МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский ядерный университет «МИФИ»

**Обнинский институт атомной энергетики –**

филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»

**(ИАТЭ НИЯУ МИФИ)**

Отделение интеллектуальных кибернетических систем

ЛАБОРАТОРНАЯ РАБОТА №4

«Машинное обучение »

по дисциплине

«Большие данные»

Выполнил студент 1 курса

группы ИВТ-М20

Архипов Д.А.

Проверил:

кандидат технических наук

Грицюк С. В.

Обнинск, 2020

# Цель работы

Определить простую задачу машинного обучения и решить ее. Точность на испытательном наборе должна быть не менее 60%.

# Краткая теория

Apache Spark MLlib используется для создания приложения машинного обучения. Приложение выполняет прогнозный анализ на открытом наборе данных. MLlib — это основная библиотека Spark, которая предоставляет множество служебных программ, полезных для задач машинного обучения, таких как:

1. Классификация;
2. Регрессия;
3. Кластеризация;
4. Моделирование сингулярного разложения и анализа по методу главных компонент;
5. Проверки гипотез и статистической выборки.

## Общие сведения о выбранном алгоритме машинного обучения

Random forest (с англ. — «случайный лес») — алгоритм машинного обучения, предложенный Лео Брейманом и Адель Катлер, заключающийся в использовании комитета (ансамбля) решающих деревьев. Алгоритм сочетает в себе две основные идеи: метод бэггинга Бреймана, и метод случайных подпространств, предложенный Тин Кам Хо. Алгоритм применяется для задач классификации, регрессии и кластеризации. Основная идея заключается в использовании большого ансамбля решающих деревьев, каждое из которых само по себе даёт очень невысокое качество классификации, но за счёт их большого количества результат получается хорошим.

Достоинства:

* Способность эффективно обрабатывать данные с большим числом признаков и классов.
* Нечувствительность к масштабированию значений признаков.
* Одинаково хорошо обрабатываются как непрерывные, так и дискретные признаки. Существуют методы построения деревьев по данным с пропущенными значениями признаков.
* Существуют методы оценивания значимости отдельных признаков в модели.
* Внутренняя оценка способности модели к обобщению.
* Высокая параллелизуемость и масштабируемость.

Недостатки:

* Большой размер получающихся моделей. Требуется O (K) памяти для хранения модели, где K — число деревьев.

# Ход работы

В процессе работы мы рассмотрим набор данных, состоящий информации по рабочим местам за 2018 год. Сперва мы подключим контекст Spark, а также укажем в качестве dataframe, описанный выше набор данных. Первые 20 строк набора показаны на рисунке 1.

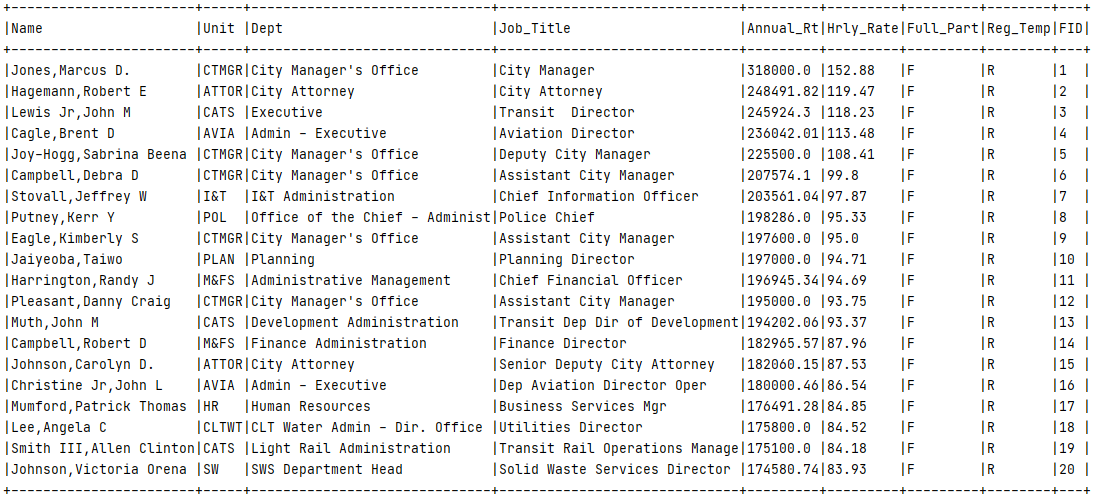


Рисунок 1. Топ 20 строк набора данных

Поставим задачу исследовать взаимосвязь зарплаты (месячной (“Annual\_Rt”) и почасовой(“Hrly\_Rate”)), а также отделения (“Dept”) с областью ("Unit”). В ходе исследования мы хотим предсказать к какой области будет относиться та или иная ЗП и отделение.

В процессе работы у нас появляются трудности с объединением данных, поэтому мы используем VectorAssembler – это преобразователь, который объединяет заданный список столбцов в один векторный столбец. Это полезно для объединения необработанных функций и функций, созданных различными преобразователями функций, в один вектор функций.

Работать напрямую с данными, хоть и в случае моего небольшого набора данных, не составляет труда, но для удобства обращения воспользуемся StringIndexer, который кодирует строковый столбец меток в столбец индексов меток. Также введем столбец features, который будет агрегировать значения ЗП и отделения. Результаты представлены на рисунке 2.

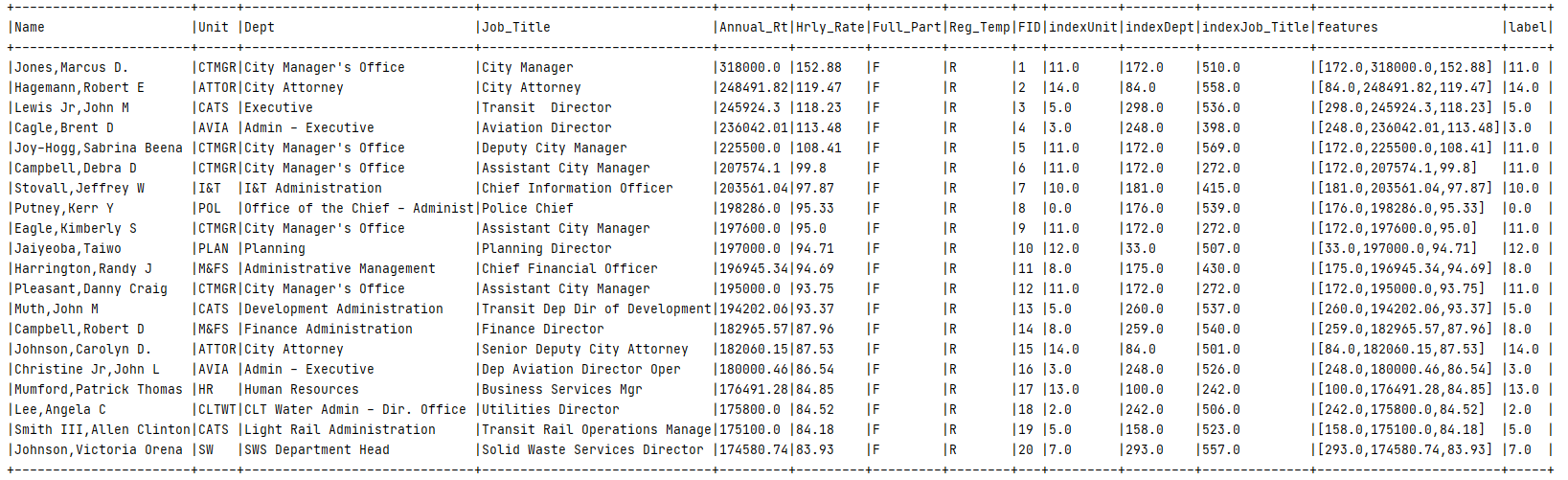


Рисунок 2. Результаты агрегирования данных

Определим данные, которые мы ищем, дополнительно переименовав значения со стандартами Spark MLlib. Ввиду наличия большого объема данных, нам представляется возможным разбить их на более мелкие части. Таким образом, мы подготовим данные для случайного леса.

Определим изначальное количество данных в наборе, а также число в обучающую выборку и в тестовую.

dataframe count 7454

trainingData count 5190

testData count 2264

Зададим необходимые параметры для обучения и поддержания точности на уровне, указанном в цели работы. Полученные результаты отобразим на рисунке 3.

В результате, мы можем увидеть, что теперь наша модель может предсказать по ЗП и отделению к какой области относиться сотрудник. Крайний правый столбец prediction, если исходить из наших предложений, совпадает с действительным значением label (в большинстве случаев), что является верным результатом исследования.

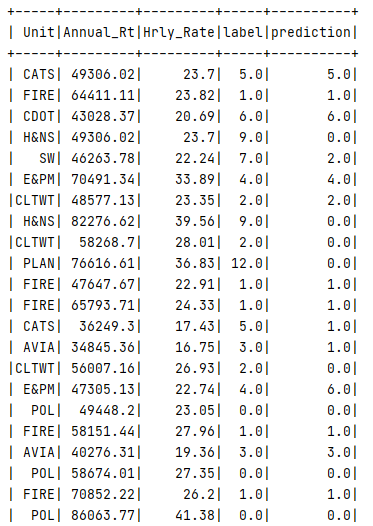


Рисунок 3. Результаты предсказаний

Отдельно посчитаем точность наших предсказаний и округлим значение до двух знаков после запятой: Accuracy = 0.62.

Невысокую точность можно объяснить тем, что большая часть областей похожа друг на друга, их средняя зарплата как бы размазана по всей области определения (рисунок 4), несмотря на то что общее их количество равно двадцати.

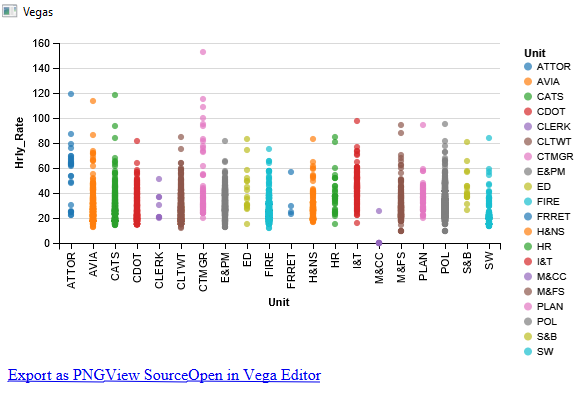


Рисунок 4. Бокс-плот областей

# Вывод

В ходе данной лабораторной работы я изучил особенности и возможности способов работы со Spark ML и предсказания данных на основе случайного дерева классификаций. Для этого я использовал набор данных информации о работах за 2018 год. Полученная модель умеет по заработной плате и отделении определять к какой области относится сотрудник. Данная информация может быть полезна при планировке бюджета.

# Листинг кода

package LabFour  
  
import org.apache.log4j.Level.*WARN*import org.apache.log4j.LogManager  
import org.apache.spark.ml.feature.VectorAssembler  
import org.apache.spark.sql.{DataFrame, SparkSession}  
import org.apache.spark.ml.classification.{RandomForestClassifier}  
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator  
import org.apache.spark.ml.feature.{StringIndexer}  
  
object LabFour {  
 val *PATH*: String = "src/main/data"  
 val *NODES*: Int = 3  
  
  
 def main(args: Array[String]): Unit = {  
  
 *//SparkSession initialization* val spark: SparkSession = SparkSession  
 .*builder*()  
 .appName("Lab4")  
 .master(s"local[**$***NODES*]")  
 .getOrCreate  
 LogManager.*getRootLogger*.setLevel(*WARN*)  
  
 *//Reading .csv file into dataframe* val df: DataFrame = spark.read  
 .format("csv")  
 .option("header", "true")  
 .option("delimiter", ",")  
 .option("inferSchema", true)  
 .load(s"**$***PATH*/var.csv")  
 df.show(false)  
 *println*("dataframe count " + df.count())  
  
 *// Indexing some columns for using as features* val unitIndexer = new StringIndexer()  
 .setInputCol("Unit")  
 .setOutputCol("indexUnit")  
 val unitIndexedDf = unitIndexer.fit(df).transform(df)  
 val deptIndexer = new StringIndexer()  
 .setInputCol("Dept")  
 .setOutputCol("indexDept")  
 val deptIndexedDF = deptIndexer.fit(unitIndexedDf).transform(unitIndexedDf)  
 val jobIndexer = new StringIndexer()  
 .setInputCol("Job\_Title")  
 .setOutputCol("indexJob\_Title")  
 val jobIndexedDF = jobIndexer.fit(deptIndexedDF).transform(deptIndexedDF)  
  
 *//Vectorization of required columns* val cols = *Array*("indexDept", "Annual\_Rt", "Hrly\_Rate")  
 val assembler = new VectorAssembler ()  
 .setInputCols(cols)  
 .setOutputCol("features")  
 val featureDf = assembler.transform(jobIndexedDF)  
  
 *//Renaming columns for suiting SparkML* val indexer = new StringIndexer()  
 .setInputCol("Unit")  
 .setOutputCol("label")  
 val labelDf = indexer.fit(featureDf).transform(featureDf)  
 labelDf.show(false)  
  
 *//Splitting dataframe into training and test dataframes* val seed = 9999  
 val *Array*(trainingData, testData) = labelDf.randomSplit(*Array*(0.7, 0.3), seed)  
  
 *println*("trainingData count " + trainingData.count())  
 *println*("testData count " + testData.count())  
  
 *// Train RandomForestClassifier model with training data set  
 // Setting max feature bins at 330* val Regression = new RandomForestClassifier()  
 .setLabelCol("label")  
 .setFeaturesCol("features")  
 .setMaxBins(330)  
 val model = Regression.fit(trainingData)  
  
 *// run model with test data set to get predictions  
 // this will add new columns rawPrediction, probability and prediction* val predictionDf = model.transform(testData)  
 predictionDf.select("Unit", "Annual\_Rt", "Hrly\_Rate", "label", "prediction").show(50)  
  
 *// Select (prediction, label) and compute accuracy.* val evaluator = new MulticlassClassificationEvaluator()  
 .setLabelCol("label")  
 .setPredictionCol("prediction")  
 .setMetricName("accuracy")  
 val accuracy = evaluator.evaluate(predictionDf)  
 *//round accuracy up to 2 digits  
 println*(s"Accuracy = **$**{*BigDecimal*(accuracy).setScale(2, *BigDecimal*.RoundingMode.*HALF\_UP*).toDouble}")  
  
 }  
}