MapReduce 高级编程实验报告

151180045 侯汶昕

1. 实验主题、背景、目标及要求

- 主题：上市公司财经新闻情感分析
- 背景：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。
- 实验目标：使用多种机器学习算法对文本进行情感判别，包括 KNN、决策树、Naive Bayes、支持向量机等，学习如何进行模型训练，如何进行分类预测。要求使用至少两种分类方法。
- 要求：核心程序在 MapReduce 上运行，要求使用至少两种分类方法。

2. 实验数据集说明：

- 实验 3 数据集 download_data.zip
- 样本数据集 training_data.zip：分别保存在 negative、neutral 和 positive 三个文件夹下的样本数据，文件夹名即其中样本的分类。
- 可以参考 chi_words.txt，将其中的词语作为特征，并计算特征向量。也可以自己选择特征。

3. 实验重点与难点

1. 训练集和测试集的预处理，特征选择及提取
2. 编写基于 MapReduce 的程序实现基于 TF-IDF 的 Naive Bayes 算法并对测试集进行分类
3. 编写基于 MapReduce 的程序实现基于 TF-IDF 的 KNN 算法并对测试集进行分类

4. 实验环境

实验中使用了两台虚拟机，虚拟机具有相同的环境和配置：

系统：ubuntu 16.04 LTS 64-bit

内存：1.9GB

5. 实验设计思路

该实验要求用两种分类方法实现对测试集的情感分类，经过查阅相关资料和比较不同的方法之后，决定使用 Naïve Bayes 和 KNN 两种算法在 MapReduce 上实现，特征选用 TF-IDF。

5.1 特征提取和数据预处理

本实验依据 TF-IDF 来选择和提取特征。TF-IDF 是一种统计方法，用以评估某个词对其所在文档的重要程度。字词的重要性随着它在文档中出现的次数成正比增加，但同时会随着它在文档集合中出现的频率成反比下降。

词频 TF（Term Frequency）是指某个词在文档中的出现频率，倒文档频率 IDF（Inverse Document Frequency）反映包含该词的文档数占文档总数的比例的倒数。

TF-IDF 的计算公式为：

$$TF = \frac{\text{在文档中某个词出现的次数}}{\text{该文档的所有词条数目}}$$

$$IDF = log\frac{\text{语料库中文档总数}}{\text{包含该词的文档数} + 1}$$

$$TF \cdot IDF = TF * IDF$$

所以说，TF-IDF 能够体现的是：一个词在某个文档中出现次数越多，同时在所有文档中“普及率”越低，就说明该词越能代表该文档。因此，这里我选用了 TF-IDF 来构成特征向量，即用一个针对整个训练集提取出的词表来构建每个训练（或测试）样本的 TF-IDF 向量。

实验中采用了基于 Python 语言的 sklearn.feature_extraction.text 库中的 TfidfTransformer, CountVectorizer 和 TfidfVectorizer 实现从文本到 TF-IDF（或者词频）向量的转换。使用 jiaba 库进行中文分词，停用词表使用实验 3 给出的 stopword.txt。

为了节省存储和内存空间，输出为稀疏矩阵的形式。

稀疏矩阵格式（对每个向量）：

Feature1:Weight1, Feature12:Weight12, …

训练数据为 negative, neutral, positive 下的文件，输出格式为：

index, 类别, TF-IDF 向量

测试数据使用 fulldata.txt，输出格式为：

index, TF-IDF 矩阵（KNN）/ 词频矩阵（Naïve Bayes）

5.2 基于 TF-IDF 的 Naïve Bayes 算法设计

Naïve Bayes (Naive Bayes) 是一种经典的分类算法，其方法是假设样本特征相互独立，通过贝叶斯公式来计算待预测样本属于某一类的概率，最终输出最大概率所属的类别。

实验中为了利用 TF-IDF 特征，对经典的 Naive Bayes 算法做了一些调整。修改后的基于 TF-IDF 贝叶斯算法如下所示：

$$P(y_j|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|y_j)P(y_j)}{P(\boldsymbol{x})} \propto \prod_{i=1}^{n} P(x_i|y_j)P(y_j)$$

$$P(y_j) = j\text{类样本}占总数\text{比例} = \frac{第j类样本的个数}{训练样本总数} \Rightarrow 第j类样本的个数$$

$$P(x_i|y_j) = 特征i在样本j中所占比例 = \frac{特征i在样本j中出现的个数}{所有特征在样本j中出现的总数}$$

$$\Rightarrow \frac{第i个特征在标签为j的样本中的\ \mathrm{TF}\cdot\mathrm{IDF}\ 值}{所有特征在标签为j的样本中的\ \mathrm{TF}\cdot\mathrm{IDF}\ 和}$$

最后预测的样本标签为：

$$\hat{y_j} = \mathrm{argmax_j}(\mathrm{P(y_j)}|\boldsymbol{x})$$

其中$P(y_j)$表示 j 类样本占总数比例，$P(x_i|y_j)$表示特征i在样本j中所占比例。在预测过程中，因为测试样本出现的概率$P(\boldsymbol{x})$是一个常数，所以计算的时候可以省略。

由于三类样本概率$P(y_j)$的计算公式中具有相同的分母，因此为了减少计算量，并且提高精度，实验中直接用式中分子，即第 j 类样本的个数表示$P(y_j)$。

实验所用算法的主要改变在于对于每个类别下不同特征概率$P(x_i|y_j)$的计算，经典的 Naive Bayes 算法是基于离散特征出现的频率来计算这个概率的，而实际上每个类别乃至每个文章中可能同一个词的出现一次所代表的重要性都是不一样的。因此，本实验中，为了利用 TF-IDF 含有的词“重要性”的信息，将其作为一个权重来计算特征概率。也就是说，用每个词在每篇文章中的“重要性”即 TF-IDF 值代替原来的单纯计 1 进行统计，使最终计算得到的特征概率包含文章信息。

基于 TF-IDF 的 Naïve Bayes 算法流程图如下图所示：

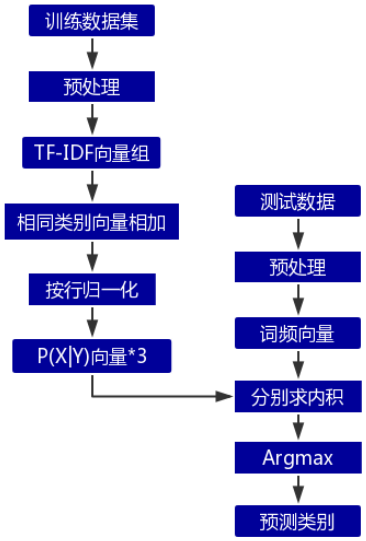


图 1：基于 TF-IDF 的 Naive Bayes 算法流程图

* 实验中使用的 TF-IDF 特征向量由选取 TF-IDF 值最高的 5000 个特征词组成，并且要求所有词的文档频率不得大于 0.5，因为太大的文档频率代表这个词可能十分常见而不具有代表性。

5.3 基于 TF-IDF 的 KNN 算法设计

KNN（K-Nearest Neighbor，K 近邻）算法是一种十分简单的分类算法，并且往往十分有效。缺点是计算复杂度太大。

KNN 的算法流程：

1. 分别计算训练数据集和测试数据集的 TF-IDF 矩阵。

对于每个测试样本：

2. 计算其与所有训练样本的特征向量之间的距离。
3. 选取距离最小的 K 个训练样本作为近邻。
4. 返回前 K 个点中出现频率最高的类别作为该测试数据的预测分类。

由于 KNN 计算复杂度太高，采用 5000 维的 TF-IDF 特征向量计算时间过长，因此仅使用了 1000 维，筛选条件和 Naïve Bayes 中相同。实验中采用 Euclidean Distance（欧式距离）：

$$Distance(\boldsymbol{y_j}, \boldsymbol{x}) = \sqrt{\sum_{i=1}^{1000} \left(y_{j,i} - x_i\right)^2}$$

基于 TF-IDF 的 KNN 算法流程图如下图所示：

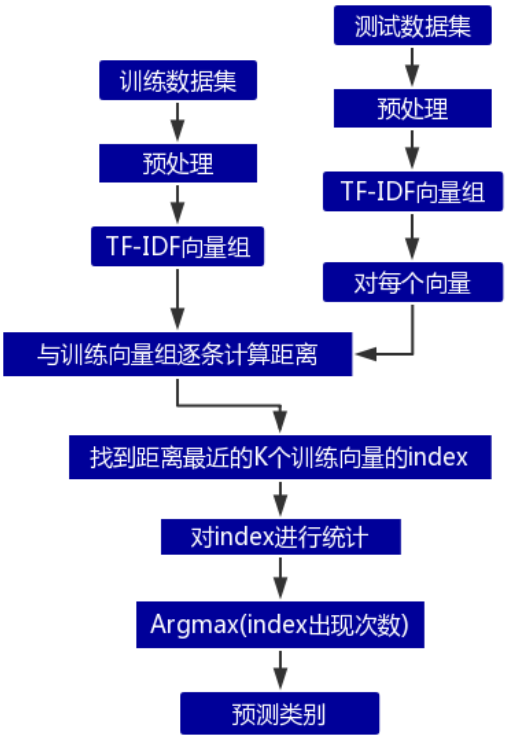


图 2：基于 TF-IDF 的 KNN 算法流程图

6. 重要类的设计说明及代码

6.1 基于 MapReduce 的 Naïve Bayes 算法实现

6.1.1 TrainMapper 类

主要功能：

- 对读入的每一个训练样本，拆分特征对。
- 输出键值对：<label, weight>（用于计算所有特征在某个标签下的样本中的 TF・IDF 和）和<label#feature, weight>（用于计算某个特征在某个标签的样本中的 TF・IDF 值）。

```
public static class TrainMapper extends Mapper<Object, Text, Text, DoubleWritable> {
        private Text outputKey = new Text();
        private DoubleWritable outputValue = new DoubleWritable();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] tmp = value.toString().split(", ");
            String label;
            String[] featurePair;
            String feature;
            double weight;
            label = tmp[1];
            for(int i = 2; i < tmp.length; i++){
                featurePair = tmp[i].split(":");
                feature = featurePair[0];
                weight = Double.parseDouble(featurePair[1]);
                outputKey.set(label + "#" + feature);
                outputValue.set(weight);
                context.write(outputKey, outputValue);
                // To calculate P(Y)
                outputKey.set(label);
                context.write(outputKey, outputValue);
            }
        }
}
```

6.1.2 TrainReducer 类

主要功能：

- 对 label 的 weight 求和，得到每个类别所有特征的权重和，用于对 label#feature 归一。
- 对 label#feature 的 weight 求和，得到每个类别中各个特征的权重概率（$P(x_i|y_j)$）。

```
public static class TrainReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable>
{
        DoubleWritable outputValue = new DoubleWritable();
        public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws
IOException, InterruptedException {
                double sum = 0;
                for(DoubleWritable value : values){
                    sum += value.get();
                }
                outputValue.set(sum);
                context.write(key, outputValue);
        }
}
```

6.1.3 TestMapper 类

主要功能：

- Setup:载入训练模型和相关信息（三个类别的样本个数）
- LoadModel 类将训练得到的数据载入为 HashMap 形式（Key: label#feature, Value: Weight）
- LoadInfo 类将$P(y_i)$载入为 HashMap 形式（Key: label, Value: Count）。
- 对每个测试样本$\boldsymbol{x}$，计算并得到三个类中最大的$P(\boldsymbol{x}|y_i)$，对应的y_i即为预测的类别。
- 输出键值对：<index, prediction>

```
public static class TestMapper extends Mapper<Object, Text, Text, Text> {
        public LoadModel model;
        public LoadInfo info;
        public void setup(Context context) throws IOException {
            Configuration conf = context.getConfiguration();
            conf.set("model", "model/part-r-00000");
            model = new LoadModel();
            model.getData(conf);
            info = new LoadInfo();
            conf.set("info", "info.txt");
            info.getInfo(conf);
        }
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            Text outputKey, outputValue;      //key: id, value: class
            String[] vals;
            String temp;
```

```
            double count;
            double Pxjyi, PXyi, normFactor, Pyi;
            vals = value.toString().split(", ");
            double maxP = -100;
            int idx = -2;
            for(int i = -1; i < 2; i++) { // For every class: negative(-1), neutral(0), positive(1)
                PXyi = 1; // Initial Probability
                normFactor = model.model_dict.get(String.valueOf(i));
                Pyi = info.Pys.get(String.valueOf(i));
                for(int j = 1; j < vals.length; j++) {
                    String feature = vals[j].split(":")[0];
                    count = Double.parseDouble(vals[j].split(":")[1]);
                    temp = String.valueOf(i) + "#" + feature;
                    //System.out.println(temp);
                    Double weight = model.model_dict.get(temp);
                    // To avoid nullpointer exception: P(xj | yi) does not exist.
                    if(weight == null) {
                        Pxjyi = 0;
                    }
                    else {
                        Pxjyi = count * weight.doubleValue() / normFactor;
                    }
                    PXyi = PXyi * Pxjyi;
                }
                if(PXyi * Pyi > maxP) {
                    maxP = PXyi * Pyi;
                    idx = i;
                }
            }
            outputKey = new Text(vals[0]);
            outputValue = new Text(String.valueOf(idx));
            context.write(outputKey, outputValue);
        }
}
```

6.1.4 LoadModel 类（LoadInfo 类和其类似）：

主要功能：读入训练后的模型数据，并在测试时将训练得到的数据载入为 HashMap 形式（Key: label#feature, Value: Weight）。

```
public class LoadModel {
    public HashMap<String, Double> model_dict;
```

```
    public LoadModel() {
        model_dict = new HashMap<String, Double>();
    }
    public void getData(Configuration conf) throws IOException {
        String line;
        String modelPath = conf.get("model");
        Path path = new Path(modelPath);
        FileSystem hdfs = path.getFileSystem(conf);
        FSDataInputStream fin = hdfs.open(path);
        InputStreamReader inReader = new InputStreamReader(fin);
        BufferedReader bfReader = new BufferedReader(inReader);
        while((line = bfReader.readLine()) != null) {
            String[] res = line.split("\\t");
            model_dict.put(res[0], new Double(res[1]));
            System.out.println(res[0] + ":" + new Double(res[1]));
        }
        bfReader.close();
        inReader.close();
        fin.close();
    }
}
```

6.2 基于 MapReduce 的 KNN 算法实现

6.2.1 LoadTrainData 类

主要功能：

- 加载训练数据，以 HashMap 的格式进行存储。
- 提供接口，返回每个训练样本转换成 ArrayList 格式的向量，便于计算距离。

```
public class LoadTrainData {
    public HashMap<String, Double> featureDict;
    public HashMap<String, String> labelDict;
    public int size;
    public int featureNum;

    public LoadTrainData() {
        featureDict = new HashMap<String, Double>();
        labelDict = new HashMap<String, String>();
        size = 0;
        featureNum = 0;
    }
```

```
public void getData(Configuration conf) throws IOException {
    String line;
    String index, label;
    String feature;
    double weight;
    featureNum = Integer.parseInt(conf.get("featureNum"));
    String filePath = conf.get("trainDataPath");
    Path path = new Path(filePath);
    FileSystem hdfs = path.getFileSystem(conf);
    FSDataInputStream fin = hdfs.open(path);
    InputStreamReader inReader = new InputStreamReader(fin);
    BufferedReader bfReader = new BufferedReader(inReader);
    while((line = bfReader.readLine()) != null) {
        String[] wholeData = line.split(", ");
        index = wholeData[0];
        label = wholeData[1];
        labelDict.put(index, label);
        for(int i = 2; i < wholeData.length; i++) {
            feature = wholeData[i].split(":")[0];
            weight = Double.parseDouble(wholeData[i].split(":")[1]);
            String temp = index + "#" + feature;
            featureDict.put(temp, weight);
        }
        size++;
    }
    bfReader.close();
    inReader.close();
    fin.close();
}
public ArrayList<Double> getWeights(int index) {
    //Format of featureDict: {index#feature: weight}
    ArrayList<Double> weights = new ArrayList<Double>(featureNum);
    String temp;
    for(int i = 0; i < featureNum; i++) {
        temp = String.valueOf(index) + "#" + String.valueOf(i);
        Double weight = featureDict.get(temp);
        if(weight == null) {
            weights.add(0.0);
        }
        else {
            weights.add(weight);
        }
```

```
        }
        return weights;
    }
}
```

6.2.2 KNNMapper 类

主要功能：

- Setup: 载入训练数据集并将每个训练样本转化为 ArrayList 形式，方便后续计算距离。
- 将测试样本读入 HashMap 并将其转化为 ArrayList 形式。
- 将测试样本与整个训练集计算距离（getDistance 函数），得到 K 个最近邻（getKNN 函数）。
- 输出 K 个最近邻标签供 Reducer 统计。
- 输出键值对：<index, trainIndex#trainLabel>。

```
public static class KNNMapper extends Mapper<Object, Text, Text, Text> {
        public LoadTrainData trainData;
        public ArrayList<ArrayList<Double>> trainDataWeights;
        public void setup(Context context) throws IOException {
            Configuration conf = context.getConfiguration();
            conf.set("trainDataPath", "traindata/traindata.txt");
            trainData = new LoadTrainData();
            trainData.getData(conf);
            trainDataWeights = new ArrayList<ArrayList<Double>>(trainData.size);
            for(int i = 0; i < trainData.size; i++){
                trainDataWeights.add(i, trainData.getWeights(i));
            }
        }
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            Configuration conf = context.getConfiguration();
            int k = Integer.parseInt(conf.get("k"));

            String[] featurePair;
            String feature;
            double weight;
            ArrayList<Double> testDataWeights = new
ArrayList<Double>(trainData.featureNum);
            HashMap<String, Double> dataDict = new HashMap<String, Double>();
            String[] line = value.toString().split(", ");
            String index = line[0];
            String mode = conf.get("mode");
```

```
            if(mode.equals("test")) {
                for(int i = 1; i < line.length; i++) {
                    featurePair = line[i].split(":");
                    feature = featurePair[0];
                    weight = Double.parseDouble(featurePair[1]);
                    dataDict.put(feature, weight);
                }
            }
            else if(mode.equals("validate")) {
                for(int i = 2; i < line.length; i++) {
                    featurePair = line[i].split(":");
                    feature = featurePair[0];
                    weight = Double.parseDouble(featurePair[1]);
                    dataDict.put(feature, weight);
                }
            }
            for(int i = 0; i < trainData.featureNum; i++) {
                Double tempWeight = dataDict.get(String.valueOf(i));
                if(tempWeight == null) {
                    testDataWeights.add(0.0);
                }
                else {
                    testDataWeights.add(tempWeight);
                }
            }
            ArrayList<String> KNN = getKNN(trainDataWeights, testDataWeights, k);
            for(int i = 0; i < k; i++) {
                String trainDataIndex = KNN.get(i);
                String trainDataLabel = trainData.labelDict.get(trainDataIndex);
                context.write(new Text(index), new Text(trainDataIndex + "#" +
trainDataLabel));
            }
        }

        private ArrayList<String> getKNN(ArrayList<ArrayList<Double>> trainDataWeights,
ArrayList<Double> testDataWeights, int k) {
            ArrayList<String> KNN = new ArrayList<String>(k);
            ArrayList<Double> distances = new ArrayList<Double>(k);
            double distance;
            for(int i = 0; i < k; i++) {
                KNN.add("-2");
                distances.add(Double.MAX_VALUE);
```

```
            }
            for(int i = 0; i < trainData.size; i++) {
                distance = getDistance(trainDataWeights.get(i), testDataWeights,
trainData.featureNum);
                for(int j = 0; j < k; j++) {
                    if(distance < distances.get(j)) {
                        distances.set(j, distance);
                        KNN.set(j, String.valueOf(i));
                        break;
                    }
                }
            }
            return KNN;
        }
        private double getDistance(ArrayList<Double> vector1, ArrayList<Double> vector2, int
dimension) {
            double distance = 0.0;
            double numerator = 0.0;
            double denominator1 = 0.0;
            double denominator2 = 0.0;
            for(int i = 0; i < dimension; i++) {
                distance = distance + Math.pow((vector1.get(i) - vector2.get(i)), 2);
            }
            distance = Math.sqrt(distance);
            return distance;
        }
}
```

6.2.3 KNNReducer 类

主要功能：

- 输入键值对：<index, trainIndex#trainLabel> （每个测试样本对应 K 个）。
- 对每个 trainLabel 计数，以 HashMap 的形式存储（Key: trainLabel, Value: Count）。
- 按 Count 对 trainLabel 进行排序，并得到最大 Count 对应的 trainLabel 作为预测类别。
- 输出键值对：<index, prediction>。

```
public static class KNNReducer extends Reducer<Text, Text, Text, Text> {
        @Override //如果 Reducer 没用上可以在这里加上@Override 检查 reduce 是否有错
误
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            String indexOfTrainData, label;
```

```
            HashMap<String, Integer> counter = new HashMap<String, Integer>();
            for(Text value : values) {
                if(counter.containsKey(value.toString())) {
                    counter.put(value.toString(), counter.get(value.toString()) + 1);
                }
                else {
                    counter.put(value.toString(), new Integer(1));
                }
            }
            // 找到出现次数最多的 train data 对应的 index
            List<Map.Entry<String,Integer>> list = new ArrayList(counter.entrySet());
             Collections.sort(list, (o1, o2) -> (o1.getValue() - o2.getValue()));
             indexOfTrainData = list.get(list.size() - 1).getKey();
             label = indexOfTrainData.split("#")[1];
             context.write(new Text(key), new Text(label));
        }
}
```

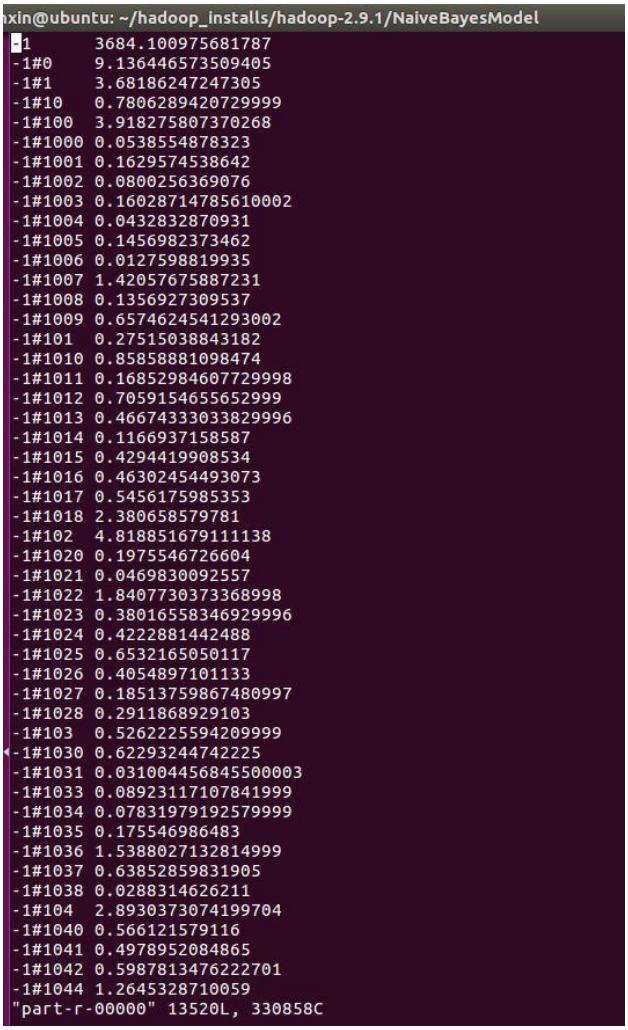
7. 实验结果及截图

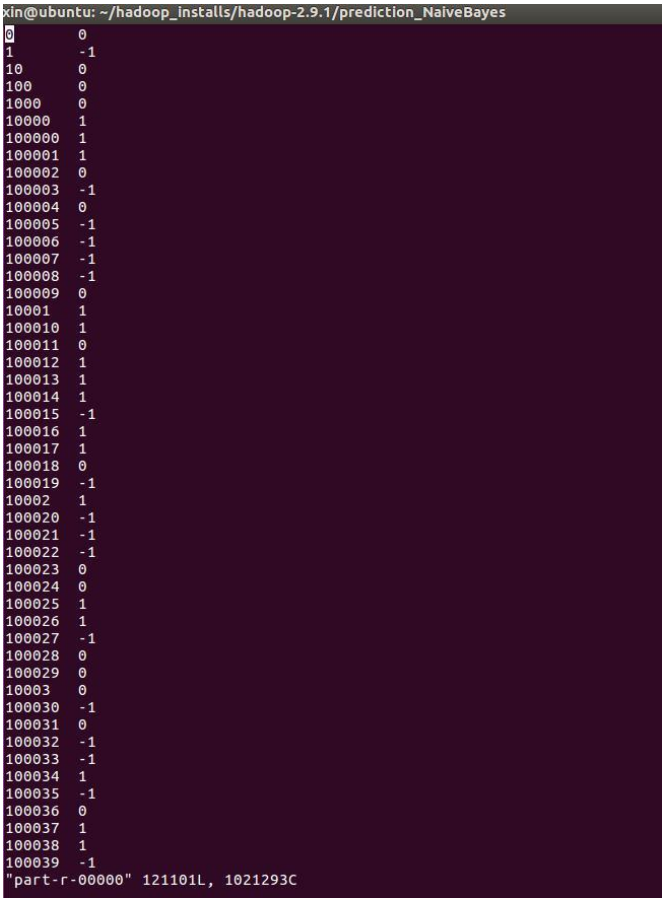
7.1 基于 Naïve Bayes 算法的分类结果

Naïve Bayes 的训练过程：

```
houwenxin@ubuntu:~/hadoop_installs/hadoop-2.9.1$ bin/hdfs dfs -put /home/houwenxin/Desktop/FBDP/Exp_4/traindata.txt traindata
houwenxin@ubuntu:~/hadoop_installs/hadoop-2.9.1$ bin/hdfs dfs -put /home/houwenxin/Desktop/FBDP/Exp_4/testdata.txt testdata
houwenxin@ubuntu:~/hadoop_installs/hadoop-2.9.1$ bin/hdfs dfs -put /home/houwenxin/Desktop/FBDP/Exp_4/info.txt ./
houwenxin@ubuntu:~/hadoop_installs/hadoop-2.9.1$ bin/hadoop jar /home/houwenxin/Desktop/HWX_NaiveBayes.jar NaiveBayesMain traindata model train
18/12/19 21:57:15 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
18/12/19 21:57:15 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
18/12/19 21:57:17 INFO input.FileInputFormat: Total input files to process : 1
18/12/19 21:57:17 INFO mapreduce.JobSubmitter: number of splits:1
18/12/19 21:57:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local712794378_0001
18/12/19 21:57:27 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
18/12/19 21:57:27 INFO mapreduce.Job: Running job: job_local712794378_0001
18/12/19 21:57:28 INFO mapred.LocalJobRunner: OutputCommitter set in config null
18/12/19 21:57:28 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
18/12/19 21:57:28 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
18/12/19 21:57:28 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
18/12/19 21:57:28 INFO mapred.LocalJobRunner: Waiting for map tasks
18/12/19 21:57:29 INFO mapred.LocalJobRunner: Starting task: attempt_local712794378_0001_m_000000_0
18/12/19 21:57:29 INFO mapreduce.Job: Job job_local712794378_0001 running in uber mode : false
18/12/19 21:57:29 INFO mapreduce.Job:  map 0% reduce 0%
18/12/19 21:57:29 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
18/12/19 21:57:29 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
18/12/19 21:57:30 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
18/12/19 21:57:30 INFO mapred.MapTask: Processing split: hdfs://host-0:9000/user/houwenxin/traindata/traindata.txt:0+4375149
18/12/19 21:57:40 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
18/12/19 21:57:40 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
18/12/19 21:57:40 INFO mapred.MapTask: soft limit at 83886080
18/12/19 21:57:40 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
18/12/19 21:57:40 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
18/12/19 21:57:40 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
18/12/19 21:57:43 INFO mapred.LocalJobRunner:
18/12/19 21:57:43 INFO mapred.MapTask: Starting flush of map output
18/12/19 21:57:43 INFO mapred.MapTask: Spilling map output
18/12/19 21:57:43 INFO mapred.MapTask: bufstart = 0; bufend = 5146702; bufvoid = 104857600
18/12/19 21:57:43 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 24590344(98361376); length = 1624053/6553600
18/12/19 21:57:44 INFO mapred.MapTask: Finished spill 0
18/12/19 21:57:44 INFO mapred.Task: Task:attempt_local712794378_0001_m_000000_0 is done. And is in the process of committing
18/12/19 21:57:44 INFO mapred.LocalJobRunner: map
18/12/19 21:57:44 INFO mapred.Task: Task 'attempt_local712794378_0001_m_000000_0' done.
```

训练出的模型（左）和测试集预测结果（右）：+1 表示积极，-1 表示消极，0 表示中性

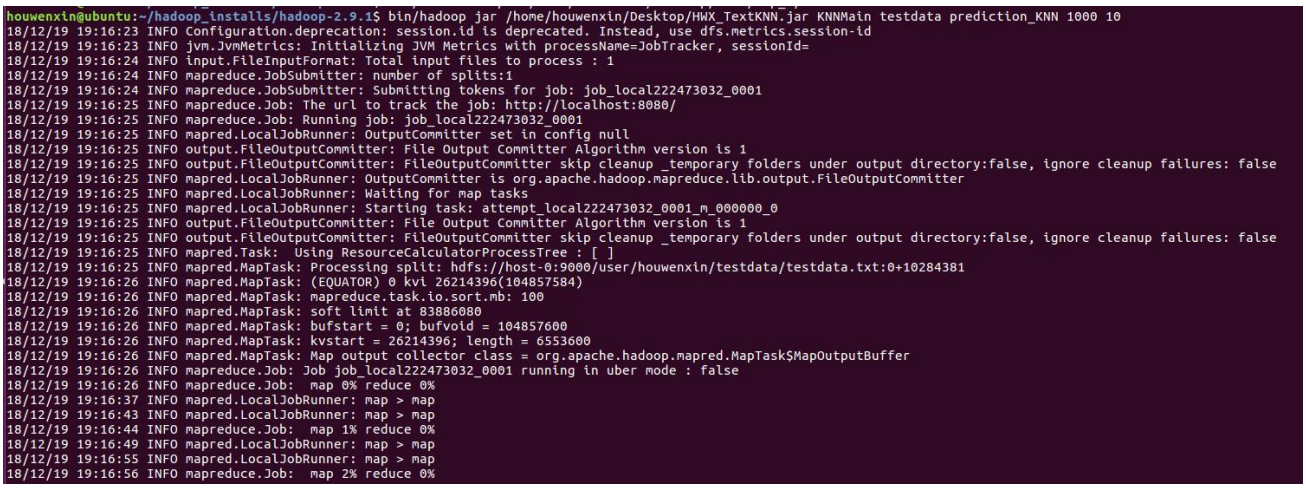




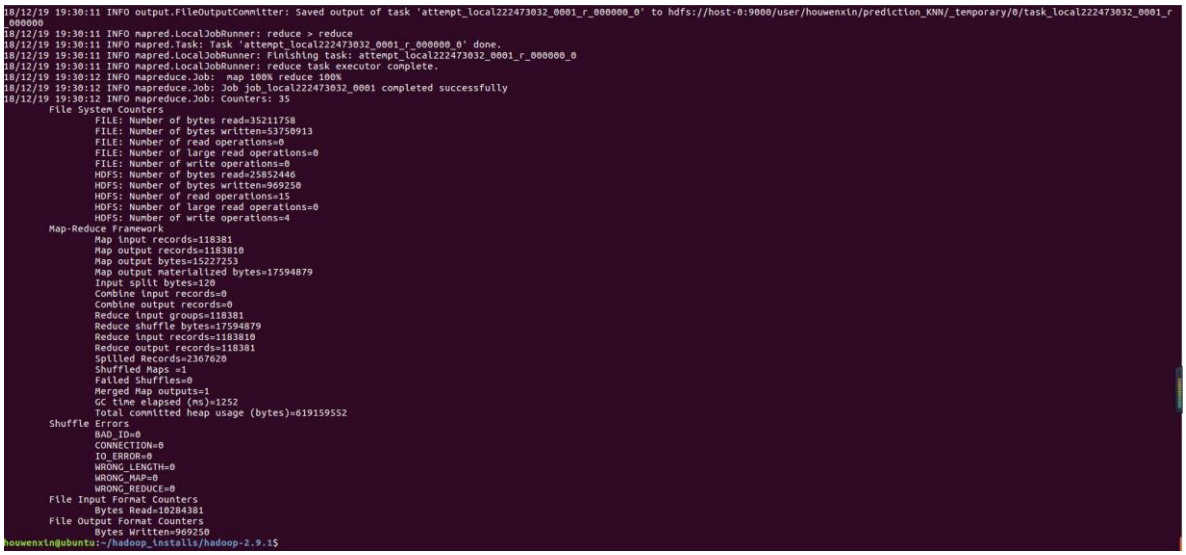
经过简单的人工比对，我发现 Naïve Bayes 的分类结果还算是比较准确的，而且由于使用了稀疏矩阵，计算速度也很快。

7.2 基于 KNN 算法的分类结果

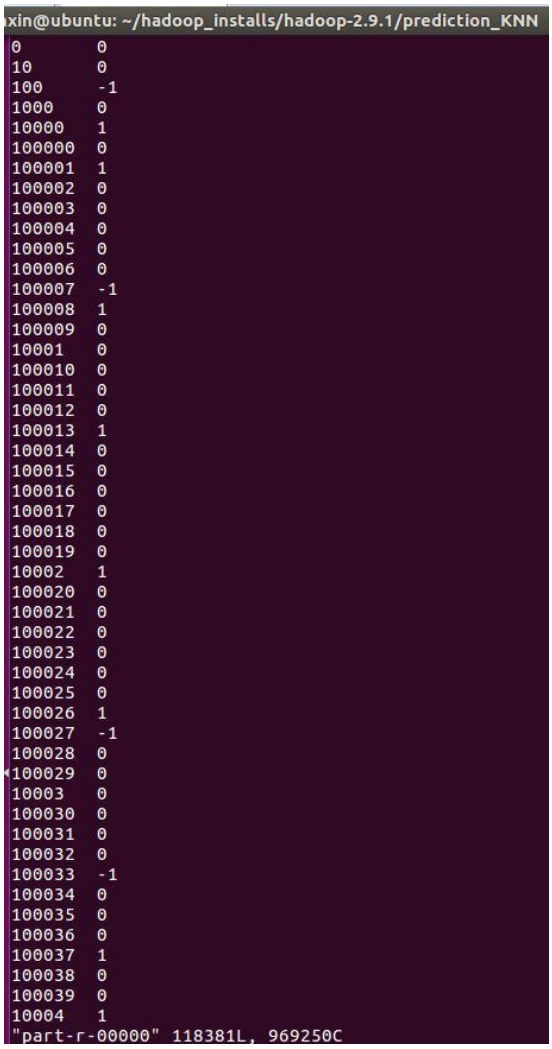
KNN 的训练开始截图：



KNN 的训练结束截图：



测试集的预测结果：



可以看出，KNN 的计算时间很长，测试集提取 1000 维的特征向量的时候测试使用了将近 15 分钟的时间。而且预测结果也不是很好，有很多中性的预测。我推测主要原因是是 K 值选

取的还是不够好，然后就是 TF-IDF 矩阵没有归一化，导致有的比较有特点的特征词的 TF-IDF 值特别高，所以很容易就被排除在 K 个以外，被选出的大部分都是没有特点的中性样本，所以就会容易预测为中性。

8. 实验中可以改进之处

首先，从实验结果即可看出，对于 KNN 算法来说，可以尝试提取不同维度的特征，选取多种 K 值和 TF-IDF 矩阵的归一化来提高 KNN 的算法表现。并且虽然预处理后的数据，使用的稀疏矩阵的方式存储的方式提高了存储和 Naïve Bayes 的运行效率，可是对于 KNN 的算法，在计算欧式距离时又要将其转化为 ArrayList 的全矩阵形式，反而提高了计算复杂度，未来可以考虑一个可以使用稀疏矩阵直接计算 KNN 的算法。

这个实验中，特征词的提取全部是依赖 TF-IDF 和 DF（词文档频率）的大小来选取的，并没有做人工筛选，可能如果使用一份人工筛选的包含金融术语的词表能够更好的反映特征。

本实验使用了 Python 和 Java 两个程序语言编写，所以整体性不是很好。因而运行的时候需要手动运行多次，不是很方便，而且预测时的结果展示形式以 index 来表示测试样本也不够直观，未来可以尝试使用 MapReduce 把预处理的程序也实现一下，达到“一键预处理+训练/预测”的效果。

最后，由于时间原因，我仅仅写了一个简单的准确率验证程序，还没有来得及编写交叉验证程序，未来可以编写并运行交叉验证得到更准确的算法评估效果。