射f ff，利用该映射关系对未知的数据进行预估，因为y为连续值，所以是回归问题。

用一组变量的（特征）的线性组合，来建立与结果之间的关系。 模型表达：

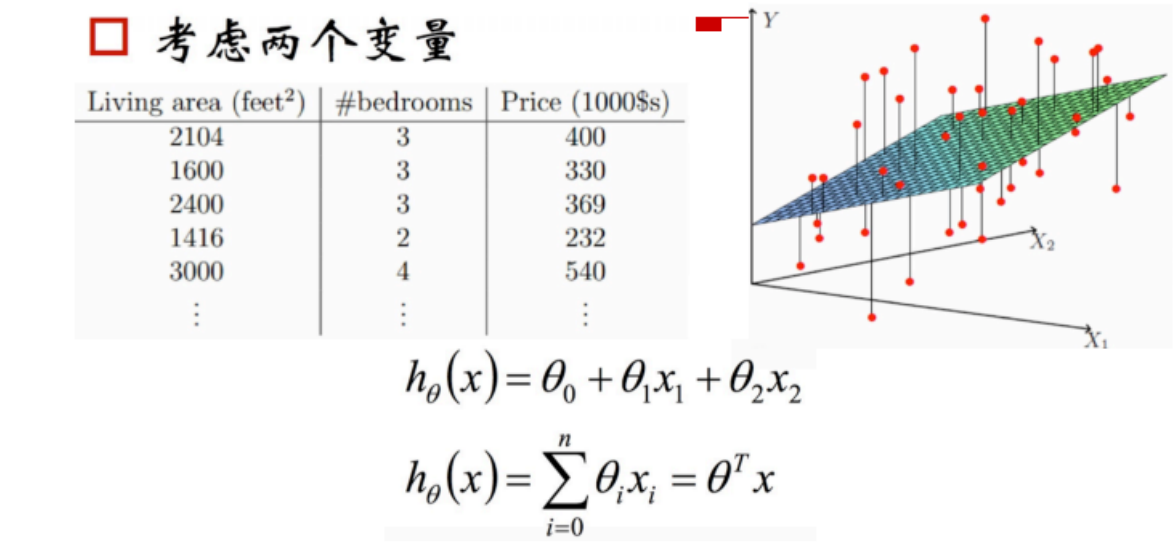
$$y(x, w) = w_0 + w_1 x_1 + \ldots + w_n x_n$$

- 单变量情况：



- 多变量情况

  二维空间的直线，转化为高维空间的平面



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

5.1.2. 逻辑回归

> 监督学习，解决二分类问题。

分类的本质：在空间中找到一个决策边界来完成分类的决策

逻辑回归：线性回归可以预测连续值，但是不能解决分类问题，我们需要根据预测的结果判定其属于正类还是负类。所以逻辑回归就是将线性回归的(−∞,+∞) 结果，通过sigmoid函数映射到(0,1)之间。

线性回归决策函数：

$$h\theta(x) = \theta T x h_\theta(x) = \theta^T x h$$

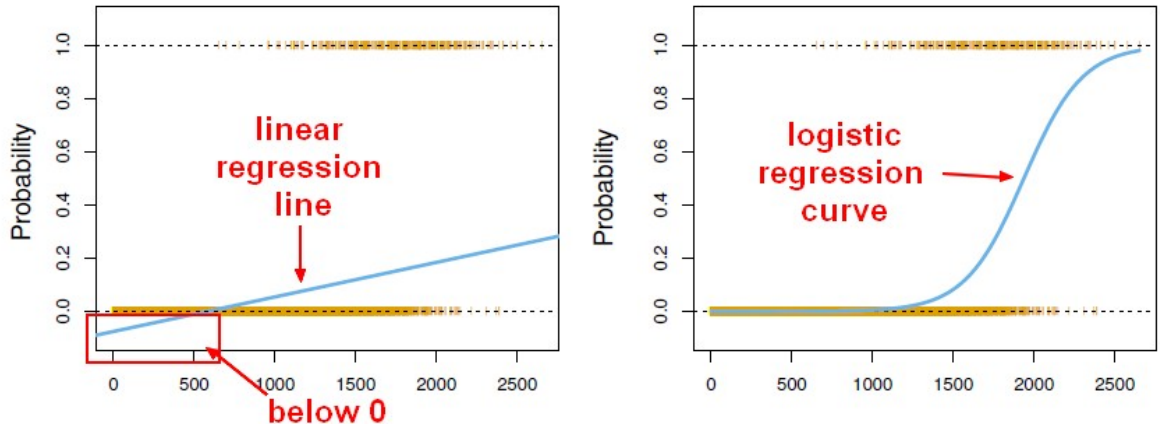
将其通过sigmoid函数，获得逻辑回归的决策函数：

$$h\theta(x) = 11 + e - \theta T x h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} h$$

5.1.3. 逻辑回归与线性回归的关系

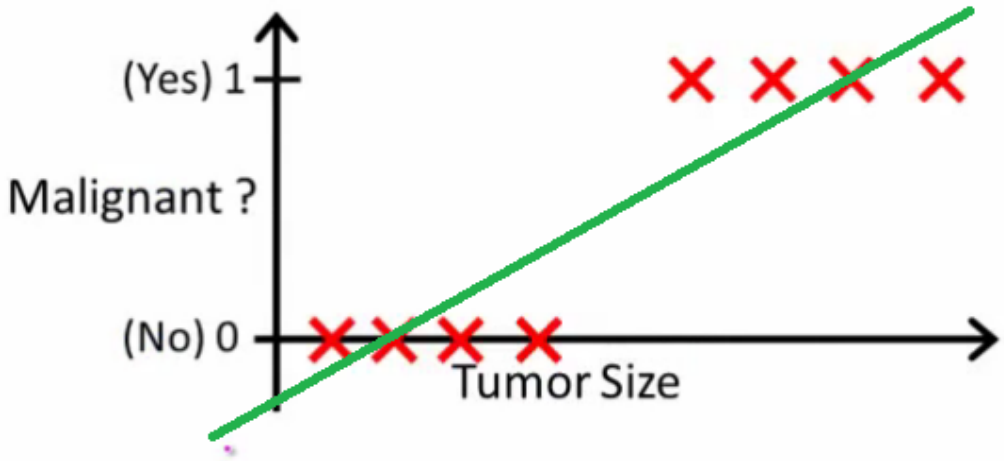
逻辑回归与线性回归都属于广义线性回归模型,其区别与联系从以下几个方面比较：

- 分类与回归:回归模型就是预测一个连续变量(如降水量，价格等)。在分类问题中，预测属于某类的概率，可以看成回归问题。这可以说是使用回归算法的分类方法。
- 输出:直接使用线性回归的输出作为概率是有问题的，因为其值有可能小于0或者大于1,这是不符合实际情况的，逻辑回归的输出正是[0,1]区间。见下图，



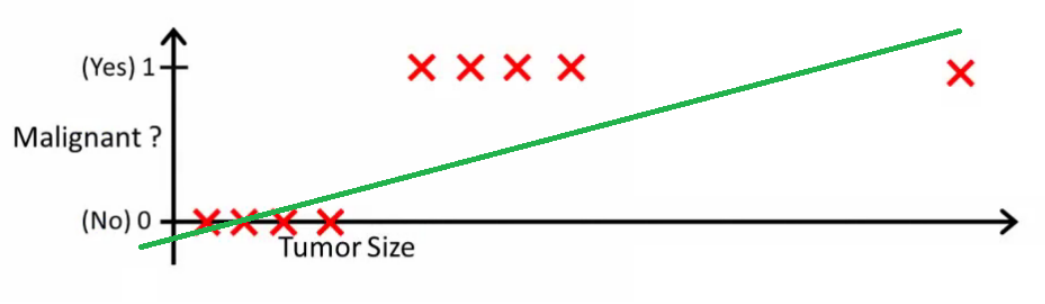
- 参数估计方法：

1. 线性回归中使用的是最小化平方误差损失函数，对偏离真实值越远的数据惩罚越严重。这样做会有什么问题呢？假如使用线性回归对{0,1}二分类问题做预测，则一个真值为1的样本，其预测值为50，那么将会对其产生很大的惩罚，这也和实际情况不符合，更大的预测值说明为1的可能性越大，而不应该惩罚的越严重。
2. 逻辑回归使用对数似然函数进行参数估计，使用交叉熵作为损失函数，对预测错误的惩罚是随着输出的增大，逐渐逼近一个常数，这就不存在上述问题了1
3. 也正是因为使用的参数估计的方法不同，线性回归模型更容易受到异常值(outlier)的影响，有可能需要不断变换阈值(threshold),线性回归分类的情况见下面两图:
    1. 无异常值的线性回归情况:

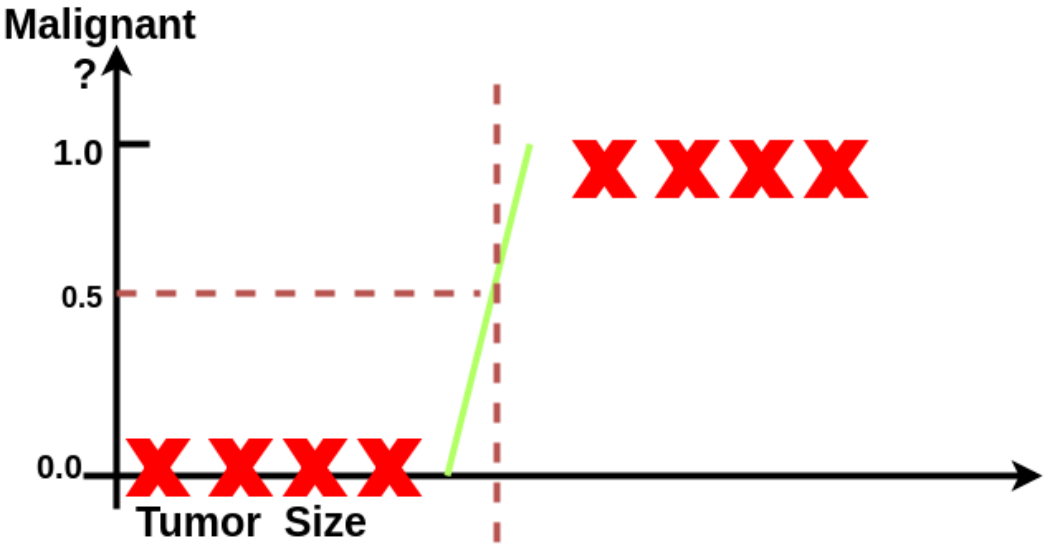


蓝线为求得的h(x)，上图中可选阈值为0.5作为判断肿瘤是否是良性。

2. 有异常值的线性回归情况:



这个时候再想有好的预测效果需调整阈值为0.2，才能准确预测。



3. 使用逻辑回归的方法进行分类，就明显对异常值有较好的稳定性。如下图:

- 参数解释: 线性回归中，独立变量的系数解释十分明了，就是保持其他变量不变时，改变单个变量因变量的改变量。 逻辑回归中，自变量系数的解释就要视情况而定了，要看选用的概率分布是什么，如二项式分布，泊松分布等

5.2. 代码实现

线性回归和逻辑回归的实现大体一致，将其抽象出一个抽象类Regression，包含整体流程，其中有三个抽象函数，将在线性回归和逻辑回归中重写。 将样本设为Sample类，其中采用数组作为特征的存储形式。

```
package com.desheng.bigdata.regression;

public class Sample {
    public double[] features; //抽取的特征
    public int feaNum; // the number of sample's features 特征个数
    public double value; // value of sample in regression 计算出的样本的值
    public int label; // class of sample   //回归的分类

    public Sample(int number) {
        feaNum = number;
        features = new double[feaNum];
    }

    public void showSample() {
        System.out.println("The sample's features are:");
        for(int i = 0; i < feaNum; i++) {
```

```
            System.out.print(features[i] + "\t");
        }
        System.out.println("\nThe label is: " + label);
        System.out.println("The value is: " + value);
    }
}
```

Regression抽象父类：

```
package com.desheng.bigdata.regression;

public abstract class Regression {
    protected double[] theta; //parameters
    protected int paraNum; //the number of parameters
    protected double rate; //learning rate
    protected Sample[] sam; // samples
    protected int samNum; // the number of samples
    protected double th; // threshold value

    /**
     * initialize the samples
     * @param s : training set
     * @param num : the number of training samples
     */
    public void initialize(Sample[] s, int num) {
        samNum = num;
        sam = new Sample[samNum];
        for(int i = 0; i < samNum; i++) {
            sam[i] = s[i];
        }
    }

    /**
     * initialize all parameters
     * @param para : theta
     * @param learning_rate
     * @param threshold
     */
    public void setPara(double[] para, double learning_rate, double threshold) {
        paraNum = para.length;
        theta = para;
        rate = learning_rate;
        th = threshold;
    }

    /**
     * predicate the value of sample s
     * @param s : prediction sample
     * @return : predicted value
     */
    public abstract double predicateVal(Sample s);

    /**
     * calculate the cost of all samples
     * @return : the cost
     */
    public abstract double costFunc();
```

```
    /**
     * update the theta
     */
    public abstract void update();

    public void outputTheta() {
        System.out.println("The parameters are:");
        for(int i = 0; i < paraNum; i++) {
            System.out.print(theta[i] + "\t");
        }
        System.out.println(costFunc());
    }
}
```

5.2.1. 线性回归

```
package com.desheng.bigdata.regression.liner;

import com.desheng.bigdata.regression.Regression;
import com.desheng.bigdata.regression.Sample;
public class LinearRegression extends Regression {
    public double predicateVal(Sample s) {
        double val = 0;
        for(int i = 0; i < paraNum; i++) {
            val += theta[i % theta.length] * s.features[i % s.features.length];
        }
        return val;
    }

    public double costFunc() {
        double sum = 0;
        for(int i = 0; i < samNum; i++) {
            double d = predicateVal(sam[i]) - sam[i].value;
            sum += Math.pow(d, 2);
        }
        return sum / (2*samNum);
    }

    public void update() {
        double former = 0; // the cost before update
        double latter = costFunc(); // the cost after update
        double[] p = new double[paraNum];
        do {
            former = latter;
            //update theta
            for(int i = 0; i < paraNum; i++) {
                // for theta[i]
                double d = 0;
                for(int j = 0; j < samNum; j++) {
                    d += (predicateVal(sam[j]) - sam[j].value) *
sam[j].features[i];
                }
                p[i] -= (rate * d) / samNum;
            }
            theta = p;
            latter = costFunc();
```

```
            if(former - latter < 0){
                System.out.println("α is larger!!!");
                break;
            }
        }while(former - latter > th);
    }
}
```

5.2.2. 逻辑回归

```
package com.desheng.bigdata.regression.logistic;

import com.desheng.bigdata.regression.Regression;
import com.desheng.bigdata.regression.Sample;

public class LogisticRegression extends Regression {

    public double predicateVal(Sample s) {
        double val = 0;
        for(int i = 0; i < paraNum; i++) {
            val += theta[i % theta.length] * s.features[i % s.features.length];
        }
        return 1/(1 + Math.pow(Math.E, -val));
    }

    public double costFunc() {
        double sum = 0;
        for(int i = 0; i < samNum; i++) {
            double p = predicateVal(sam[i]);
            double d = Math.log(p) * sam[i].label + (1 - sam[i].label) *
Math.log(1 - p);
            sum += d;
        }
        return -1 * (sum / samNum);
    }

    public void update() {
        double former = 0; // the cost before update
        double latter = costFunc(); // the cost after update
        double[] p = new double[paraNum];
        do {
            former = latter;
            //update theta
            for(int i = 0; i < paraNum; i++) {
                // for theta[i]
                double d = 0;
                for(int j = 0; j < samNum; j++) {
                    d += (predicateVal(sam[j]) - sam[j].value) *
sam[j].features[i];
                }
                p[i] -= (rate * d) / samNum;
            }
            latter = costFunc();
            if(former - latter < 0){
                System.out.println("α is larger!!!");
                break;
            }
```

```
        }while(former - latter > th);
        theta = p;
    }
}
```

5.2.3. 测试

5.2.3.0. 测试数据

liner.txt

```
2104 5 1 45 460
1416 3 2 40 232
1534 3 2 30 315
852 2 1 36 178
1254 3 3 45 321
987 2 2 35 241
1054 3 2 30 287
645 2 3 25 87
542 2 1 30 94
1065 3 1 25 241
2465 7 2 50 687
2410 6 1 45 654
1987 4 2 45 436
457 2 3 35 65
587 2 2 25 54
468 2 1 40 87
1354 3 1 35 215
1587 4 1 45 345
1789 4 2 35 325
2500 8 2 40 720
```

regression.txt

```
0.23 0.35 0
0.32 0.24 0
0.6 0.12 0
0.36 0.54 0
0.02 0.89 0
0.36 -0.12 0
-0.45 0.62 0
0.56 0.42 0
0.4 0.56 0
0.46 0.51 0
1.2 0.32 1
0.6 0.9 1
0.32 0.98 1
0.2 1.3 1
0.15 1.36 1
0.54 0.98 1
1.36 1.05 1
0.22 1.65 1
1.65 1.54 1
0.25 1.68 1
```

5.2.3.1. 线性回归测试

```
package com.desheng.bigdata.regression.test;

import com.desheng.bigdata.regression.Sample;
import com.desheng.bigdata.regression.liner.LinearRegression;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.RandomAccessFile;

/**
 * x0 x1 x2 x3 x4 y
 *
 * 2104 5 1 45 460
   1416 3 2 40 232
   1534 3 2 30 315
 */
public class TestLinerRegression {

    public static void main(String[] args) throws IOException {
        //read Sample.txt
        Sample[] sam = new Sample[25];

        long filePoint = 0;
        String s;
        BufferedReader br = new BufferedReader(new
FileReader("regression/liner.txt"));
        int count = 0;
        String line = null;
        while((line = br.readLine()) != null) {
            //s --> sample
            String[] sub = line.split("\\s+");
            sam[count] = new Sample(sub.length);
            for(int i = 0; i < sub.length; i++) {
                if(i == sub.length - 1) {
                    sam[count].value = Double.parseDouble(sub[i]);
                }
                else {
                    sam[count].features[i] = Double.parseDouble(sub[i]);
                }
            }
            count++;
        }//while read file

        LinearRegression lr = new LinearRegression();
        double[] para = {0,0,0,0,0};
        double rate = 0.5;//参数
        double th = 0.001;//阈值
        lr.initialize(sam, count);
        lr.setPara(para, rate, th);
        lr.update();
        lr.outputTheta();
    }
}
```

```
"D:\Program Files\Java\64bit\jdk1.8.0_112\bin\java" ...
α is larger!!!
The parameters are:
267398.525   691.55   257.175 6051.95 0.0 8.1277296103249824E16
```

5.2.3.2. 逻辑回归测试

```
package com.desheng.bigdata.regression.test;

import com.desheng.bigdata.regression.Sample;
import com.desheng.bigdata.regression.logistic.LogisticRegression;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/*
x0 x1 x2 class
 */
public class TestLogisticRegression {
    public static void main(String[] args) throws IOException {
        //read Sample.txt
        Sample[] sam = new Sample[25];

        BufferedReader br = new BufferedReader(new
FileReader("regression/logistic.txt"));
        int count = 0;//行号

        String line = null;
        while((line = br.readLine()) != null) {
            //s --> sample
            String[] sub = line.split("\\s+");
            sam[count] = new Sample(sub.length);
            for(int i = 0; i < sub.length; i++) {
                if(i == sub.length - 1) {
                    sam[count].label = Integer.parseInt(sub[i]);
                }
                else {
                    sam[count].features[i] = Double.parseDouble(sub[i]);
                }
            }//for
            //sam[w].outSample();
            count++;
        }//while read file

        LogisticRegression lr = new LogisticRegression();
        double[] para = {0,0,0};
        double rate = 0.5;
        double th = 0.001;
        lr.initialize(sam, count);
        lr.setPara(para, rate, th);
        lr.update();
        lr.outputTheta();
    }
}
```

```
"D:\Program Files\Java\64bit\jdk1.8.0_112\bin\java" ...
The parameters are:
-0.11687499999999999    -0.198625    0.0 0.7489948776906945

Process finished with exit code 0
```