НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота № 4-5

З дисципліни: «Оброблення надвеликих масивів даних»

**Виконав:**

студент 5 курсу

групи ТР-02мп, ТЕФ

Круглий Д.В.

**Перевірила:**

проф. Федорова Н. В.

Київ – 2020

**Тема:** побудова дерева рішень за однією з вибраних областей.

**Мета:** побудувати дерево рішень за однією з вибраних областей на розв’язати задачі засобами Spark.

**Теоретичні відомості**

***Дерева рішень*** є одним з найбільш ефективних інструментів інтелектуального аналізу даних і самий корінь аналітики, які дозволяють вирішувати завдання класифікації і регресії.

Вони являють собою ієрархічні деревоподібні структури, що складаються з вирішальних правил виду «Якщо ..., то ...». Правила автоматично генеруються в процесі навчання на навчальній множині і, оскільки вони формулюються практично на природній мові (наприклад, «Якщо обсяг продажів більше 1000 шт., То товар перспективний»), дерева рішень як аналітичні моделі більш вербалізуючою і інтерпретованих, ніж, скажімо, нейронні сіті. Оскільки правила в деревах рішень виходять шляхом узагальнення безлічі окремих спостережень (навчальних прикладів), що описують предметну область, то за аналогією з відповідним методом логічного висновку їх називають індуктивними правилами, а сам процес навчання - індукцією дерев рішень. Власне, саме дерево рішень - це спосіб подачі вирішальних правил в ієрархічній структурі, що складається з елементів двох типів - вузлів (node) і листя (leaf). У вузлах знаходяться вирішальні правила і проводиться перевірка відповідності прикладів цього правила з якого-небудь атрибуту навчальної множини.

У найпростішому випадку, в результаті перевірки, безліч прикладів, які потрапили в вузол, розбивається на дві підмножини, в одне з яких потрапляють приклади, що задовольняють правилу, а в інше - що не задовольняють.

***Основна сфера застосування*** дерев рішень - підтримка процесів прийняття управлінських рішень, використовувана в статистиці, аналізі даних і машинному навчанні.

Завданнями, які розв'язуються за допомогою даного апарату, є:

- *класифікація* - віднесення об'єктів до одного з заздалегідь відомих класів. Цільова змінна повинна мати дискретні значення.

- *регресія* (чисельне пророкування) - прогноз числового значення незалежної змінної для заданого вхідного вектора.

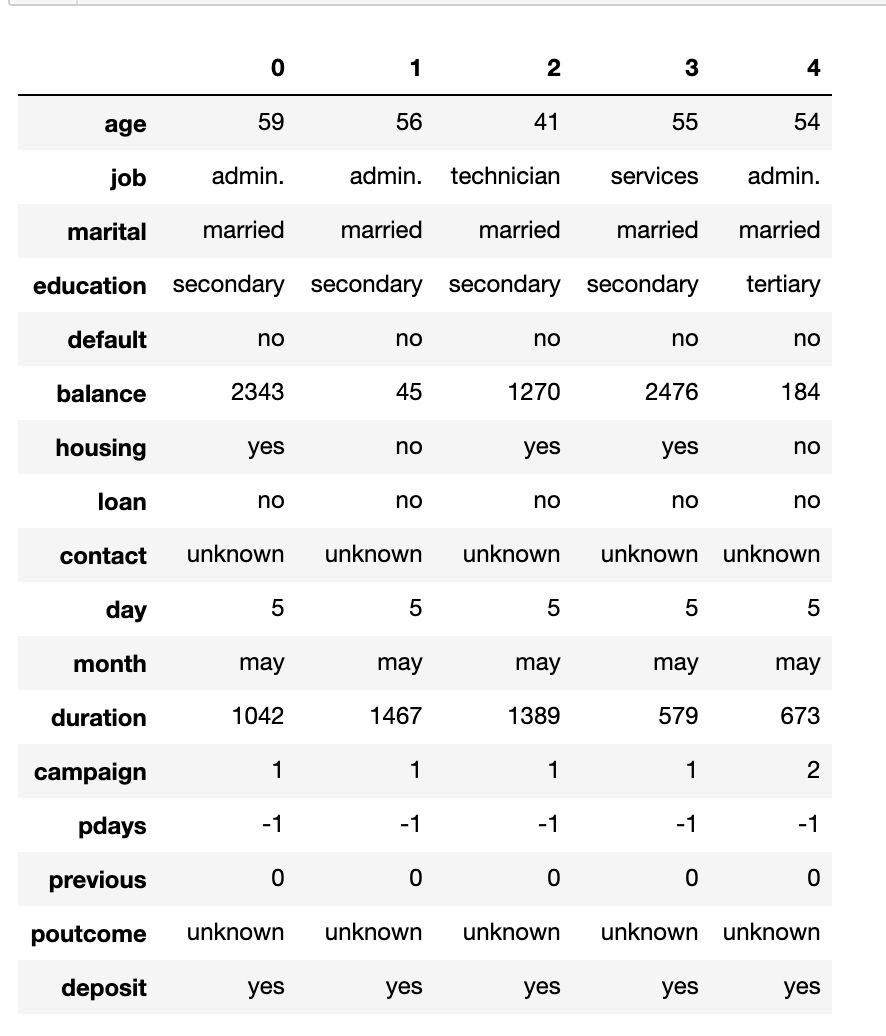
* *опис* *об'єктів* - набір правил в дереві рішень дозволяє компактно описувати об'єкти. Тому замість складних структур, що описують об'єкти, можна зберігати дерева рішень.

Процес побудови дерев рішень полягає в послідовному, рекурсивном розбитті навчальної множини на підмножини з застосуванням вирішальних правил в вузлах. Процес розбиття триває до тих пір, поки всі вузли в кінці всіх гілок НЕ будуть оголошені листям. Оголошення вузла листом може статися природним чином (коли він буде містити єдиний об'єкт, або об'єкти тільки одного класу), або після досягнення деякого умови зупинки, що задається користувачем (наприклад, мінімально допустиму кількість прикладів в вузлі або максимальна глибина дерева).

Дана лабораторна робота була виконана з використанням мови програмування Python.

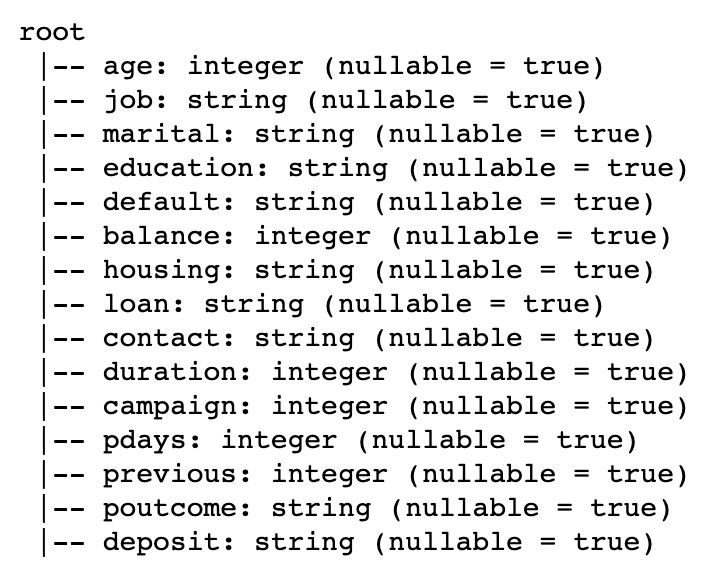
**Хід роботи**

1. Було обрано дані датасета банківського закладу з сайту https://www.kaggle.com/rouseguy/bankbalanced/data. Метою даної класифікації є передбачення чи підпишеться клієнт на депозит чи ні.
2. Відображення коду та фрагменту даних датасета:



**from** pyspark.sql **import** SparkSession  
  
spark = SparkSession.builder.appName(**'ml-bank'**).getOrCreate()  
df = spark.read.csv(**'bank.csv'**, header=**True**, inferSchema=**True**)  
**import** pandas **as** pd  
  
pd.DataFrame(df.take(5), columns=df.columns).transpose()

1. Для кожної ознаки виводимо базову статистику:

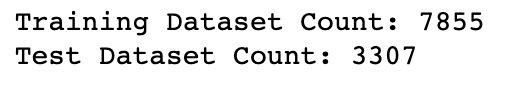


numeric\_features = [t[0] **for** t **in** df.dtypes **if** t[1] == **'int'**]  
df.select(numeric\_features).describe().toPandas().transpose()  
  
df = df.select(**'age'**, **'job'**, **'marital'**, **'education'**, **'default'**, **'balance'**, **'housing'**, **'loan'**, **'contact'**, **'duration'**,  
 **'campaign'**, **'pdays'**, **'previous'**, **'poutcome'**, **'deposit'**)  
cols = df.columns  
df.printSchema()

1. Для подання на вхід нашій моделі, закодуємо наші ознаки за допомогою VectorAssembler, OneHotEncoder та StringIndexer:

**from** pyspark.ml.feature **import** StringIndexer, VectorAssembler  
**from** pyspark.ml.feature **import** OneHotEncoder  
  
categoricalColumns = [**'job'**, **'marital'**, **'education'**, **'default'**, **'housing'**, **'loan'**, **'contact'**, **'poutcome'**]  
stages = []  
**for** categoricalCol **in** categoricalColumns:  
 stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + **'Index'**)  
 encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol +  
 **"classVec"**])  
 stages += [stringIndexer, encoder]  
 label\_stringIdx = StringIndexer(inputCol=**'deposit'**, outputCol=**'label'**)  
 stages += [label\_stringIdx]  
 numericCols = [**'age'**, **'balance'**, **'duration'**, **'campaign'**, **'pdays'**, **'previous'**]  
 assemblerInputs = [c + **"classVec" for** c **in** categoricalColumns] + numericCols  
 assembler = VectorAssembler(inputCols=assemblerInputs, outputCol=**"features"**)  
 stages += [assembler]  
  
**from** pyspark.ml **import** Pipeline  
  
pipeline = Pipeline(stages=stages)  
pipelineModel = pipeline.fit(df)  
df = pipelineModel.transform(df)  
selectedCols = [**'label'**, **'features'**] + cols  
df = df.select(selectedCols)

1. Розділимо дані на тренувальну та тестувальну вибірки:



train, test = df.randomSplit([0.5, 0.3], seed=2018)  
print(**"Training Dataset Count: "** + str(train.count()))  
print(**"Test Dataset Count: "** + str(test.count()))

1. Тепер побудуємо наш класифікатор. Так як модель DecisionTree має недолік перенавчання та майже ніде не використовується, а також відповідно до Вашої рекомендації об’єднання лабораторних робіт, мною була вибрана модель **RandomForest («Випадковий ліс»)**. Ця модель являє собою ансамбль дерев рішень, які тренуються на різних випадкових вибірках тренувального датасета. При евалюації моделі результатом передбачення є середьно-зважувана сума всіх передбачень кожного дерева в цій системі.

**from** pyspark.ml.classification **import** RandomForestClassifier  
  
rf = RandomForestClassifier(featuresCol=**'features'**, labelCol=**'label'**)  
rfModel = rf.fit(train)  
predictions = rfModel.transform(test)

1. Тепер протестуємо нашу натреновану модель на тестувальній вибірці. В якості метрики оцінювання було використано площу під кривою RocAuc, яка представляє з себе оцінювання співвідношення TruePositive rate та FalsePositive rate. У відмінності від метрики accuracy нам для цього підходу не потрібно підбирати поріг для дискретної класифікації. Значення 0.5 для цієї метрики означає, що наша модель видає випадкові передбачення, а значення 1.0 - означає ідеальний класифікатор. Точність нашої моделі 82.7%, що є досить непоганим результатом.

C:\Users\Dima Kruhlyi\Desktop\1.jpg

evaluator = BinaryClassificationEvaluator()  
print(**"Test Area Under ROC: "** + str(evaluator.evaluate(predictions, {evaluator.metricName: **"areaUnderROC"**})))

**Висновок:** під час виконання лабораторної роботи було реалізовано програму для обробки банківських даних та було натреновано модель ансамблю дерев рішень RandomForest для передбачення того чи візьме клієнт депозит чи ні. Точність алгоритму набула значення у 82.7%, що є досить точним значенням.