第2章-智能分类

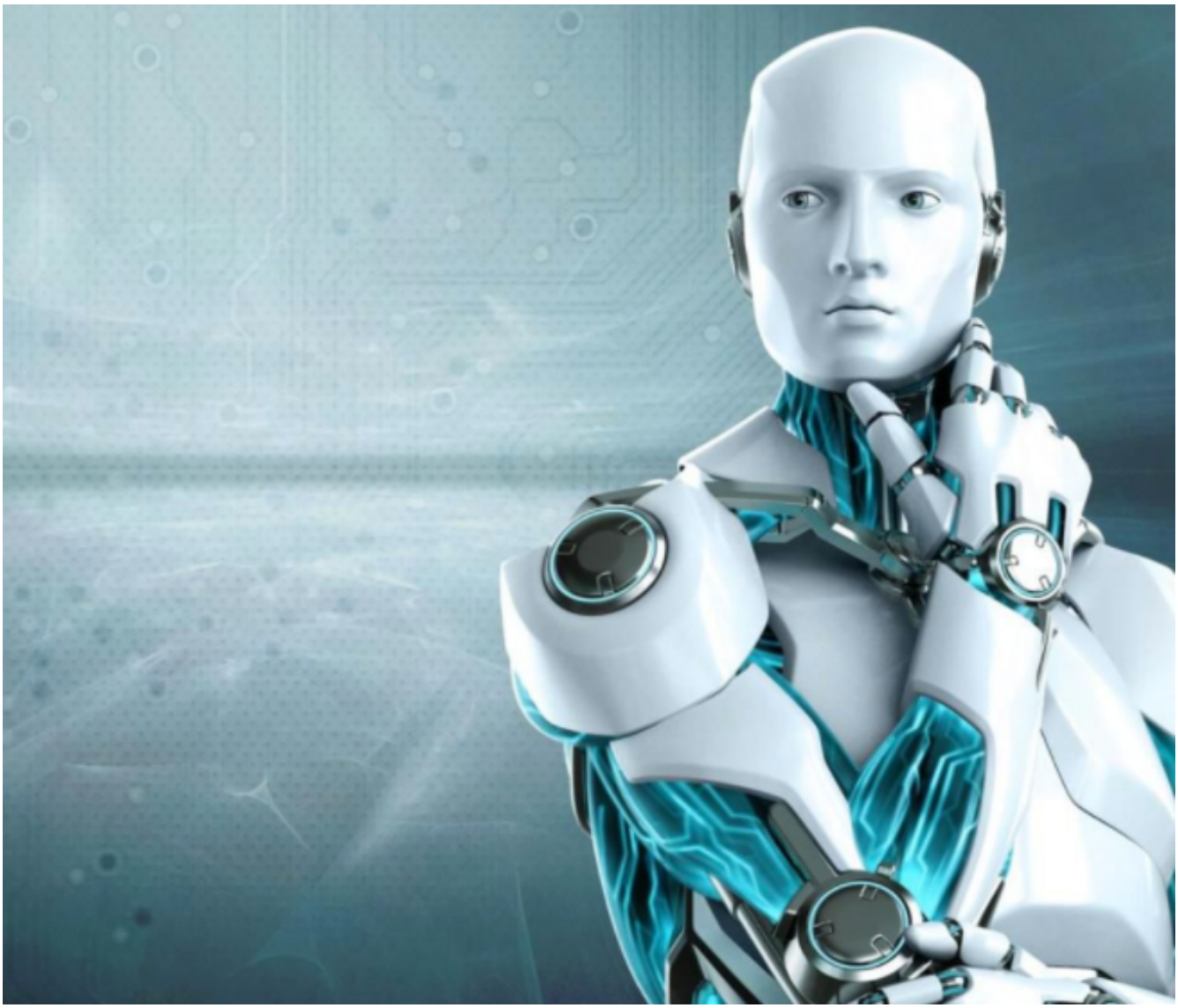
学习目标：

-
-

1 人工智能与机器学习

1.1 谈谈人工智能

人工智能（Artificial Intelligence），英文缩写为AI。它是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学。



人工智能是计算机科学的一个分支，它企图了解智能的实质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器，该领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等。人工智能从诞生以来，理论和技术日益成熟，应用领域也不断扩大，可以设想，未来人工智能带来的科技产品，将会是人类智慧的“容器”。人工智能可以对人的意识、思维的信息过程的模拟。人工智能不是人的智能，但能像人那样思考、也可能超过人的智能。

1.2 人工智能的三次浪潮

1956 Artificial Intelligence提出

- 1950-1970 符号主义流派：专家系统占主导地位
  - 1950：图灵设计国际象棋程序
  - 1962：IBM 的跳棋程序战胜人类高手（人工智能第一次浪潮）
- 1980-2000统计主义流派：主要用统计模型解决问题
  - 1993： SVM模型
  - 1997：IBM 深蓝战胜象棋选手卡斯帕罗夫（人工智能第二次浪潮）
- 2010-至今神经网络、深度学习、大数据流派
  - 2006 DNN（深度神经网络）
  - 2016：Google AlphaGO 战胜围棋选手李世石（人工智能第三次浪潮）

1.3 机器学习

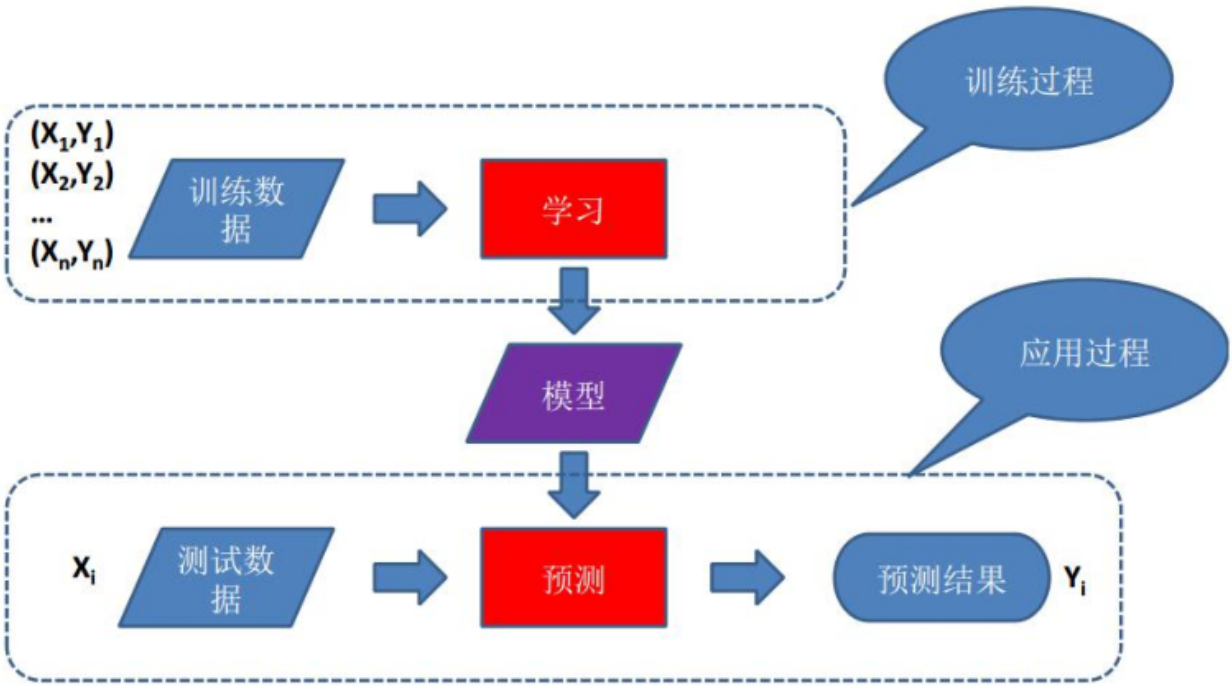
1.3.1 什么是机器学习

机器学习，它正是这样一门学科，它致力于研究如何通过计算（CPU和GPU计算）的手段，利用经验来改善（计算机）系统自身的性能。

它是人工智能的核心，是使计算机具有智能的根本途径，应用遍及人工智能各领域。

数据+ 机器学习算法= 机器学习模型

有了学习算法我们就可以把经验数据提供给它，它就能基于这些数据产生模型。



1.3.2 人工智能、机器学习、深度学习的关系

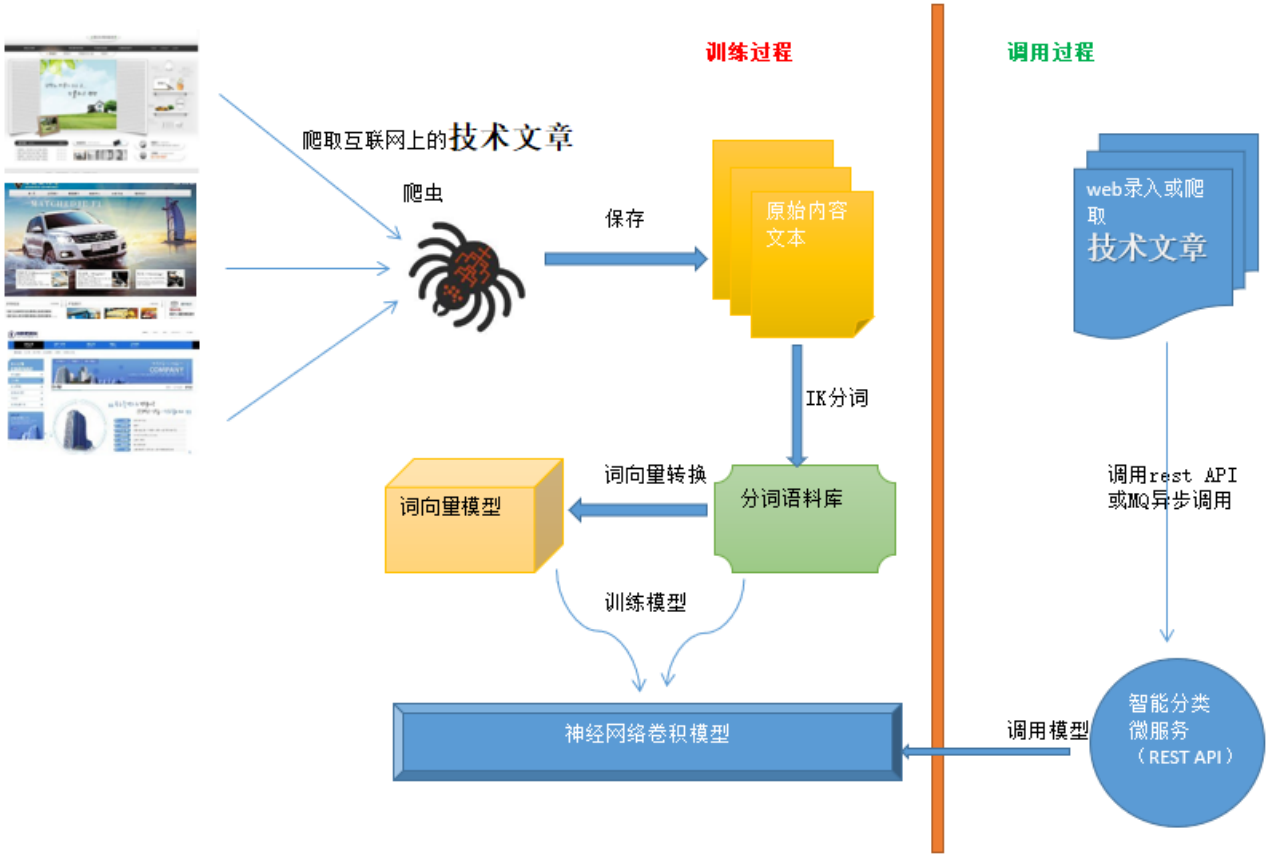
机器学习是人工智能的一个分支，深度学习是实现机器学习的一种技术。

2 智能分类

2.1 需求分析

通过机器学习，当用户录入一篇文章或从互联网爬取一篇文章时可以预测其归属的类型。

2.2 智能分类的执行流程



3 IK分词器

3.1 IK分词器简介

IK Analyzer 是一个开源的,基于java 语言开发的轻量级的中文分词工具包。从2006年12 月推出1.0 版开始,IKAnalyzer 已经推出了4 个大版本。最初,它是以开源项目Luence为应用主体的,结合词典分词和文法分析算法的中文分词组件。从 3.0版本开始,IK发展为面向 Java的公用分词组件,独立于 Lucene项目,同时提供了对 Lucene的默认优化实现。
在 2012版本中,IK实现了简单的分词歧义排除算法,标志着 IK分词器从单纯的词典分词向模拟语义分词衍化。

3.2 快速入门

（1）创建工程引入依赖

```
        <dependency>
            <groupId>com.janeluo</groupId>
            <artifactId>ikanalyzer</artifactId>
            <version>2012_u6</version>
        </dependency>
```

（2）编写代码

```
package cn.itcast.demo;
import org.wltea.analyzer.core.IKSegmenter;
import org.wltea.analyzer.core.Lexeme;
import java.io.IOException;
import java.io.StringReader;
public class Test {
    public static void main(String[] args) throws IOException {
        String text="基于java语言开发的轻量级的中文分词工具包";
        StringReader sr=new StringReader(text);
        IKSegmenter ik=new IKSegmenter(sr, true);
        Lexeme lex=null;
        while((lex=ik.next())!=null){
            System.out.print(lex.getLexemeText()+" ");
        }
    }
}
```

IKSegmenter 的第一个构造参数为StringReader类型 。 StringReader是装饰Reader的类，其用法是读取一个String字符串

IKSegmenter 的第二个构造参数userSmart 为切分粒度 true表示最大切分 false表示最细切分

Lexeme： 词单位类

3.3 构建分词语料库

需求：按照十次方划分的频道，分别在CSDN上抓取各类文章，并以分词形式保存，词之间用空格分隔。

步骤：

（1）修改tensquare_common模块的pom.xml，引入依赖

```
        <dependency>
            <groupId>com.janeluo</groupId>
            <artifactId>ikanalyzer</artifactId>
            <version>2012_u6</version>
        </dependency>
```

（2）修改tensquare_common模块，创建分词工具类

```
package util;
import org.wltea.analyzer.core.IKSegmenter;
import org.wltea.analyzer.core.Lexeme;
import java.io.IOException;
import java.io.StringReader;
/**
 * 分词工具类
 */
public class IKUtil {

    /**
     * 根据文本返回分词后的文本
     * @param content
     * @return
     */
    public static String split(String content,String splitChar) throws
IOException {
        StringReader sr=new StringReader(content);
        IKSegmenter ik=new IKSegmenter(sr, true);
        Lexeme lex=null;
        StringBuilder sb=new StringBuilder("");
        while((lex=ik.next())!=null){
            sb.append(lex.getLexemeText()+splitChar);//拼接
        }
        return sb.toString();
    }
}
```

（3）修改tensquare_common模块，创建HTML工具类(资源中已提供)，用于将文本中的html标签剔除

```
package util;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/**
 * html标签处理工具类
 */
public class HTMLUtil {

    public static String delHTMLTag(String htmlStr){
        String regEx_script="<script[^>]*?>[\\s\\S]*?<\\/script>"; //定义
script的正则表达式
        String regEx_style="<style[^>]*?>[\\s\\S]*?<\\/style>"; //定义
style的正则表达式
        String regEx_html="<[^>]+>"; //定义HTML标签的正则表达式

        Pattern
p_script=Pattern.compile(regEx_script,Pattern.CASE_INSENSITIVE);
        Matcher m_script=p_script.matcher(htmlStr);
        htmlStr=m_script.replaceAll(""); //过滤script标签

        Pattern
p_style=Pattern.compile(regEx_style,Pattern.CASE_INSENSITIVE);
        Matcher m_style=p_style.matcher(htmlStr);
        htmlStr=m_style.replaceAll(""); //过滤style标签

        Pattern
p_html=Pattern.compile(regEx_html,Pattern.CASE_INSENSITIVE);
        Matcher m_html=p_html.matcher(htmlStr);
        htmlStr=m_html.replaceAll(""); //过滤html标签

        return htmlStr.trim(); //返回文本字符串
    }
}
```

（4）修改tensquare_crawler模块，创建ArticleTxtPipeline，用于负责将爬取的内容存入文本文件

```
package com.tensquare.crawler.pipeline;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;
import us.codecraft.webmagic.ResultItems;
import us.codecraft.webmagic.Task;
import us.codecraft.webmagic.pipeline.Pipeline;
import util.HTMLUtil;
import util.IKUtil;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Map;
import java.util.UUID;

/**
 * 入库类(分词后保存为txt)
 */
@Component
public class ArticleTxtPipeline implements Pipeline {

    @Value("${ai.dataPath}")
    private String dataPath;

    private String channelId;//频道ID

    public void setChannelId(String channelId) {
        this.channelId = channelId;
    }

    @Override
    public void process(ResultItems resultItems, Task task) {

        String title = resultItems.get("title");//获取标题
        String content=  HTMLUtil.delHTMLTag(resultItems.get("content"))
;//获取正文并删除html标签
        try {

            //将标题+正文分词后保存到相应的文件夹
```

```
            PrintWriter printWriter=new PrintWriter(new
File(dataPath+"/"+channelId+ "/"+UUID.randomUUID()+".txt"));
            printWriter.print(IKUtil.split(title+" "+content," "));
            printWriter.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

（5）在配置文件application.yml中添加配置

```
ai:
  #语料库目录
  dataPath : E:\\article
```

（6）修改ArticleTask类，新增爬取db 和web的任务

```
package com.tensquare.crawler.task;
import com.tensquare.crawler.pipeline.ArticleTxtPipeline;
import com.tensquare.crawler.processor.ArticleProcessor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import us.codecraft.webmagic.Spider;
import us.codecraft.webmagic.scheduler.RedisScheduler;
/**
 * 文章任务类
 */
@Component
public class ArticleTask {

    //@Autowired
    //private ArticlePipeline articlePipeline;

    @Autowired
    private ArticleTxtPipeline articleTxtPipeline;

    @Autowired
    private RedisScheduler redisScheduler;

    /**
     * 爬取ai数据
     */
    @Scheduled(cron="0 38 10 * * ?")
    public void aiTask(){
        System.out.println("爬取AI文章");
        Spider spider = Spider.create(new ArticleProcessor());
        spider.addUrl("https://blog.csdn.net/nav/ai");
        articleTxtPipeline.setChannelId("ai");
        spider.addPipeline(articleTxtPipeline);
        spider.setScheduler(redisScheduler);
        spider.start();
        spider.stop();
    }



    /**
```

```
     * 爬取db数据
     */
    @Scheduled(cron="20 17 11 * * ?")
    public void dbTask(){
        System.out.println("爬取DB文章");
        Spider spider = Spider.create(new ArticleProcessor());
        spider.addUrl("https://blog.csdn.net/nav/db");
        articleTxtPipeline.setChannelId("db");
        spider.addPipeline(articleTxtPipeline);
        spider.setScheduler(redisScheduler);
        spider.start();
        spider.stop();
    }


    /**
     * 爬取web数据
     */
    @Scheduled(cron="20 27 11 * * ?")
    public void webTask(){
        System.out.println("爬取WEB文章");
        Spider spider = Spider.create(new ArticleProcessor());
        spider.addUrl("https://blog.csdn.net/nav/web");
        articleTxtPipeline.setChannelId("web");
        spider.addPipeline(articleTxtPipeline);
        spider.setScheduler(redisScheduler);
        spider.start();
        spider.stop();
    }

}
```

4 Deeplearning4j之Word2Vec

4.1 Deeplearning4j简介

DeepLearning4J（DL4J）是一套基于Java语言的神经网络工具包，可以构建、定型和部署神经网络。DL4J与Hadoop和Spark集成，支持分布式CPU和GPU，为商业环境（而非研究工具目的）所设计。Skymind是DL4J的商业支持机构。

Deeplearning4j拥有先进的技术，以即插即用为目标，通过更多预设的使用，避免多余的配置，让非企业也能够进行快速的原型制作。DL4J同时可以规模化定制。

https://deeplearning4j.org/cn/index.html

4.2 Word2Vec简介

Word2vec是一种比较流行的自然语言算法，能创建可以输入深度神经网络的神经词向量。

4.3 代码实现

4.3.1 分词语料库合并

（1）创建模块tensquare_ai ,创建启动类

```
package com.tensquare.ai;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;
@SpringBootApplication
@EnableScheduling
public class AiApplication {

    public static void main(String[] args) {
        SpringApplication.run(AiApplication.class, args);
    }
}
```

（2）在tensquare_common创建文件工具类 FileUtil (资源中提供)，用于文本文件的读取，合并，以及获取某目录下的文本文件名称

```
package com.tensquare.ai.util;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

/**
 * 文件工具类
 */
public class FileUtil {


    /**
     * 将多个文本文件合并为一个文本文件
     * @param outFileName
     * @param inFileNames
     * @throws IOException
     */
    public static void merge(String outFileName ,List<String>
inFileNames) throws IOException {

        FileWriter writer =  new FileWriter(outFileName, false);
        for(String inFileName :inFileNames ){
            try {
                String txt= readToString(inFileName);
                writer.write(txt);
                System.out.println(txt);
            }catch (Exception e){
            }
        }
        writer.close();
    }


    /**
     * 查找某目录下的所有文件名称
     * @param path
     * @return
     */
    public static List<String> getFiles(String path) {

        List<String> files = new ArrayList<String>();
```

```
        File file = new File(path);
        File[] tempList = file.listFiles();

        for (int i = 0; i < tempList.length; i++) {
            if (tempList[i].isFile()) {//如果是文件
                files.add(tempList[i].toString());
            }
            if (tempList[i].isDirectory()) {//如果是文件夹
                files.addAll( getFiles(tempList[i].toString())  );
            }
        }
        return files;
    }


    /**
     * 读取文本文件内容到字符串
     * @param fileName
     * @return
     */
    public static String readToString(String fileName) throws IOException
{
        String encoding = "UTF-8";
        File file = new File(fileName);
        Long filelength = file.length();
        byte[] filecontent = new byte[filelength.intValue()];
        FileInputStream in = new FileInputStream(file);
        in.read(filecontent);
        in.close();
        return new String(filecontent, encoding);
    }

}
```

（3）创建application.yml

```
ai:
  #语料库汇总文本文件
  wordLib : E:\\article.txt
  #语料库目录
  dataPath : E:\\article
```

（4）创建服务类

```
@Service
public class Word2VecService {
    //模型分词路径
    @Value("${ai.wordLib}")
    private String wordLib;


    //爬虫分词后的数据路径
    @Value("${ai.dataPath}")
    private String dataPath;


    /**
     * 合并
     */
    public void mergeWord(){
        List<String> fileNames = FileUtil.getFiles(dataPath);
        try {
            FileUtil.merge(wordLib,fileNames);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

（5）创建任务类TrainTask

```
/**
 * 训练任务类
 */
@Component
public class TrainTask {

    @Autowired
    private Word2VecService word2VecService;

    /**
     * 训练模型
     */
    @Scheduled(cron="0 30 16 * * ?")
    public void trainModel(){
        System.out.println("开始合并语料库......");
        word2VecService.mergeWord();
        System.out.println("合并语料库结束------");
    }

}
```

4.3.2 构建词向量模型

（1）pom.xml引入依赖

```
    <properties>
        <dl4j.version>1.0.0-beta</dl4j.version>
        <nd4j.version>1.0.0-beta</nd4j.version>
        <nd4j.backend>nd4j-native-platform</nd4j.backend>
     </properties>
     <dependencies>
        <dependency>
            <groupId>org.deeplearning4j</groupId>
            <artifactId>deeplearning4j-core</artifactId>
            <version>${dl4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.deeplearning4j</groupId>
            <artifactId>deeplearning4j-nlp</artifactId>
            <version>${dl4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.nd4j</groupId>
            <artifactId>${nd4j.backend}</artifactId>
            <version>${nd4j.version}</version>
        </dependency>
     </dependencies>
```

（2）在application.yml中增加配置

```
ai:
  #词向量模型
  vecModel: E:\\article.vecmodel
```

（3）修改服务类Word2VecService，增加方法用于构建词向量模型

```
package com.tensquare.ai.service;
import com.tensquare.ai.util.FileUtil;
import org.deeplearning4j.models.embeddings.loader.WordVectorSerializer;
import org.deeplearning4j.models.word2vec.Word2Vec;
import org.deeplearning4j.text.sentenceiterator.LineSentenceIterator;
import org.deeplearning4j.text.sentenceiterator.SentenceIterator;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.io.File;
import java.io.IOException;
import java.util.List;
/**
 * 把分词文本内容训练成机器符号
 * Created by lgh on 2018/3/28.
 */
@Service
public class Word2VecService {

    //模型分词路径
    @Value("${ai.wordLib}")
    private String wordLib;

    //模型训练保存的路径
    @Value("${ai.vecModel}")
    private String vecModel;

    //爬虫分词后的数据路径
    @Value("${ai.dataPath}")
    private String dataPath;


    /**
     * 合并
     */
    public void mergeWord(){
        //..........
    }

    /**

     * 根据分词模型生成 词向量模型
```

```
     */
    public void build() {
        boolean flag = false;
        try {
            //加载数爬虫分词数据集
            SentenceIterator iter = new LineSentenceIterator(new
File(wordLib));
            Word2Vec vec = new Word2Vec.Builder()
                    .minWordFrequency(5)
                    .iterations(1)
                    .layerSize(100)
                    .seed(42)
                    .windowSize(5)
                    .iterate(iter)
                    .build();
            vec.fit();
            //保存模型之前先删除
            new File(vecModel).delete();//删除
            WordVectorSerializer.writeWordVectors(vec, vecModel);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return flag;
    }
}
```

参数说明：

- *minWordFrequency*是一个词在语料中必须出现的最少次数。本例中出现不到5次的词都不予学习。词语必须在多种上下文中出现，才能让模型学习到有用的特征。对于规模很大的语料库，理应提高出现次数的下限。
- *iterations*是网络在处理一批数据时允许更新系数的次数。迭代次数太少，网络可能来不及学习所有能学到的信息；迭代次数太多则会导致网络定型时间变长。
- *layerSize*指定词向量中的特征数量，与特征空间的维度数量相等。以500个特征值表示的词会成为一个500维空间中的点。
- windowSize：表示当前词与预测词在一个句子中的最大距离是多少
- seed：用于随机数发生器。与初始化词向量有关
- *iterate*: 告知网络当前定型的是哪一批数据集。
- *vec.fit()* 让已配置好的网络开始定型。

（4）修改任务类TrainTask

```
package com.tensquare.ai.task;
import com.tensquare.ai.service.CnnService;
import com.tensquare.ai.service.Word2VecService;
import com.tensquare.ai.util.FileUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import java.io.IOException;
import java.util.List;

/**
 * 模型任务类
 */
@Component
public class TrainTask {

    @Autowired
    private Word2VecService word2VecService;

    /**
     * 训练模型
     */
    @Scheduled(cron="0 30 16 * * ?")
    public void makeModel(){

        System.out.println("开始合并语料库......");
        word2VecService.mergeWord();
        System.out.println("合并语料库结束------");

        System.out.println("开始构建词向量模型");
        word2VecService.build();
        System.out.println("构建词向量模型结束");
    }

}
```

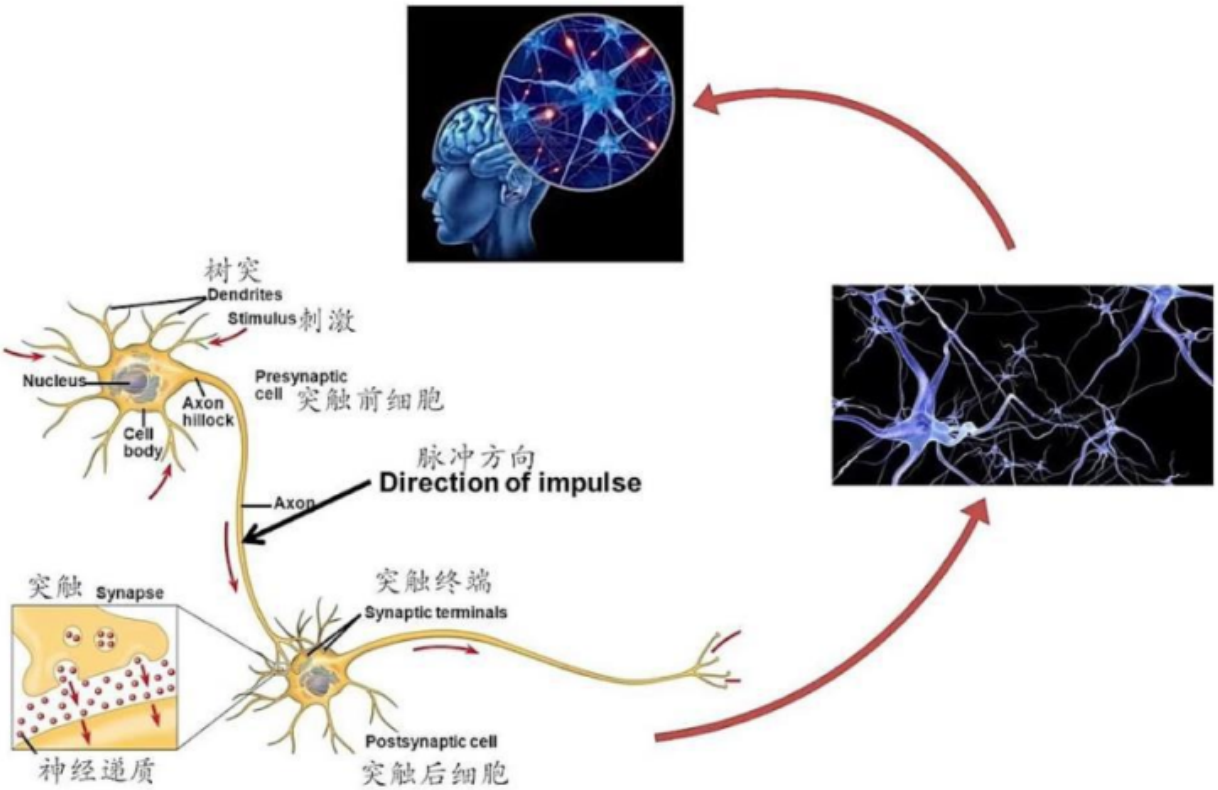
5 构建卷积神经网络模型（CNN）

5.1 卷积神经网络模型

5.1.1 神经网络

神经网络可以指向两种，一个是生物神经网络，一个是人工神经网络。

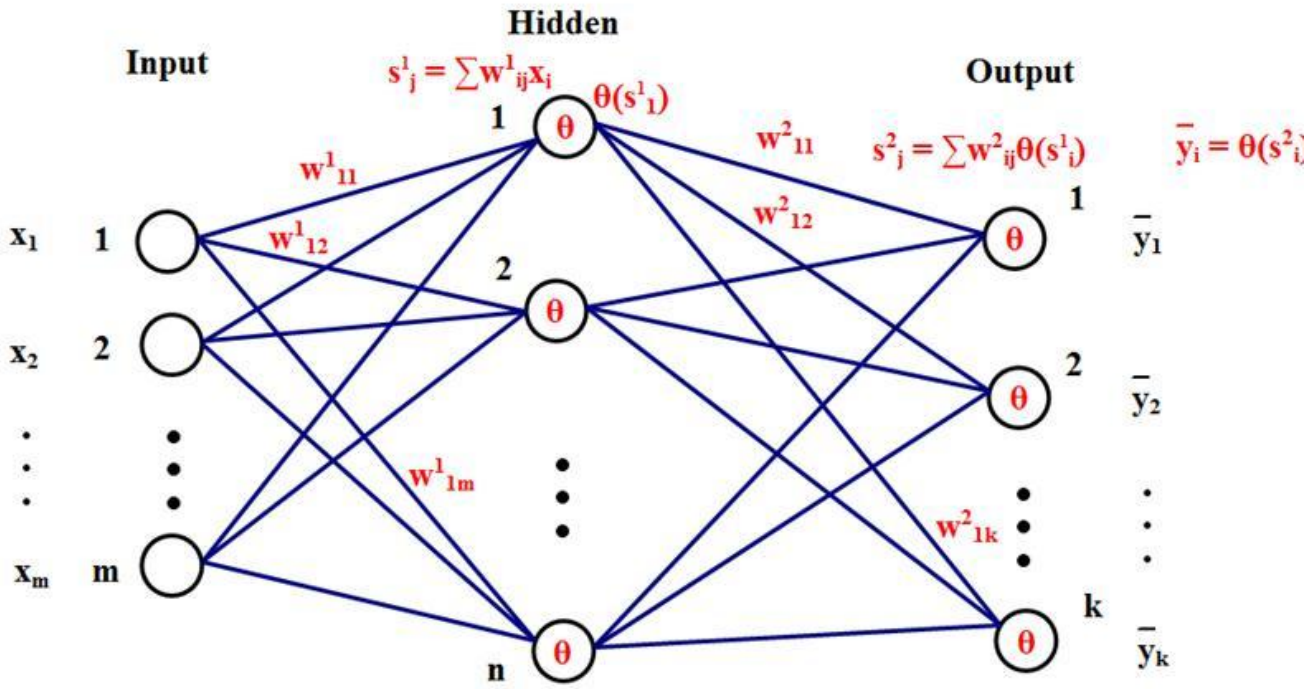
生物神经网络：一般指生物的大脑神经元，细胞，触点等组成的网络，用于产生生物的意识，帮助生物进行思考和行动。



一个神经元可以通过轴突作用于成千上万的神经元，也可以通过树突从成千上万的神经元接受信息。上级神经元的轴突在有电信号传导时释放出化学递质，作用于下一级神经元的树突，树突受到递质作用后产生出电信号，从而实现了神经元间的信息传递。化学递质可以使下一级神经元兴奋或抑制。

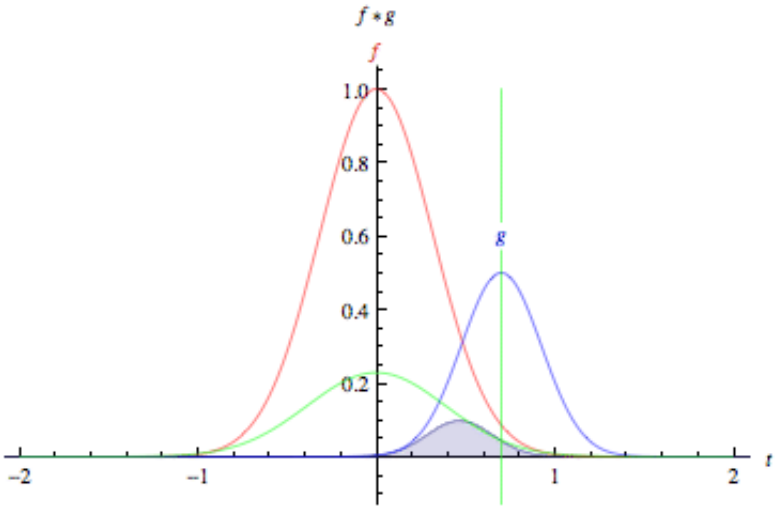
人工神经网络：人工神经网络（Artificial Neural Networks，简写为ANNs）也简称为神经网络（NNs）或称作连接模型（Connection Model），它是一种模仿动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。

是一种应用类似于大脑神经突触联接的结构进行信息处理的数学模型。在工程与学术界也常直接简称为“神经网络”或类神经网络。



5.1.2 卷积

英文中的 **to convolve** 词源为拉丁文 **convolvere**，意为“卷在一起”。从数学角度说，卷积是指用来计算一个函数通过另一个函数时，两个函数有多少重叠的积分。卷积可以视为通过相乘的方式将两个函数进行混合。



https://blog.csdn.net/vactivx/article/details/61427203

https://www.zhihu.com/question/22298352?rf=21686447

https://www.zhihu.com/question/54677157/answer/141245297

5.2 代码实现

（1）修改配置文件application.yml

```
ai:
  #神经网络卷积模型
  cnnModel: E:\\article.cnnmodel
```

（2）添加工具类CnnUtil（资源中提供）

```
package com.tensquare.ai.util;
import org.deeplearning4j.iterator.CnnSentenceDataSetIterator;
import org.deeplearning4j.iterator.LabeledSentenceProvider;
import org.deeplearning4j.iterator.provider.FileLabeledSentenceProvider;
import org.deeplearning4j.models.embeddings.loader.WordVectorSerializer;
import org.deeplearning4j.models.embeddings.wordvectors.WordVectors;
import org.deeplearning4j.nn.conf.ComputationGraphConfiguration;
import org.deeplearning4j.nn.conf.ConvolutionMode;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.graph.MergeVertex;
import org.deeplearning4j.nn.conf.layers.ConvolutionLayer;
import org.deeplearning4j.nn.conf.layers.GlobalPoolingLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.graph.ComputationGraph;
import org.deeplearning4j.util.ModelSerializer;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import java.io.File;
import java.io.IOException;
import java.util.*;

/**
 * CNN工具类
 */
public class CnnUtil {


    public static ComputationGraph  createComputationGraph(int
cnnLayerFeatureMaps){
        //训练模型
        int vectorSize = 300;                   //向量大小
        //int cnnLayerFeatureMaps = 100;        ////每种大小卷积层的卷积核的数
量=词向量维度
        ComputationGraphConfiguration config = new
NeuralNetConfiguration.Builder()
                .convolutionMode(ConvolutionMode.Same)// 设置卷积模式
                .graphBuilder()
                .addInputs("input")
                .addLayer("cnn1", new ConvolutionLayer.Builder()//卷积层

                        .kernelSize(3,vectorSize)//卷积区域尺寸
```

```
                        .stride(1,vectorSize)//卷积平移步幅
                        .nIn(1)
                        .nOut(cnnLayerFeatureMaps)
                        .build(), "input")
                .addLayer("cnn2", new ConvolutionLayer.Builder()
                        .kernelSize(4,vectorSize)
                        .stride(1,vectorSize)
                        .nIn(1)
                        .nOut(cnnLayerFeatureMaps)
                        .build(), "input")
                .addLayer("cnn3", new ConvolutionLayer.Builder()
                        .kernelSize(5,vectorSize)
                        .stride(1,vectorSize)
                        .nIn(1)
                        .nOut(cnnLayerFeatureMaps)
                        .build(), "input")
                .addVertex("merge", new MergeVertex(), "cnn1", "cnn2",
"cnn3")//连接
                .addLayer("globalPool", new
GlobalPoolingLayer.Builder()//池化层
                        .build(), "merge")
                .addLayer("out", new OutputLayer.Builder()//输出层
                        .nIn(3*cnnLayerFeatureMaps)
                        .nOut(3)
                        .build(), "globalPool")
                .setOutputs("out")
                .build();
        ComputationGraph net = new ComputationGraph(config);
        net.init();
        return net;
    }

    /**
     *  返回训练数据集
     * @param path 数据集所在目录
     * @param childPaths 子目录
     * @param wordVectors 词向量模型
     * @param minibatchSize 最小批处理数量
     * @param maxSentenceLength 最大句子长度
     * @param rng 随机种子
     * @return
```

```
     */
    public static DataSetIterator getDataSetIterator(String path,
String[] childPaths, WordVectors wordVectors, int minibatchSize,
                                                      int maxSentenceLength,
Random rng ){
        //词标记分类比标签
        Map<String,List<File>> reviewFilesMap = new HashMap<>();

        for( String childPath: childPaths){
            reviewFilesMap.put(childPath, Arrays.asList(new
File(path+"/"+ childPath ).listFiles()));
        }
        //标记跟踪
        LabeledSentenceProvider sentenceProvider = new
FileLabeledSentenceProvider(reviewFilesMap, rng);
        return new CnnSentenceDataSetIterator.Builder()
                .sentenceProvider(sentenceProvider)
                .wordVectors(wordVectors)
                .minibatchSize(minibatchSize)
                .maxSentenceLength(maxSentenceLength)
                .useNormalizedWordVectors(false)
                .build();
    }

    public static Map<String, Double> predictions(String vecModel,String
cnnModel,String dataPath,String[] childPaths,String content) throws
IOException {
        Map<String, Double> map = new HashMap<>();
        //模型应用
        ComputationGraph model =
ModelSerializer.restoreComputationGraph(cnnModel);//通过cnn模型获取计算图对
象
        WordVectors wordVectors =
WordVectorSerializer.loadStaticModel(new File(vecModel));//加载词向量模型对
象
        //加载数据集
        DataSetIterator dataSet =
CnnUtil.getDataSetIterator(dataPath,childPaths, wordVectors, 32, 256, new
Random(12345));
        //通过句子获取概率矩阵对象
        INDArray featuresFirstNegative = ((CnnSentenceDataSetIterator)
```

```
dataSet).loadSingleSentence(content);
        INDArray predictionsFirstNegative
=model.outputSingle(featuresFirstNegative);
        List<String> labels = dataSet.getLabels();

        for (int i = 0; i < labels.size(); i++) {
            map.put(labels.get(i) + "",
predictionsFirstNegative.getDouble(i));
        }
        return map;
    }
}
```

（3）创建服务类CnnService

```
package com.tensquare.ai.service;
import com.tensquare.ai.util.CnnUtil;
import org.deeplearning4j.models.embeddings.loader.WordVectorSerializer;
import org.deeplearning4j.models.embeddings.wordvectors.WordVectors;
import org.deeplearning4j.nn.graph.ComputationGraph;
import org.deeplearning4j.util.ModelSerializer;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.io.File;
import java.io.IOException;
import java.util.*;
/**
 * 人工智能分类模型
 */
@Service
public class CnnService {

    //词向量模型
    @Value("${ai.vecModel}")
    private String vecModel;

    //训练模型结果保存路径
    @Value("${ai.cnnModel}")
    private String cnnModel;

    //爬虫分词后的数据路径
    @Value("${ai.dataPath}")
    private String dataPath;

    /**
     * 构建卷积模型
     * @return
     */
    public boolean build(){
        try{
            //创建计算图对象
            ComputationGraph net = CnnUtil.createComputationGraph(100);
            //加载词向量 训练数据集

            WordVectors wordVectors =
```

```
WordVectorSerializer.loadStaticModel(new File(vecModel));
            String[] childPaths={"ai","db","web"};
            DataSetIterator trainIter =
CnnUtil.getDataSetIterator(dataPath,childPaths,  wordVectors, 32, 256,
new Random(12345));
            //模型训练
            net.fit(trainIter);
            //保存模型之前先删除
            new File(cnnModel).delete();
            //保存模型
            ModelSerializer.writeModel(net,cnnModel,true);
            return  true;
        }catch (Exception e){
            e.printStackTrace();
        }
        return false;
    }

}
```

（4）修改任务类 TrainTask

```
package com.tensquare.ai.task;
import com.tensquare.ai.service.CnnService;
import com.tensquare.ai.service.Word2VecService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
/**
 * 训练任务类
 */
@Component
public class TrainTask {

    @Autowired
    private Word2VecService word2VecService;

    @Autowired
    private CnnService cnnService;

    /**
     * 训练模型
     */
    @Scheduled(cron="0 30 16 * * ?")
    public void makeModel(){

        System.out.println("开始合并语料库......");
        word2VecService.mergeWord();
        System.out.println("合并语料库结束------");

        System.out.println("开始构建词向量模型");
        word2VecService.build();
        System.out.println("构建词向量模型结束");

        System.out.println("开始构建神经网络卷积模型");
        cnnService.build();
        System.out.println("构建神经网络卷积模型结束");
    }

}
```

6 实现智能分类

6.1 需求分析

传入文本，得到所属分类信息

6.2 代码实现

（1）修改CnnService，增加方法

```
    /**
     * 返回map集合 分类与百分比
     * @param content
     * @return
     */
    public Map textClassify(String content) {
        System.out.println("content:"+content);
        //分词
        try {
            content=util.IKUtil.split(content," ");
        } catch (IOException e) {
            e.printStackTrace();
        }
        String[] childPaths={"ai","db","web"};
        //获取预言结果
        Map map = null;
        try {
            map = CnnUtil.predictions(vecModel, cnnModel, dataPath,
childPaths, content);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return map;
    }
```

（2）创建AiController

```
package com.tensquare.ai.controller;
import com.tensquare.ai.service.CnnService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import java.util.Map;
@RestController
@RequestMapping("/ai")
public class AiController {

    @Autowired
    private CnnService cnnService;

    @RequestMapping(value="/textclassify",method = RequestMethod.POST)
    public Map textClassify( @RequestBody Map<String,String> content){
        return cnnService.textClassify(content.get("content"));
    }
}
```

（3）使用postman测试 http://localhost:8080/ai/textclassify 提交格式：

```
{
        "content":"测试文本"
}
```