作业九报告

目录

[技术方法 1](#_Toc11263822)

[导入package 2](#_Toc11263823)

[转换格式 2](#_Toc11263824)

[Logistic回归 3](#_Toc11263825)

[多层感知机 6](#_Toc11263826)

[使用已训练的模型 7](#_Toc11263827)

[试验结果 7](#_Toc11263828)

[Logistic回归 7](#_Toc11263829)

[多层感知机 7](#_Toc11263830)

[心得 8](#_Toc11263831)

[代码 8](#_Toc11263832)

[gen\_libsvm\_file 8](#_Toc11263833)

[Logistic Regression 9](#_Toc11263834)

[Multi-layer Perceptron 10](#_Toc11263835)

[use saved model 11](#_Toc11263836)

组队作业。在以下机器学习任务中**任选其一**。

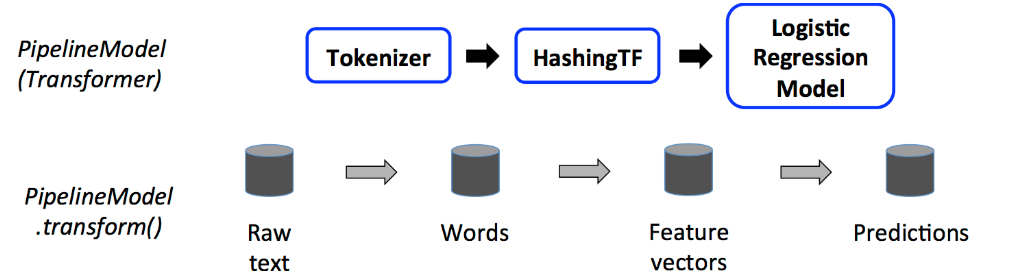
一、训练模型分开希格斯玻色子信号和背景。教学网上给出的数据 HIGGS\_SMALL.7z 是原数据的前一百万行。第一列是标签，1表示信号(signal)，0表示背景(background)，剩下的28个列是28种特征，其中，后7种特征是物理学家根据前21种特征提取出来的，前21种特征是探测器的测量值，所以也许你只愿使用前21种特征。自己划分训练集和测试集，如果需要多次调参的话，应该再从训练集中分出验证集作为指导。

# 技术方法

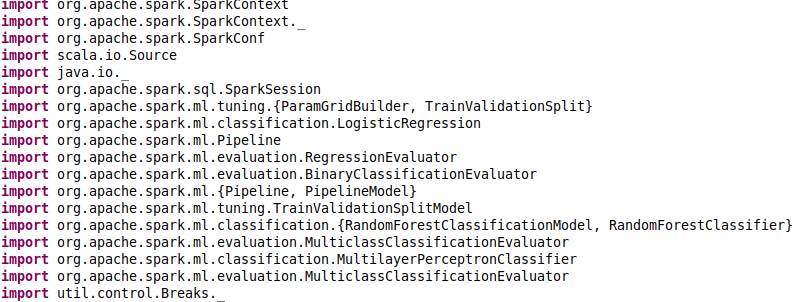
数据每行29列，第一列为0-1标签，后28列为特征，

使用scala语言

pipeline将transformer和estimator组合到一起，训练相当于求解pipeline中的参数，解出其中参数后pipeline即为一个transformer，输入数据得出分类结果



## 导入package



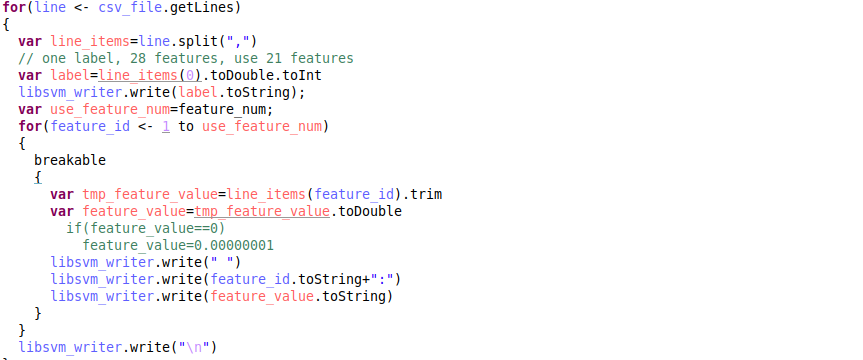
## 转换格式

libsvm格式：每行为一条数据，一个标签和多个特征，每个特征为特征id:特征值的形式。



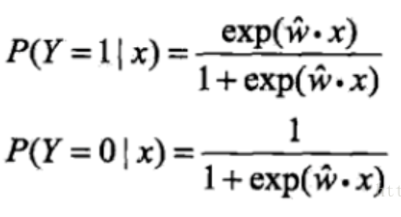
libsvm格式便于spark读取，可直接转换为dataframe，有label和features两个域，作为transformer,estimator或pipeline的输入。

将csv根据指定的特征值个数转换为libsvm格式,指定28个特征值



## Logistic回归

Logistic回归相当于一层神经网络，一个线性函数加上一个非线性函数，是对数似然函数的最优化。

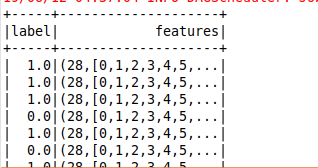


新建sparksession



读取libsvm格式的数据，可以指定特征值个数，得到域为label和features的dataframe





将数据随机分成训练集和测试集，训练集占90%，指定随机种子



新建estimator，logistic回归，最大迭代次数为10



新建pipeline，将estimator放入pipeline，因为此处不需要transformer转换数据格式，所以也可以不用pipeline

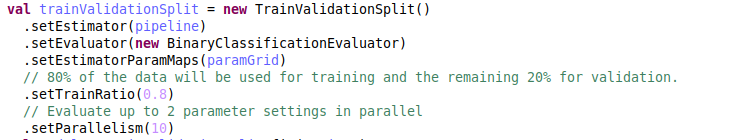


新建参数网格，指定logistic回归的超参数正则项系数的一些可能值，训练时由机器选择最佳的超参数。



新建k折交叉验证对象，将多次将训练集分成训练集和测试集以调整超参数。

指定训练的pipeline，指定evaluator来评估不同的超参数的结果好坏，指定可能超参数的集合即paramGrid，从中选择最佳超参数，指定“训练集”占比

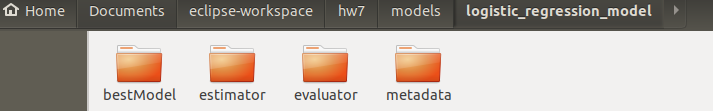


用该trainvalidationsplit对象训练出模型的参数以及选取最佳超参数，得到模型，模型即为一个transformer。



保存模型，下一次直接使用。

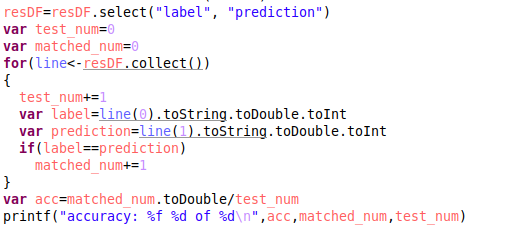


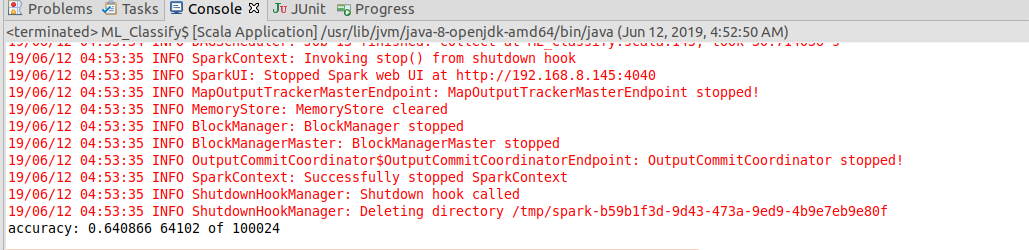


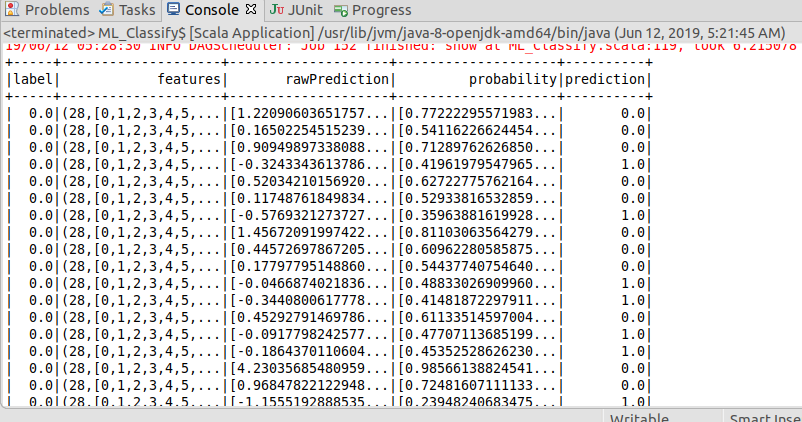
模型是一个transformer，放入测试集得到结果，结果为dataframe，新增prediction域



将结果的label和prediction域比较，得到分类的accuracy







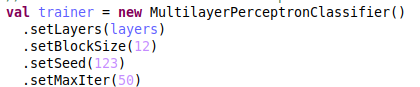
## 多层感知机

多层感知机相当于多个卷积层，相比logistic的一层全连接层相比层次更深，但不是全连接。

指定每层的卷积核的尺寸，第一层的维度应与数据的特征值个数相同



新建multi-layer-perceptron的estimator，指定层数和每层卷积核尺寸，指定blocksize，指定最大迭代次数



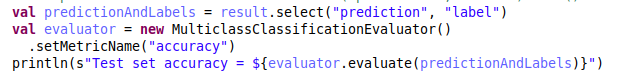
训练模型，保存模型

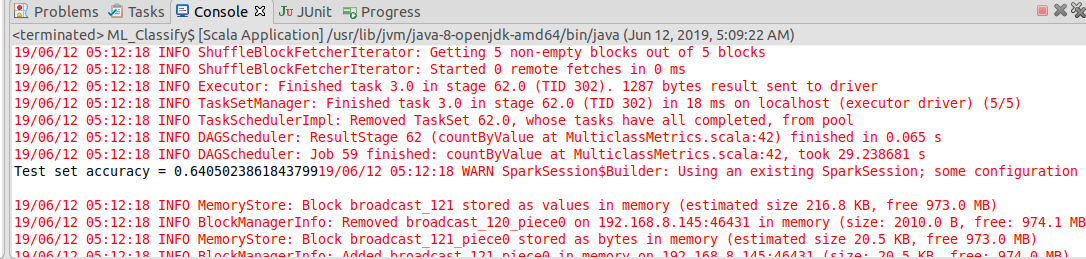


得到结果



新建多分类evaluator评价结果，得到accuracy





## 使用已训练的模型

已训练的模型本质上是transformer，但需要用训练模型时的对象对应的库来提取出模型的参数生成模型，可能是pipeline,estimator,tranvalidationsplit等

加载模型



用该模型作用于测试集得到结果



# 试验结果

## Logistic回归

训练集和验证集9比1，训练集再以8：2多次划分成训练集和验证集进行交叉验证，正则项系数从1,0.5,0.3,0.1, 0.05,0.01,0.001中选取最佳超参数次，10次迭代，



## 多层感知机

训练集和验证机9：1，4层，卷积核尺寸分别为28，5，4，3，blocksize为12,迭代50次



# 心得

多层感知机相当于多层神经网络，第一层的维度应于数据特征值的个数一致

读取数据时可通过option("numFeatures",feature\_num)指定feature个数以避免警告。

虽然多层感知机相比logistic回归的层次更深，更多的考虑到非线性，但不是全连接层，每层是一个卷积，所以没有更好的效果

该数据特征较多，可能需要更深的模型，更多的迭代次数和全连接层

# 代码

## gen\_libsvm\_file

**def** gen\_libsvm\_file(csv\_path:String,libsvm\_path:String,feature\_num:Int): Unit=

{

**val** csv\_file=Source.fromFile(csv\_path)

**val** libsvm\_writer = **new** PrintWriter(**new** File(libsvm\_path))

**for**(line <- csv\_file.getLines)

{

**var** line\_items=line.split(",")

// one label, 28 features, use 21 features

**var** label=line\_items(0).toDouble.toInt

libsvm\_writer.write(label.toString);

**var** use\_feature\_num=feature\_num;

**for**(feature\_id <- 1 to use\_feature\_num)

{

breakable

{

**var** tmp\_feature\_value=line\_items(feature\_id).trim

**var** feature\_value=tmp\_feature\_value.toDouble

// if(feature\_value==0)

// feature\_value=0.00000001

libsvm\_writer.write(" ")

libsvm\_writer.write(feature\_id.toString+":")

libsvm\_writer.write(feature\_value.toString)

}

}

libsvm\_writer.write("\n")

}

csv\_file.close

libsvm\_writer.close

}

## Logistic Regression

**def** lr\_classify(libsvm\_path:String,model\_path:String,feature\_num:Int):Unit=

{

**val** spark = SparkSession.builder().appName("ml").master("local").getOrCreate()

**val** dataDF = spark.read.format("libsvm").option("numFeatures",feature\_num).load(libsvm\_path)

// dataDF.show()

// return

**val** Array(trainDF, testDF) = dataDF.randomSplit(Array(0.9, 0.1), seed = 123)

**val** lr = **new** LogisticRegression().setMaxIter(10)

**val** pipeline = **new** Pipeline().setStages(Array(lr))

**val** paramGrid = **new** ParamGridBuilder()

.addGrid(lr.regParam, Array(1,0.5,0.3,0.1, 0.05,0.01,0.001))

.build()

**val** trainValidationSplit = **new** TrainValidationSplit()

.setEstimator(pipeline)

.setEvaluator(**new** BinaryClassificationEvaluator)

.setEstimatorParamMaps(paramGrid)

// 80% of the data will be used for training and the remaining 20% for validation.

.setTrainRatio(0.8)

// Evaluate up to 2 parameter settings in parallel

.setParallelism(10)

**val** model = trainValidationSplit.fit(trainDF)

model.write.overwrite().save(model\_path)

**var** resDF=model.transform(testDF)

resDF=resDF.select("label", "prediction")

**var** test\_num=0

**var** matched\_num=0

**for**(line<-resDF.collect())

{

test\_num+=1

**var** label=line(0).toString.toDouble.toInt

**var** prediction=line(1).toString.toDouble.toInt

**if**(label==prediction)

matched\_num+=1

}

**var** acc=matched\_num.toDouble/test\_num

printf("accuracy: %f %d of %d\n",acc,matched\_num,test\_num)

}

## Multi-layer Perceptron

**def** perceptron\_classify(libsvm\_path:String,model\_path:String,feature\_num:Int):Unit=

{

// var libsvm\_path="sample\_multiclass\_classification\_data.txt"

**val** spark = SparkSession.builder().appName("example").master("local").getOrCreate()

**import** spark.implicits.\_

// Load the data stored in LIBSVM format as a DataFrame.

**val** data = spark.read.format("libsvm").option("numFeatures",feature\_num)

.load(libsvm\_path)

// Split the data into train and test

**val** splits = data.randomSplit(Array(0.6, 0.4), seed = 1234L)

**val** train = splits(0)

**val** test = splits(1)

// train.show()

// test.show()

// break

// specify layers for the neural network:

// input layer of size 4 (features), two intermediate of size 5 and 4

// and output of size 3 (classes)

**val** layers = Array[Int](feature\_num, 5, 4, 3)

// create the trainer and set its parameters

**val** trainer = **new** MultilayerPerceptronClassifier()

.setLayers(layers)

.setBlockSize(12)

.setSeed(123)

.setMaxIter(50)

// train the model

**val** model = trainer.fit(train)

model.write.overwrite().save(model\_path)

// compute accuracy on the test set

**val** result = model.transform(test)

// result.show()

// val predictionAndLabels = result.select("prediction", "label").show()

**val** predictionAndLabels = result.select("prediction", "label")

**val** evaluator = **new** MulticlassClassificationEvaluator()

.setMetricName("accuracy")

println(s"Test set accuracy = ${evaluator.evaluate(predictionAndLabels)}")

}

## use saved model

**def** use\_saved\_model(libsvm\_path:String,model\_path:String,feature\_num:Int):Unit=

{

**val** spark = SparkSession.builder().appName("ml").master("local").getOrCreate()

**val** dataDF = spark.read.format("libsvm").option("numFeatures",feature\_num).load(libsvm\_path)

// dataDF.show()

**val** Array(trainDF, testDF) = dataDF.randomSplit(Array(0.9, 0.1), seed = 123)

**val** my\_model = TrainValidationSplitModel.load(model\_path)

**var** resDF=my\_model.transform(testDF)

resDF=resDF.select("label", "prediction")

**var** test\_num=0

**var** matched\_num=0

**for**(line<-resDF.collect())

{

test\_num+=1

**var** label=line(0).toString.toDouble.toInt

**var** prediction=line(1).toString.toDouble.toInt

**if**(label==prediction)

matched\_num+=1

}

**var** acc=matched\_num.toDouble/test\_num

printf("accuracy: %f %d of %d\n",acc,matched\_num,test\_num)

}