Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Лабораторна робота №4

з дисципліни «Технології паралельного програмування»

за темою «Big Data з використанням засобів Apache Spark»

**Виконали:**

Студент групи ІС-11мн

Іванов Анатолій Ігорович

Студент групи ІА-11мн

Новиков Данило Михайлович

**Перевірив:**

вик. Жереб Константин Андрійович

Київ 2022

**Тема:** Big Data з використанням засобів Apache Spark.

**Завдання:** Необхідно реалізувати вирішення обраної задачі з використанням технології Apache Spark. Можна запустити реалізацію локально, але якщо є можливість запустити на розподіленій системі – за це можна отримати додаткові бали. Порівняти реалізації лабораторних робіт №3 та №4. Результатом виконання даної лабораторної роботи є працююча програма, а також звіт про використані технології та можливості, з результатами вимірів.

**Хід роботи**

1. По-перше, встановимо spark. Оскільки його встановлення є стандартним процесом, на який є багато гайдів в мережі інтернет, його встановлення не відображено у звіті.
2. По-друге, підвищимо рівень логування spark до warn, щоб бачити менше зайвої інформації.
3. Створимо програму, що буде відтворювати дії програми у лабораторній роботі №3. Для цього також напишемо програму для підрахунку кількості слів, але цього разу використовуючи spark, а саме його версію, що працює з Python – pyspark. Вихідний код даної програми:

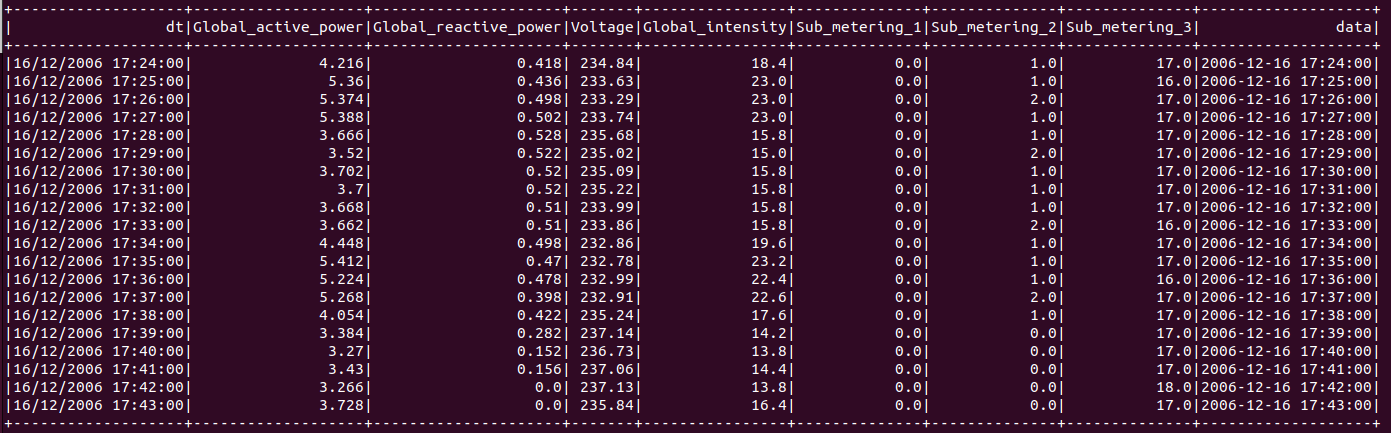


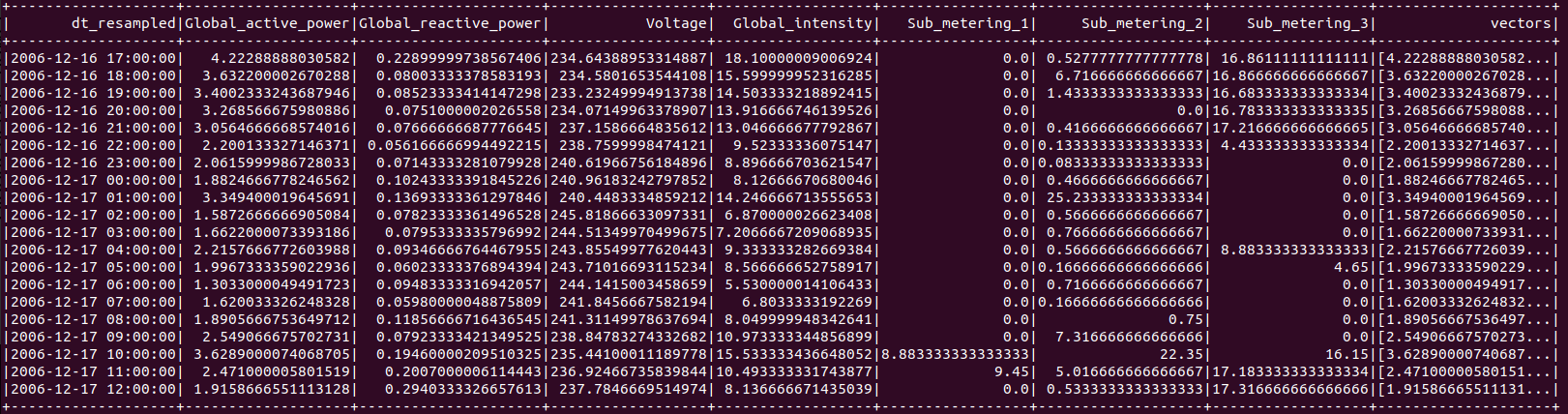
|  |  |
| --- | --- |
|  |  |

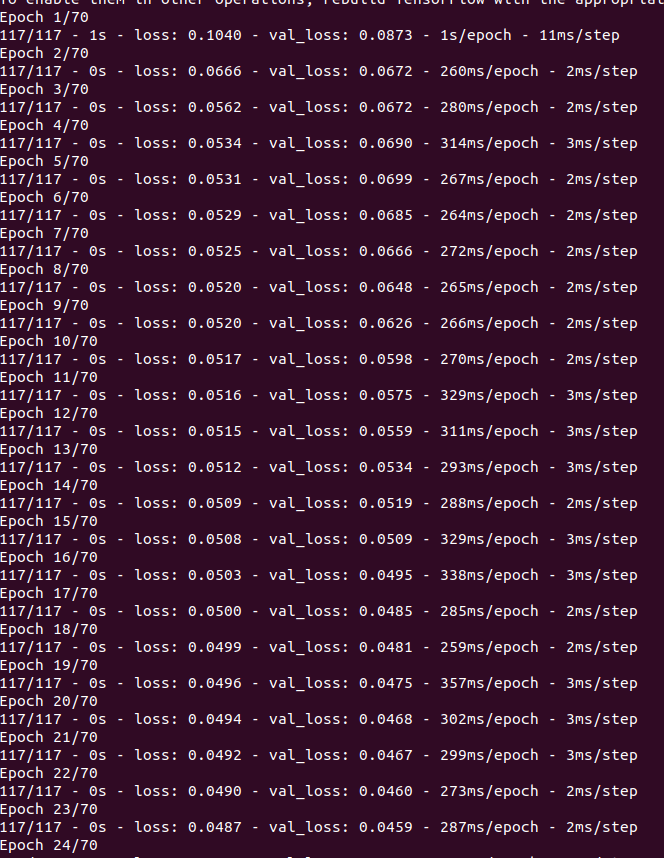
1. Виконаємо її, викоистовуючи як дані файл з попередньої лабораторної. Отримаємо у консолі наступний результат:
2. Також першим значущим словом отримаємо слово said. Оскільки в даному випадку було зроблено кращу обробку тексту, то кількість входжень трохи більша, але в цілому воно знаходиться в 10% від попереднього значення.
3. Також, для більшого відображення потужності spark у порівнянні з Hadoop запустимо програму, написану на одному з попереднії курсів з іншого предмету, що використовуємо машине навчання для підрахунку кількості використання електроенергії підприємством. Для неї був використаний датасет, що містить більше 200000 записів та знаходиться за посиланням: <http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption>
4. Вихідний код даної програми:

|  |
| --- |
| from pyspark.ml.feature import Bucketizer  import numpy  from pyspark.sql.types import \*  from pyspark.sql.functions import \*  from pyspark.sql import functions as F  import sys  from pyspark.sql import SparkSession  import functools  from pyspark.ml.feature import MinMaxScaler  from pyspark.ml.feature import VectorAssembler  from pyspark.ml.linalg import Vectors, VectorUDT  spark = SparkSession.builder.appName("coursach").getOrCreate()  fileName = 'file:///home/anatolii/hadoop-examples/Coursach/test.csv'  schema = StructType([  StructField("dt", StringType(), False),  StructField("Global\_active\_power", FloatType(), False),  StructField("Global\_reactive\_power", FloatType(), False),  StructField("Voltage", FloatType(), False),  StructField("Global\_intensity", FloatType(), False),  StructField("Sub\_metering\_1", FloatType(), False),  StructField("Sub\_metering\_2", FloatType(), False),  StructField("Sub\_metering\_3", FloatType(), False)  ])  df1 = spark.read.csv(fileName, schema=schema, header=True).withColumn('data', to\_timestamp('dt'))  df1.show()  def resample(column, agg\_interval=3600, time\_format='yyyy-MM-dd HH:mm:ss'):  if type(column)==str:  column = F.col(column)  col\_ut = F.unix\_timestamp(column, format=time\_format)  col\_ut\_agg = F.floor(col\_ut / agg\_interval) \* agg\_interval  return F.from\_unixtime(col\_ut\_agg)  resdf = df1.withColumn('dt\_resampled', resample(df1.dt, agg\_interval=3600))  resdf.createOrReplaceTempView("eData")  grouped\_by\_hour = spark.sql('SELECT dt\_resampled, MEAN(Global\_active\_power), MEAN(Global\_reactive\_power), MEAN(Voltage), MEAN(Global\_intensity), MEAN(Sub\_metering\_1), MEAN(Sub\_metering\_2), MEAN(Sub\_metering\_3) FROM eData GROUP BY dt\_resampled ORDER BY dt\_resampled')  df\_by\_hour = grouped\_by\_hour.toDF('dt\_resampled', 'Global\_active\_power', 'Global\_reactive\_power', 'Voltage', 'Global\_intensity', 'Sub\_metering\_1', 'Sub\_metering\_2', 'Sub\_metering\_3')  df\_by\_hour.show()  # Vectorize columns  assembler = VectorAssembler(inputCols = ['Global\_active\_power', 'Global\_reactive\_power', 'Voltage', 'Global\_intensity', 'Sub\_metering\_1', 'Sub\_metering\_2', 'Sub\_metering\_3'], outputCol = 'vectors')  df\_transformed = assembler.transform(df\_by\_hour)  df\_transformed.show()  # MinMaxScaler  mmScaler = MinMaxScaler(outputCol="scaled")  mmScaler.setInputCol("vectors")  model = mmScaler.fit(df\_transformed)  df\_scaled = model.transform(df\_transformed)  df\_scaled.show()  columns = ['Global\_active\_power', 'Global\_reactive\_power', 'Voltage', 'Global\_intensity', 'Sub\_metering\_1', 'Sub\_metering\_2', 'Sub\_metering\_3']  df\_prepared = df\_scaled.rdd.map(lambda x: [float(y) for y in x['scaled']]).toDF(columns)  assembler = VectorAssembler(inputCols = ['Global\_reactive\_power', 'Voltage', 'Global\_intensity', 'Sub\_metering\_1', 'Sub\_metering\_2', 'Sub\_metering\_3'], outputCol = 'features')  df\_prepared\_grouped = assembler.transform(df\_prepared).select('features', 'Global\_active\_power')  values = df\_prepared.toPandas().values  n\_train\_time = 365\*24\*2  train = values[:n\_train\_time, :]  test = values[n\_train\_time:, :]  import numpy as np  train\_X, train\_y = train[:, :-1], np.asarray(train[:, -1])  test\_X, test\_y = test[:, :-1], np.asarray(test[:, -1])  train\_X = np.asarray(train\_X.reshape((train\_X.shape[0], 1, train\_X.shape[1])))  test\_X = np.asarray(test\_X.reshape((test\_X.shape[0], 1, test\_X.shape[1])))  print(train\_X.shape, train\_y.shape, test\_X.shape, test\_y.shape)  from keras import models, layers  model = models.Sequential()  model.add(layers.LSTM(100, input\_shape=(train\_X.shape[1], train\_X.shape[2])))  model.add(layers.Dropout(0.2))  model.add(layers.Dense(1))  model.compile(loss='mean\_squared\_error', optimizer='adam')  # fit network validation\_data=(test\_X, test\_y), 70/100  history = model.fit(train\_X, train\_y, epochs=70, validation\_data=(test\_X, test\_y), batch\_size=150, verbose=2, shuffle=False)  import matplotlib.pyplot as plt  plt.plot(history.history['loss'])  plt.plot(history.history['val\_loss'])  plt.title('model loss')  plt.ylabel('loss')  plt.xlabel('epoch')  plt.legend(['train', 'test'], loc='upper right')  plt.show()  from sklearn.metrics import mean\_squared\_error,r2\_score  # prediction  yhat = model.predict(test\_X)  test\_X2 = test\_X.reshape((test\_X.shape[0], 6))  # invert scaling for forecast  inv\_yhat = np.concatenate((yhat, test\_X2[:, -6:]), axis=1)  inv\_yhat = inv\_yhat[:,0]  # invert scaling for actual  test\_y = test\_y.reshape((len(test\_y), 1))  inv\_y = np.concatenate((test\_y, test\_X2[:, -6:]), axis=1)  inv\_y = inv\_y[:,0]  # calculate RMSE  rmse = np.sqrt(mean\_squared\_error(inv\_y, inv\_yhat))  print('Test RMSE: %.3f' % rmse)  aa=[x for x in range(200)]  plt.plot(aa, inv\_y[:200], marker='.', label="actual")  plt.plot(aa, inv\_yhat[:200], 'r', label="prediction")  plt.ylabel('Global\_active\_power', size=15)  plt.xlabel('Time step', size=15)  plt.legend(fontsize=15)  plt.show() |

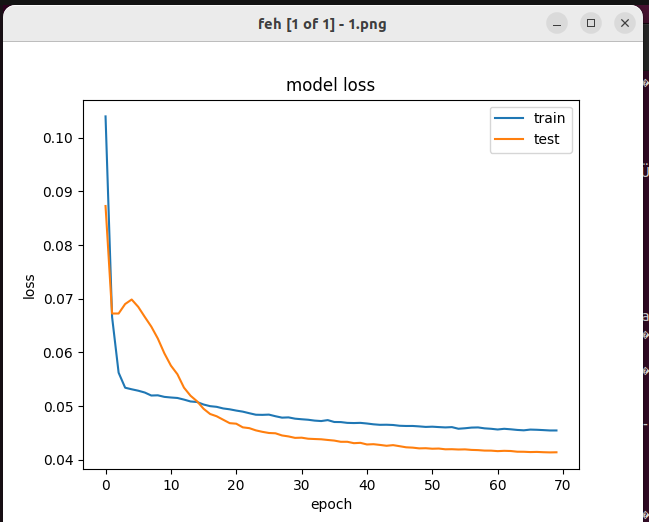
1. Результат виконання, що спочатку відображує вхідний датасет, після цього один з датасетів під час перетворення та після цього модель навчання за епохами.



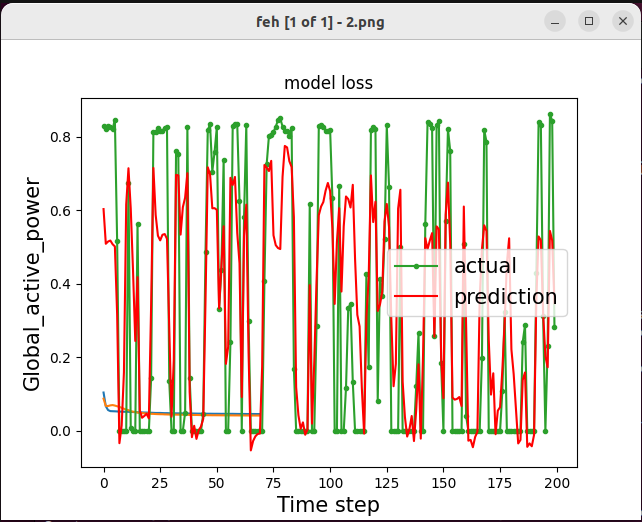




1. Графік навчання моделі на основних та тестових даних.



1. Графік передбачень та реальні значення.



**Висновок:**

в результаті виконання даної лабораторної роботи була створена програма підрахунку кількості входжень слів у тексті та програма, що використовує машине навчання для навчання моделі. Порівнюючи першу частину роботи з реалізацією Hadoop Map-reduce, можна побачити, що код, використовуючи spark є набагато простішим та читабельнішим, через те, що Spark має бібліотеки, що використовуються для більш зручної обробки табличних даних та перетворення текстових даних у табличні. При цьому, spark є надбудовою над Hadoop, тобто при створенні він будувався на основі Hadoop та не може працювати без нього.