# 金融大数据处理技术：实验4

杨浩然 161220156

## 1 实验目的

互联网技术不断发展，给人类带来了更快速的信息传播媒介。在这个互联网时代，不仅是时事新闻，股市新闻传播地也更加快速。股市新闻中往往包含了大量信息，除了上市公司的财务数据外，还包括经营公告、行业动向、国家政策等大量文本信息，这些文本信息中常常包含了一定的情感倾向，会影响股民对公司股票未来走势的预期，进一步造成公司的股价波动。如果能够挖掘出这些新闻中蕴含的情感信息，则可以对股票价格进行预测，对于指导投资有很大的作用。本实验尝试使用文本挖掘技术和机器学习算法，挖掘出新闻中蕴含的情感信息，分别将每条新闻的情感判别为“positive”、“neutral”、“negative”这三种情感中的一种，可根据抓取的所有新闻的情感汇总分析来对股票价格做预测。

## 2 实验流程

#### 2.1 流程概述

本实验首先对给定的训练集和测试集进行预处理，运用MapReduce框架提高处理速率，获取能供后续实验步骤使用的特定数据格式。然后使用处理好的训练集数据，结合K-最近邻算法和朴素贝叶斯分类器来对测试集数据进行情感判别。

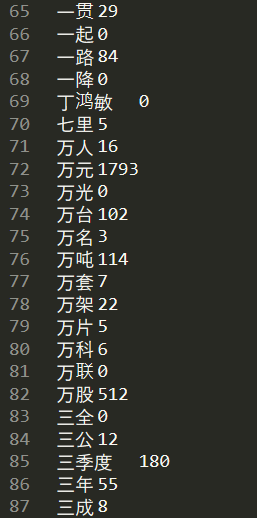
#### 2.2 数据预处理

###### 2.2.1 训练集预处理

由于使用了给定的chi\_words.txt中的词语作为特征，因此，第一步要做的是就是统计chi\_words.txt中各个词在训练集中出现的频次，该步骤使用MapReduce框架，以提高数据处理的速率，核心代码如下：

1. **public** **static** **class** BagMapper **extends** Mapper<Object, Text, Text, IntWritable> {
2. @Override
3. **public** **void** map(Object key, Text value, Context context)
4. **throws** IOException, InterruptedException {
5. String content = value.toString();
6. System.out.println(content);
7. List<Word> words = WordSegmenter.seg(content.replaceAll("[a-zA-Z0-9]", ""));
9. **for** (**int** i = 0; i < chi\_words.size(); i++){
10. Text newKey = **new** Text();
11. IntWritable newValue = **new** IntWritable();
12. newKey.set(chi\_words.get(i));
13. newValue.set(0);
14. context.write(newKey, newValue);
15. }
17. **for** (**int** i = 0; i < words.size(); i++){
18. **if** (chi\_words.contains(words.get(i).toString())){
19. Text newKey = **new** Text();
20. IntWritable newValue = **new** IntWritable();
21. newKey.set(words.get(i).toString());
22. newValue.set(1);
23. context.write(newKey, newValue);
24. }
25. }
26. }
27. }

经过预处理后，得到训练集“词袋”，格式如下：



利用上述步骤得到的“词袋”文件，我们可以利用其生成文本向量，文本向量的每一个分量由特征词的TF-IDF值表示，TF-IDF值计算步骤如下：

1. 词频(TF) = 某个特征词在文本中出现的频数/文本中含有的特征词总数；
2. 逆文档频率(IDF) = ；
3. 计算TF-IDF = 。

核心代码如下：

1. List<Word> words = WordSegmenter.seg(content.replaceAll("[a-zA-Z0-9]", ""));
2. **for** (Word word : words){
3. **int** count = 0;
4. ArrayList<String> sub\_words = **new** ArrayList<String>();
6. **if**(chi\_words.contains(word.toString())){
7. sub\_words.add(word.toString());
8. **for** (**int** i = 0; i < words.size(); i++){
9. **if** (word.toString().equals(words.get(i).toString())) {
10. count = count + 1;
11. }
12. }
14. **double** tf = count/sub\_words.size();
15. **double** idf = Math.log(sum/frequency.get(chi\_words.indexOf(word.toString())));
16. **double** tf\_idf = tf \* idf;
18. vector.set(chi\_words.indexOf(word.toString()), tf\_idf);
19. }
20. }

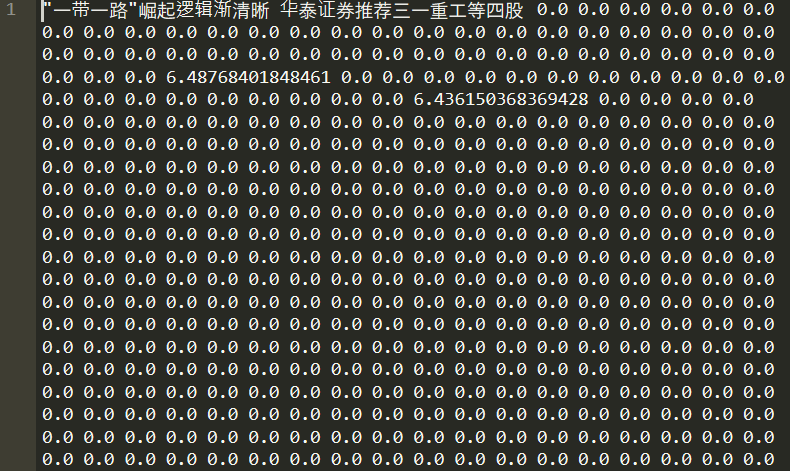
经过预处理后，得到的文本向量格式为：



前半部分是训练集的文件名和情感类别（1表示positive，0表示neutral，-1表示negative），后半部分是文本向量，共由2978个维度，代表着chi\_words.txt中2978个特征词。

###### 2.2.2 测试集预处理

训练集预处理其思想和流程与训练集一样，只是在生成文本向量时，与训练集的处理略有不一致，具体代码和流程不再赘述，生成的文本向量格式如下，前半部分是测试集中的新闻标题文本，后半部分是文本向量，共2978维：



#### 2.3 使用K-最近邻算法给测试集分类

###### 2.3.1 K-最近邻算法介绍

KNN算法全名为k-Nearest Neighbor，就是K最近邻的意思。KNN也是一种分类算法。但是与之前说的决策树分类算法相比，这个算法算是最简单的一个了。算法的主要过程为：

1、给定一个训练集数据，每个训练集数据都是已经分好类的。

2、计算给定的初始数据到训练集所有数据的欧几里得距并排序；

3、选出训练集中离a距离最近的K个训练集数据；

4、比较k个训练集数据，选出里面出现最多的分类类型，此分类类型即为最终测试数据a的分类；

###### 2.3.2 核心代码

该部分的功能是，从训练集的文本向量中找出与测试集某一文本向量最相似的前15个，然后统计这15个已知类别的文本向量分别所属的类别，15个文本向量中哪个类别所占数目最多，我们就可以把该测试集文本向量也归为这一类。

1. **for** (String each : vectors) {
2. String[] target\_vector = each.split(" ");
3. **double** sub\_score = 0;
5. **if** (own\_vector.length != target\_vector.length) {
6. System.out.println("............................................");
7. System.out.println(own\_vector.length);
8. System.out.println(target\_vector.length);
9. System.out.println(value.toString().split("\t")[0]);
10. }
11. **for** (**int** i = 0; i < own\_vector.length; i++){
12. **double** own = Double.parseDouble(own\_vector[i]);
13. **double** target = Double.parseDouble(target\_vector[i]);
15. sub\_score = sub\_score + Math.pow((own - target), 2);
16. }
18. sub\_score = Math.sqrt(sub\_score);
19. **for** (**int** i = 0; i < kneighbor.size(); i++) {
20. **if** (kneighbor.get(i) < sub\_score) {
21. kneighbor.set(i, sub\_score);
22. emotion.set(i, Integer.parseInt(emotions.get(vectors.indexOf(each))));
23. }
24. }
25. }

###### 2.3.3 部分计算结果展示

#### 2.4 使用朴素贝叶斯分类器给测试集分类

###### 2.4.1朴素贝叶斯分类器介绍

朴素贝叶斯分类的正式定义：

1. 设为一个待分类项，而每个a为x的一个特征属性；
2. 有类别集合；
3. 计算；
4. 如果，则。

对于上述步骤3中的各个条件概率计算方法如下：

1. 找到一个已知分类的待分类集合，这个集合叫作训练样本；
2. 统计得到在各类别下各个特征属性的条件概率估计，即：
3. 如果各个特征属性是条件独立的，则根据贝叶斯定理与如下推到：

###### 2.4.2 核心代码

首先是计算2.4.1中步骤3的代码：

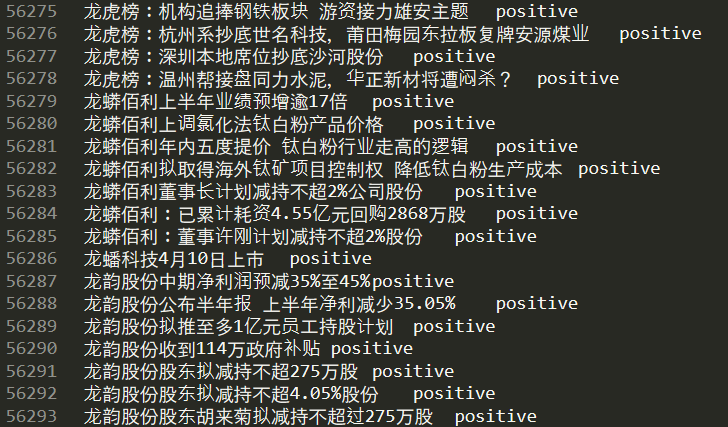
1. **int** count = 0;
2. **try** {
3. Scanner scanner = **new** Scanner(file1);
4. **while** (scanner.hasNextLine()) {
5. count = count + 1;
6. String str = scanner.nextLine();
7. String emotion = str.split("\t")[0].split(",")[1];
8. String vector = str.split("\t")[1];
10. **if** (emotion.equals("1"))
11. pos.add(vector);
12. **else** **if** (emotion.equals("0"))
13. neu.add(vector);
14. **else** **if** (emotion.equals("-1"))
15. neg.add(vector);
16. }
17. scanner.close();
18. } **catch** (FileNotFoundException e) {
19. e.printStackTrace();
20. }
22. p\_pos = pos.size()/count;
23. p\_neu = neu.size()/count;
24. p\_neg = neg.size()/count;

然后是利用上述代码求出的概率计算2.4.1中的步骤4：

1. **double** p1 = p\_pos;
2. **double** p2 = p\_neu;
3. **double** p3 = p\_neg;
5. **for** (**int** i = 0; i < sub\_vector.length; i++) {
6. **double** num = Double.parseDouble(sub\_vector[i]);
7. **if** (num > 0) {
8. p1 = p1 \* probabilities\_pos.get(i);
9. p2 = p2 \* probabilities\_neu.get(i);
10. p3 = p3 \* probabilities\_neg.get(i);
11. }
12. **else** {
13. p1 = p1 \* (1-probabilities\_pos.get(i));
14. p2 = p2 \* (1-probabilities\_neu.get(i));
15. p3 = p3 \* (1-probabilities\_neg.get(i));
16. }
17. }
19. **if** (p1 >= p2 && p1 >= p3)
20. result[count] = title + "\t" + "positive";
21. **else** **if** (p2 >= p1 && p2 >= p3)
22. result[count] = title + "\t" + "neutral";
23. **else** **if** (p3 >= p1 && p3 >= p2)
24. result[count] = title + "\t" + "negative";

###### 2.4.3 部分计算结果展示

统计结果如下所示，前部分为新闻标题文本内容，后半部分是通过算法进行的情感判别结果：



## 3 实验总结

通过本次实验，本人大致了解了文本挖掘的流程，学习使用了KNN、朴素贝叶斯等机器学习算法对文本进行情感判别，学习了如何进行模型训练，如何进行分类预测。

在此次实验中，进行KNN算法的计算时速度太慢，经分析可知，本人在代码中使用了几处嵌套循环，大大提高了计算量，在数据庞大的情况下，程序运行速度极为缓慢。为了解决该问题，本人查阅了Java API手册，通过寻找一些已经封装好的方法或者类来代替嵌套循环，通过这样处理，程序运行速度有明显的提高。

另外，此次实验的分类结果并没有预期中的好，本人认为有如下原因：

* 1. 训练集规模太小；
  2. 训练集是较为丰富的新闻文本，测试集是较为抽象的新闻标题，两者用词上面有较大的差异；
  3. 特征选取不精确；
  4. 使用的分类算法都较为简单，无法取得很好的效果。