ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA HỆ THỐNG THÔNG TIN

ĐỀ TÀI DỮ LIỆU LỚN

DỰ BÁO SỐ LƯỢNG THƯƠNG VONG Ở ANH TỪ NĂM 2005 ĐẾN 2007 BẰNG CÁCH SỬ DỤNG MÔ HÌNH LSTM

Giảng viên hướng dẫn: ThS. Nguyễn Hồ Duy Tri

Sinh viên thực hiện:

Lê Nguyễn Nhật Minh	21522338
Hoàng Nhật Minh	21522336
Trần Thị Luyến	21521107
Nguyễn Thị Thúy Hằng	21520822

TP. Hồ Chí Minh, tháng 12 năm 2024

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

LỜI CẢM ƠN

Lời đầu tiên nhóm chúng em xin gửi lời cảm ơn đến thầy Nguyễn Hồ Duy Tri–giảng viên môn Dữ liệu lớn Thầy đã nhiệt tình giảng dạy, tận tình hướng dẫn cho nhóm trong suốt quá trình thực hiện dự án. Nhờ đó, chúng em đã tiếp thu được nhiều kiến thức, kỹ năng ứng dụng vào dự án. Một lần nữa nhóm xin chân thành cảm ơn Thầy.

Chúng em rất biết ơn sự hướng dẫn chuyên môn và những kiến thức quý báu mà Thầy đã chia sẻ. Những kiến thức sâu sắc của Thầy đã giúp chúng em hiểu rõ hơn về kiến thức cũng như cách thực hành đề tài. Tuy nhiên, trong quá trình thực hiện, chúng em không tránh khỏi những thiếu sót. Chính vì vậy, nhóm chúng em rất mong nhận được những góp ý từ phía Thầy nhằm hoàn thiện cũng như nâng cao kiến thức môn học và chuẩn bị tốt cho các đề tài khác trong tương lai.

Cuối cùng, chúng em xin chân thành cảm ơn thầy, kính chúc cô sức khỏe và thành công trong những dự án tương lai.

Thành phố Hồ Chí Minh, tháng 12 năm 2024

Nhóm 5

DANH MỤC BẢNG BIỂU

DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt	Từ đầy đủ
1	EDA	Exploratory Data Analyst

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

1.1. Lý do chọn đề tài

Tai nạn giao thông là một trong những nguyên nhân hàng đầu gây ra tử vong và thương tích trên toàn thế giới. Theo báo cáo của Tổ chức Y tế Thế giới (WHO), tai nạn giao thông đã cướp đi sinh mạng của khoảng 1,35 triệu người mỗi năm và là nguyên nhân gây ra hơn 50 triệu ca thương tích nghiêm trọng [1]. Điều này không chỉ gây thiệt hại về mặt nhân mạng mà còn ảnh hưởng sâu rộng đến kinh tế xã hội. Tại Vương quốc Anh, số liệu của Bộ Giao thông Vận tải (Department for Transport) cho thấy trong năm 2020, có hơn 1.460 người thiệt mạng và gần 15.600 người bị thương trong các vụ tai nạn giao thông [2]. Con số này cho thấy mức độ nghiêm trọng của vấn đề và nhu cầu cấp thiết trong việc cải thiện công tác dự báo và phòng ngừa tai nạn giao thông.

Với sự phát triển mạnh mẽ của công nghệ và dữ liệu lớn (Big Data), việc áp dụng các phương pháp phân tích dữ liệu và học máy (Machine Learning) vào dự báo và phòng ngừa tai nạn giao thông đã trở thành một hướng đi tiềm năng. Một trong những công nghệ hứa hẹn nhất trong việc giải quyết vấn đề này là các mô hình học sâu (Deep Learning), đặc biệt là mô hình Long Short-Term Memory (LSTM). Mô hình LSTM có khả năng xử lý và phân tích các dữ liệu chuỗi thời gian, giúp phát hiện các xu hướng tiềm ẩn trong quá trình xảy ra tai nạn, qua đó dự báo khả năng tai nạn trong tương lai một cách chính xác. Trong bối cảnh đó, nghiên cứu "Dự báo về khả năng xảy ra tai nạn giao thông tại Anh trong năm 2005 – 2007 bằng cách sử dụng mô hình LSTM" không chỉ có ý nghĩa lý thuyết mà còn mang tính ứng dụng cao trong thực tế. Nghiên cứu này sẽ mở ra các giải pháp mới trong việc sử dụng công nghệ để cải thiện an toàn giao thông, đồng thời giảm thiểu những hậu quả đáng tiếc từ tai nạn giao thông.

1.2. Mục tiêu đề tài

Mục tiêu chính của nghiên cứu là sử dụng mô hình học sâu Long Short-Term Memory (LSTM) để dự báo khả năng xảy ra tai nạn giao thông tại Anh trong giai đoạn 2005-2007. Mô hình LSTM, một dạng mạng nơ-ron hồi tiếp (RNN), được lựa chọn nhờ khả năng xử lý dữ liệu chuỗi thời gian hiệu quả, giúp nhận diện các mẫu và xu hướng trong dữ liệu tai

nạn giao thông. Mục tiêu của đề tài là phát triển một hệ thống có khả năng dự báo tai nạn giao thông, từ đó đưa ra những cảnh báo sớm và các biện pháp phòng ngừa.

Cụ thể, nghiên cứu sẽ thực hiện:

- Phân tích các yếu tố ảnh hưởng đến tai nạn giao thông như thời gian, địa điểm, điều kiện thời tiết và các yếu tố liên quan đến người tham gia giao thông.
- Xây dựng mô hình dự báo tai nạn giao thông bằng LSTM.
- Đánh giá hiệu quả của mô hình và so sánh kết quả với các phương pháp khác.

1.3. Cấu trúc báo cáo

Báo cáo này được chia thành các chương như sau:

- **Chương 1: Tổng quan đề tài**: Trình bày lý do chọn đề tài, mục tiêu nghiên cứu, và cấu trúc báo cáo.
- **Chương 2: Mô tả về tập dữ liệu**: Cung cấp chi tiết về bộ dữ liệu "1.5 million UK Traffic Accidents EDA 2005-2007", các thuộc tính của dữ liệu và quá trình phân tích dữ liệu.
- **Chương 3: Cơ sở lý thuyết**: Giới thiệu các kiến thức lý thuyết cơ bản, bao gồm khái niệm về phân tán và mô hình LSTM.
- **Chương 4: Áp dụng thuật toán khai thác dữ liệu LSTM**: Trình bày chi tiết về quá trình áp dụng mô hình LSTM để khai thác và dự báo khả năng xảy ra tai nạn giao thông.
- **Chương 5: Kết quả đạt được**: Đưa ra kết quả thực nghiệm, so sánh và đánh giá mô hình.
- **Chương 6: Kết luận**: Tổng kết các kết quả nghiên cứu, đề xuất những hướng phát triển và cải tiến trong tương lai.

Mục tiêu cuối cùng của báo cáo là cung cấp một cái nhìn toàn diện về khả năng ứng dụng mô hình học sâu LSTM trong việc dự báo tai nạn giao thông, cũng như đóng góp vào việc phát triển các phương pháp và công cụ hỗ trợ an toàn giao thông trong tương lai.

CHƯƠNG 2. MÔ TẢ VỀ TẬP DỮ LIỆU

- Nguồn của bộ dữ liệu: [1.5 million UK Traffic Accidents EDA 2005 – 2007]
- Tên bộ dữ liệu: **1.5 Million UK Traffic Accidents EDA** (1,5 triệu vụ tai nạn giao thông ở Anh EDA)

Hình 2.1: Tập dữ liệu tai nạn tại Anh lấy từ Kaggle

2.1. Chi tiết về tập dữ liệu

- Ngày cập nhật gần nhất: 2017.
- Tên tác giả: Isidro Navarro Oporto & Dave Fisher-Hickey
- Kích thước: 163.58 MB với 33 cột thuộc tính và ≈ 570000 dòng dữ liệu
- Nội dung: Bộ dữ liệu ghi lại thông tin giao thông và tai nạn giao thông tại Vương quốc Anh từ năm 2005 đến 2007. Dữ liệu bao gồm hơn 570000 vụ tai nạn được ghi nhận từ báo cáo của cảnh sát và thông tin chi tiết về lưu lượng giao thông trên các tuyến đường chính, dữ liệu này cung cấp cái nhìn toàn diện để nghiên cứu về xu hướng giao thông, an toàn đường bộ và phân tích cơ sở hạ tầng.
- Ý tưởng khai thác bộ dữ liệu:
  - Lưu lượng giao thông thay đổi đã ảnh hưởng đến tai nạn như thế nào?
  - Có thể dự đoán tỷ lệ tai nạn theo thời gian không? Những gì có thể cải thiện tỷ lệ này?
  - Vẽ bản đồ tương tác về xu hướng thay đổi, ví dụ:
    - London đã thay đổi như thế nào đối với người đi xe đạp?

- Những con đường đông đúc nhất ở Anh là gì?

- Khu vực nào không thay đổi và tại sao? Nhận diện các nhu cầu, thất bại và thành công về cơ sở hạ tầng.
- Các khu vực nông thôn và đô thị khác nhau như thế nào (xem RoadCategory)? Sự khác biệt giữa Anh, Scotland và Wales thì sao?
- Chính phủ Anh cũng thường quan tâm đến số dặm xe chạy. Bạn có thể tính bằng cách nhân AADF với độ dài đường (link length) và số ngày trong năm. Điều này nói gì về hệ thống đường bộ ở Anh?

2.2. Danh sách thuộc tính

Bảng 2-1: Danh sách các thuộc tính của tập dữ liệu

Trường	Kiểu dữ liệu	Giá trị/ Phạm vi	Ý nghĩa
Accident_Index	String		ID nhận dạng duy nhất của vụ tai nạn.
Location_Easting_OSGR	Float		Tọa độ x trong hệ lưới địa lý của Anh.
Location_Northing_OSGR	Float		Tọa độ y trong hệ lưới địa lý của Anh.
Longitude	Float		Kinh độ nơi xảy ra tai nạn.
Latitude	Float		Vĩ độ nơi xảy ra tai nạn.
Police_Force	Police_Force		Mã số lực lượng cảnh sát quản lý hiện trường tai nạn.
Accident_Severity	Integer	1 = Chết người, 2 = Nghiêm trọng, 3 = Nhẹ.	Mức độ nghiêm trọng của tai nạn.

Number_of_Vehicles	Integer	Số nguyên >= 1 (ví dụ: 2, 3, 5).	Tổng số phương tiện liên quan đến vụ tai nạn.
Number_of_Casualties	Integer	Số nguyên >= 0 (ví dụ: 0, 1, 2).	Tổng số người bị thương hoặc tử vong.
Date	Date	định dạng dd/mm/yyyy	Ngày xảy ra vụ tai nạn.
Day_of_Week	Integer	1 = Chủ Nhật, 2 = Thứ Hai, ... 7 = Thứ Bảy.	Ngày trong tuần khi tai nạn xảy ra.
Time	Time	Thời gian (định dạng HH:MM)	Thời điểm tai nạn xảy ra.
Local_Authority_(District)	Integer		Mã khu vực hành chính cấp quận chịu trách nhiệm.
Local_Authority_(Highway)	String		Khu vực chịu sự quản lý của cơ quan đường bộ địa phương.
1st_Road_Class	Integer	1 = Cao tốc, 2 = Đường hai chiều, 3 = Đường một chiều, ...	Loại đường chính xảy ra tai nạn.
1st_Road_Numbe r	Integer		Số nhận dạng đường chính.
Road_Type	String		Loại hình đường xảy ra tai nạn.
Speed_limit	Integer		Mức tốc độ tối đa cho phép tại đoạn đường xảy ra tai nạn (đơn vị: mph).

Junction_Detail	String		Chi tiết nút giao xảy ra tai nạn (nếu có).
Junction_Control	String		Hình thức kiểm soát giao thông tại nút giao.
2nd_Road_Class	Integer		Loại đường phụ liên quan đến tai nạn (nếu có).
2nd_Road_Number	Integer		Loại đường phụ liên quan đến tai nạn (nếu có).
Pedestrian_Crossing -Human_Control	String		Số nhận dạng đường phụ.
Pedestrian_Crossing - Physical_Facilities	String		Loại cơ sở vật chất dành cho người đi bộ tại nơi xảy ra tai nạn.
Light_Conditions	String		Điều kiện ánh sáng tại hiện trường.
Weather_Conditions	String		Tình trạng thời tiết tại hiện trường.
Road_Surface_Conditions	String		Tình trạng mặt đường tại nơi xảy ra tai nạn.
Special_Conditions_at_Site	String		Các vấn đề đặc biệt tại nơi xảy ra tai nạn (nếu có).
Carriageway_Hazards	String		Các vật cản xuất hiện trên mặt đường.

Urban_or_Rural_ Area	Integer	1 = Đô thị, 2 = Nông thôn.	Loại khu vực xảy ra tai nạn.
Did_Police_Officer _Attend_Scene_of _Accident	Boolean	1 = Có, 0 = Không.	Xác định cảnh sát có mặt tại hiện trường hay không.
LSOA_of_ Accident_Location	String		Khu vực hành chính chi tiết nơi xảy ra tai nạn.
Year	Integer	Năm xảy ra tai nạn (2005-2014).	Thời gian xảy ra tai nạn được ghi nhận theo năm.

2.3. Phân tích dữ liệu

2.3.1. Import thư viện và dataset

```
from pyspark.sql import SparkSession
from pyspark import SparkContext, SparkConf
from pyspark.sql.functions import split, col, lit, udf, array, avg, lag, year, month
from pyspark.sql.types import IntegerType, DoubleType, ArrayType, StructType, StructField, FloatType, StringType
from pyspark.sql import functions as F, Row
import matplotlib.pyplot as plt
from pyspark.ml.feature import VectorAssembler, MinMaxScaler
from pyspark.sql.window import Window
from matplotlib.backends.backend_pdf import PdfPages
import seaborn as sns
import pandas as pd
import numpy as np
from matplotlib.dates import DateFormatter
from pyspark.ml.linalg import Vectors
from datetime import datetime, timedelta
```

Hình 2.2: Đoạn mã import thư viện

```
spark = SparkSession.builder.appName("ProjectFinal").getOrCreate()
data = spark.read.csv("file:///home/minh/Desktop/Final_BigData_UIT/accident_data.csv", header = True, inferSchema = True)

data.printSchema()
```

Hình 2.3: Đoạn mã thêm dữ liệu vào mô hình

2.3.2. Giá trị lớn nhất, nhỏ nhất

Để hiểu rõ hơn về phạm vi và phân bố của các dữ liệu số trong bộ dữ liệu tai nạn giao thông, chúng tôi đã thực hiện việc tính toán giá trị lớn nhất (max) và nhỏ nhất (min) cho từng cột có kiểu dữ liệu số.

Những cột này bao gồm các thông tin như: "Number_of_Vehicles" (Số lượng phương tiện tham gia), "Number_of_Casualties" (Số người bị thương và tử vong) và "Speed_limit" (Giới hạn tốc độ).

```
numeric_columns = ["Number_of_Vehicles", "Number_of_Casualties", "Speed_limit"]

df_double = data.select([F.col(col).cast(DoubleType()).alias(col) if col in numeric_columns else col for col in data.columns])

min_max_df = df_double.agg(*[F.min(col).alias(f"{col}_min") for col in numeric_columns],
                           *[F.max(col).alias(f"{col}_max") for col in numeric_columns])

min_max_rows = []
for col in numeric_columns:
    min_val = min_max_df.select(F.col(f"{col}_min")).first()[0]
    max_val = min_max_df.select(F.col(f"{col}_max")).first()[0]
    min_max_rows.append(Row(Column=col, Min_Value=min_val, Max_Value=max_val))

min_max_df_readable = spark.createDataFrame(min_max_rows)
min_max_df_readable.show()
```

Hình 2.4: Đoạn mã sử dụng để phân tích dữ liệu Min, Max của tập dữ liệu

```
+--------------------+---------+---------+
|              Column|Min_Value|Max_Value|
+--------------------+---------+---------+
|  Number_of_Vehicles|      1.0|     28.0|
|Number_of_Casualties|      1.0|     68.0|
|         Speed_limit|     10.0|     70.0|
+--------------------+---------+---------+
```

Hình 2.5: Kết quả khi tiến hành phân tích Min, Max các trường dữ liệu

Kết quả tính toán giá trị lớn nhất và nhỏ nhất cho từng cột được trình bày như hình 2.3 cho thấy rằng:

- **Number_of_Vehicles** (Số lượng phương tiện tham gia): Giá trị nhỏ nhất là 1, có thể là các vụ tai nạn đơn phương, và giá trị lớn nhất là 28, cho thấy có những vụ tai nạn liên quan đến nhiều phương tiện.
- **Number_of_Casualties** (Số người bị thương và tử vong): Giá trị nhỏ nhất là 1, trong khi giá trị lớn nhất là 68, cho thấy số người bị ảnh hưởng trong một số vụ tai nạn có thể rất lớn.

- **Speed_limit** (Giới hạn tốc độ): Các giá trị trong cột này dao động từ 10 km/h đến 70 km/h, cho thấy các vụ tai nạn có thể xảy ra ở nhiều mức độ giới hạn tốc độ khác nhau trên các tuyến đường.

Những giá trị này cung cấp thông tin quan trọng giúp phân tích sự phân bố của tai nạn giao thông và có thể hỗ trợ việc xây dựng các mô hình dự báo hiệu quả hơn.

2.3.3. Giá trị phổ biến nhất và hiếm nhất

Trong phần này, chúng tôi thực hiện phân tích giá trị phổ biến nhất (mode) và giá trị hiếm nhất (least common) cho từng cột trong bộ dữ liệu. Kết quả thu được cho phép nhận diện các giá trị xuất hiện với tần suất cao nhất và thấp nhất, giúp hiểu rõ hơn về sự phân bố dữ liệu.

```
results = []

for col in data.columns:
    freq_df = data.groupBy(col).count().orderBy(F.col("count").desc())

    most_common = freq_df.first()
    most_common_value = most_common[0]
    most_common_count = most_common[1]

    least_common = freq_df.orderBy(F.col("count").asc()).first()
    least_common_value = least_common[0]
    least_common_count = least_common[1]

    results.append((col, most_common_value, most_common_count, least_common_value, least_common_count))

mode_df = spark.createDataFrame(results, ["Column", "Most_Common_Value", "Most_Common_Count", "Least_Common_Value", "Least_Common_Count"])
mode_df.show(34)
```

Hình 2.6: Đoạn mã sử dụng để phân tích giá trị phổ biến nhất và hiếm nhất của tập dữ liệu

Column	Most_Common_Value	Most_Common_Count	Least_Common_Value	Least_Common_Count
Accident_Index	2.01E+12	203663	200501BS70192	1
Location_Easting_...	533650	116	606890	1
Location_Northing...	181050	117	485190	1
Longitude	NULL	101	-0.184017	1
Latitude	NULL	101	51.52561	1
Police_Force	1	73899	48	1007
Accident_Severity	3	487161	1	8553
Number_of_Vehicles	2	336662	22	1
Number_of_Casualties	1	432213	28	1
Date	21/10/2005	822	25/12/2007	157
Day_of_Week	6	93994	1	63739
Time	java.util.Gregori...	5500	java.util.Gregori...	12
Local_Authority_(...	300	11713	933	107
Local_Authority_(...	E10000016	14383	E06000053	7
1st_Road_Class	3	253965	2	1368
1st_Road_Number	0	160449	3749	1
Road_Type	Single carriageway	423414	Unknown	4316
Speed_limit	30	358591	15	7
Junction_Detail	NULL	570011	NULL	570011
Junction_Control	Giveway or uncont...	271570	Authorised person	1055
2nd_Road_Class	-1	241676	2	438
2nd_Road_Number	0	443236	5803	1
Pedestrian_Crossi...	None within 50 me...	566636	NULL	17
Pedestrian_Crossi...	No physical cross...	486482	NULL	34
Light_Conditions	Daylight: Street ...	414282	Darkness: Street ...	2467
Weather_Conditions	Fine without high...	454074	NULL	20
Road_Surface_Cond...	Dry	389897	NULL	662
Special_Condition...	None	555325	NULL	11
Carriageway_Hazards	None	558858	NULL	23
Urban_or_Rural_Area	1	360761	3	132
Did_Police_Office...	Yes	459311	NULL	2375
LSOA_of_Accident_...	NULL	47511	E01013112	1
Year	2005	198735	2007	182115

Hình 2.7: Kết quả khi tiến hành phân tích giá trị phổ biến nhất và hiếm nhất của các trường dữ liệu

Kết quả tính toán giá trị lớn nhất và nhỏ nhất cho từng cột được trình bày như hình 2.5 cho thấy rằng:

- **Accident_Index**: Giá trị phổ biến nhất là "2.01E+12", xuất hiện 203,663 lần, trong khi giá trị hiếm nhất là "200501BS70192", xuất hiện chỉ 1 lần.
- **Location_Easting và Location_Northing**: Các giá trị phổ biến nhất là "533650" và "181050" lần lượt với tần suất cao nhất, trong khi các giá trị hiếm nhất xuất hiện chỉ 1 lần.
- **Police_Force**: Lực lượng cảnh sát có giá trị phổ biến nhất là "1", xuất hiện 73,899 lần, trong khi giá trị hiếm nhất là "48" với tần suất 1,007 lần.
- **Accident_Severity**: Mức độ nghiêm trọng của tai nạn có giá trị phổ biến nhất là "3" (tần suất cao nhất là 487,161), trong khi giá trị "1" (mức độ thấp nhất) hiếm gặp hơn với tần suất 8,553.

- **Day_of_Week**: Ngày trong tuần có giá trị phổ biến nhất là "6" (Thứ Bảy) với 93,994 lần xuất hiện, trong khi ngày "1" (Thứ Hai) ít phổ biến hơn với 63,739 lần.
- **Speed_limit**: Giá trị phổ biến nhất là "30" km/h, chiếm 358,591 lần, trong khi giá trị hiếm nhất là "15" km/h với chỉ 7 lần.

Những phân tích trên cung cấp cái nhìn sâu sắc về dữ liệu và có thể giúp trong việc tìm hiểu các yếu tố liên quan đến sự xuất hiện của tai nạn giao thông.

Những phân tích này cung cấp cái nhìn sâu sắc về sự phân bố của các vụ tai nạn giao thông và có thể giúp ích trong việc phát triển các chiến lược dự báo tai nạn hoặc cải thiện các biện pháp an toàn giao thông.

2.3.4. Giá trị trung bình và trung vị

```
numeric_columns = ["Number_of_Vehicles", "Number_of_Casualties","Speed_limit"]

stats = []

for col in numeric_columns:
    mean_value = data.select(F.mean(col).alias("mean")).first()["mean"]
    median_value = data.approxQuantile(col, [0.5], 0.01)[0]
    stats.append((col, mean_value, median_value))

result_df = spark.createDataFrame(stats, ["Column", "Mean", "Median"])
result_df.show()
```

Hình 2.8: Đoạn mã sử dụng để phân tích giá trị giá trị trung bình và trung vị của tập dữ liệu

```
+--------------------+------------------+------+
|              Column|              Mean|Median|
+--------------------+------------------+------+
|  Number_of_Vehicles|1.8407732482355603|   2.0|
|Number_of_Casualties|1.3634842134625471|   1.0|
|         Speed_limit| 39.78832864628928|  30.0|
+--------------------+------------------+------+
```

Hình 2.9: Kết quả khi tiến hành phân tích giá trị giá trị trung bình và trung vị của các trường dữ liệu

Kết quả tính toán giá trị trung bình và trung vị cho từng cột được trình bày như hình 2.7 cho thấy rằng:

- Phần lớn các vụ tai nạn liên quan đến 2 phương tiện và chỉ có 1 người bị thương hoặc tử vong, cho thấy tính chất của các tai nạn thường là va chạm nhẹ.

- Tai nạn thường xảy ra ở các khu vực giới hạn tốc độ thấp, nhưng những tai nạn ở tốc độ cao vẫn tồn tại, làm tăng giá trị trung bình.

2.4. Trực quan hóa dữ liệu

Tạo bảng tạm thời từ dataframe với tên là “road_accidents”

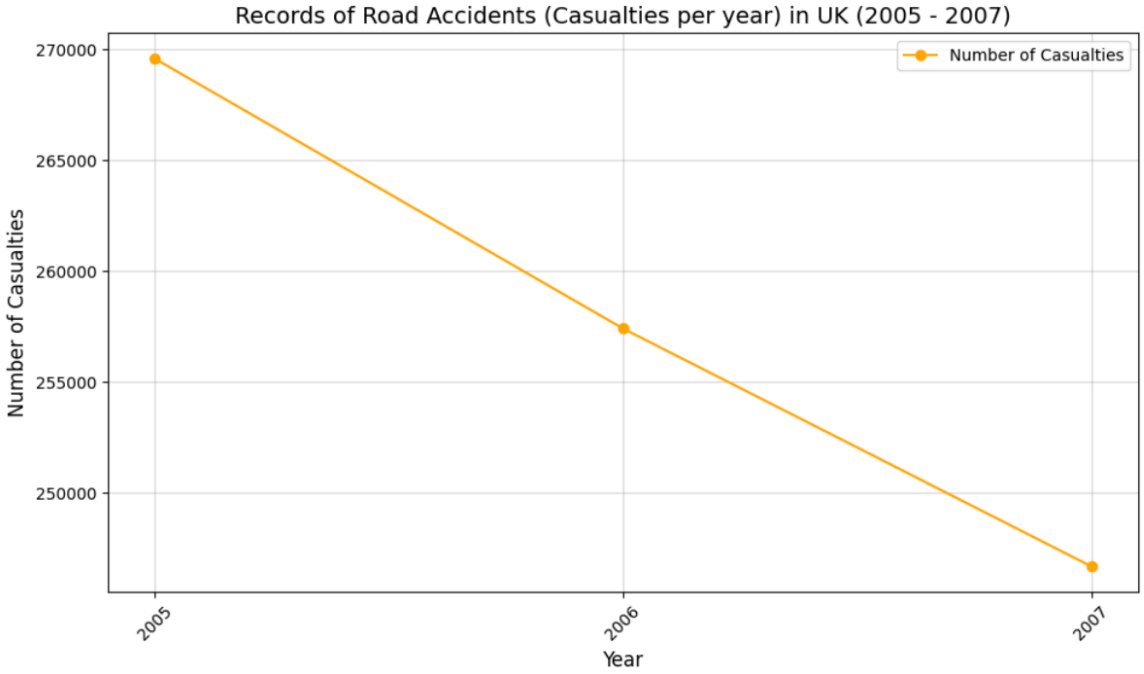
```
data.createOrReplaceTempView("road accidents")
```

Hình 2.10: Đoạn mã tạo dataframe

2.4.1. Thống kê số lượng nạn nhân ở Anh từ 2005 đến 2007

```
query = """
    SELECT
        Year,
        SUM(Number_of_Casualties) as total_casualties,
        COUNT(Number_of_Casualties) as accident_count,
        AVG(Number_of_Casualties) as mean_casualties
    FROM road_accidents
    GROUP BY Year
    ORDER BY Year
"""
result_df = spark.sql(query)

result_pandas = result_df.toPandas()

plt.figure(figsize=(10, 6))
plt.plot(
    result_pandas["Year"],
    result_pandas["total_casualties"],
    marker="o", color="orange", label="Number of Casualties"
)
plt.title("Records of Road Accidents (Casualties per year) in UK (2005 - 2007)", fontsize=14)
plt.xlabel("Year", fontsize=12)
plt.ylabel("Number of Casualties", fontsize=12)
plt.xticks(result_pandas["Year"], rotation=45)
plt.grid(alpha=0.5)
plt.legend()
plt.tight_layout()

plt.show()
```

Hình 2.11: Đoạn mã thống kê số lượng nạn nhân ở Anh từ 2005 đến 2007



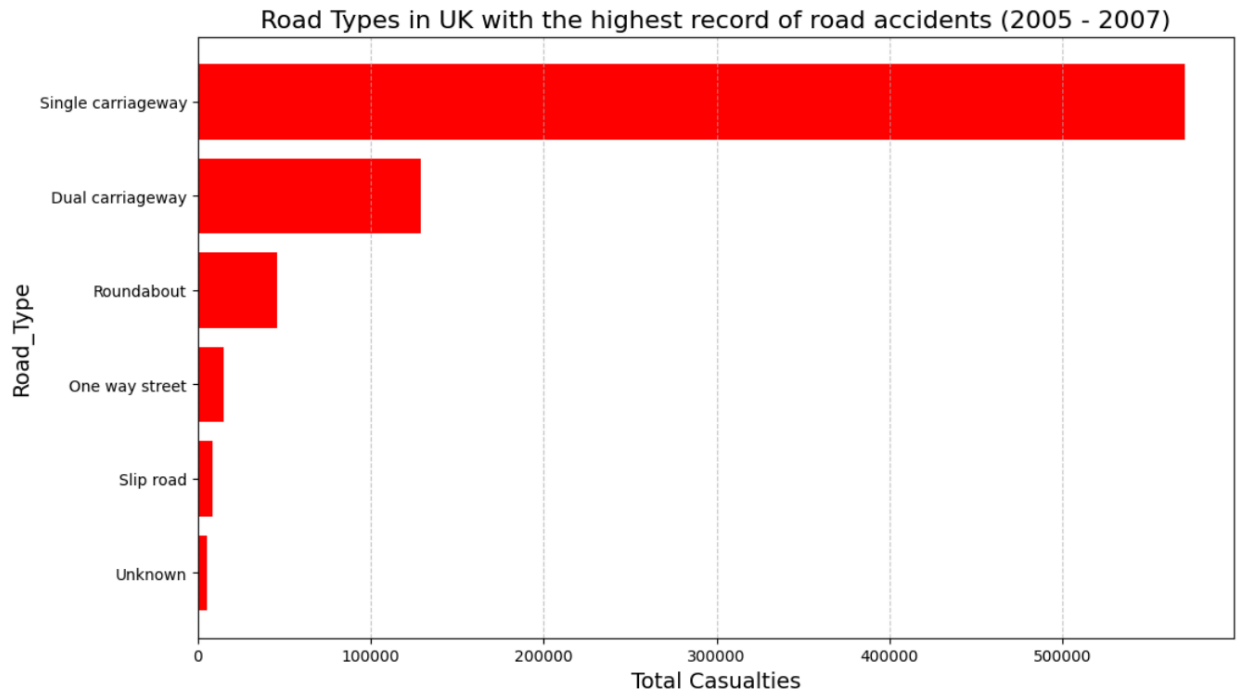
Hình 2.12: Kết quả thống kê số lượng nạn nhân ở Anh từ 2005 đến 2007

Từ kết quả, ta nhận thấy: Tỉ lệ tai nạn qua các năm đều giảm với năm 2005 đạt gần 270000 người thương vong và giảm tới 245000 người

2.4.2. Thống kê những đường thường xảy ra tai nạn từ 2005 đến 2007

```
query = """
SELECT
    Road_Type,
    SUM(Number_of_Casualties) AS total_casualties
FROM road_accidents
GROUP BY Road_Type
ORDER BY total_casualties DESC
"""
road_agg = spark.sql(query)

road_agg_pd = road_agg.toPandas()

def visualize_road_type(df, column, color):
    plt.figure(figsize=(12, 7))
    plt.barh(df[column], df['total_casualties'], color=color)
    plt.xlabel('Total Casualties', fontsize=14)
    plt.ylabel(column, fontsize=14)
    plt.title('Road Types in UK with the highest record of road accidents (2005 - 2007)', fontsize=16)
    plt.gca().invert_yaxis()  # Đảo ngược trục y để hiển thị từ cao xuống thấp
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.show()

visualize_road_type(road_agg_pd, column='Road_Type', color='red')
```

Hình 2.13: Đoạn mã Thống kê những đường thường xảy ra tai nạn từ 2005 đến 2007



Hình 2.14: Kết quả Thống kê những đường thường xảy ra tai nạn từ 2005 đến 2007

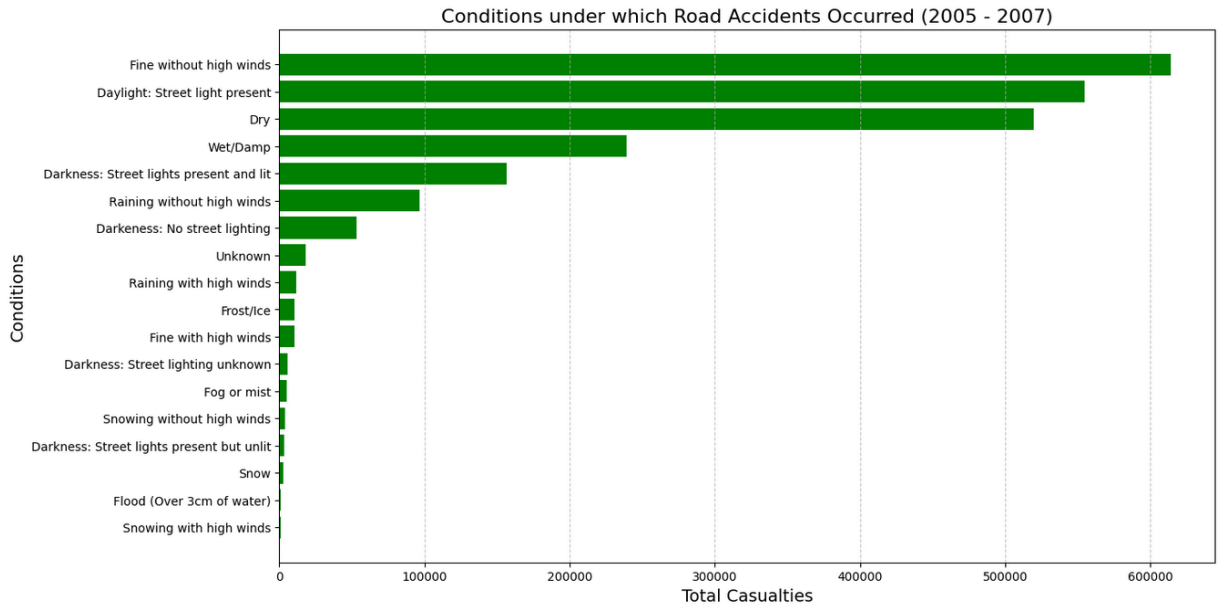
Từ kết quả ta nhận thấy: Đường 1 chiều (Single carriageway) có số lượng tai nạn cao nhất và đường nhánh (Slip road) có số lượng tai nạn thấp nhất

2.4.3. Thống kê những điều kiện ảnh hưởng tới vụ tai nạn từ năm 2005 đến 2007

```
query_roadsurface = """
SELECT
    Road_Surface_Conditions AS Conditions,
    SUM(Number_of_Casualties) AS total_casualties
FROM road_accidents
GROUP BY Road_Surface_Conditions
"""
roadsurface_agg = spark.sql(query_roadsurface)

query_weather = """
SELECT
    CASE
        WHEN Weather_Conditions = 'Other' THEN 'Unknown'
        ELSE Weather_Conditions
    END AS Conditions,
    SUM(Number_of_Casualties) AS total_casualties
FROM road_accidents
GROUP BY Weather_Conditions
"""
weather_agg = spark.sql(query_weather)

query_light = """
SELECT
    Light_Conditions AS Conditions,
    SUM(Number_of_Casualties) AS total_casualties
FROM road_accidents
GROUP BY Light_Conditions
"""
light_agg = spark.sql(query_light)
```

```
from pyspark.sql import DataFrame

def union_all(*dfs: DataFrame) -> DataFrame:
    return dfs[0].unionByName(dfs[1]) if len(dfs) == 2 else union_all(dfs[0].unionByName(dfs[1]), *dfs[2:])

condition_agg = union_all(roadsurface_agg, weather_agg, light_agg).orderBy("total_casualties", ascending=False)

condition_agg_pd = condition_agg.toPandas()


def visualize_conditions(df):
    plt.figure(figsize=(14, 8))
    plt.barh(df['Conditions'], df['total_casualties'], color='green')
    plt.xlabel('Total Casualties', fontsize=14)
    plt.ylabel('Conditions', fontsize=14)
    plt.title('Conditions under which Road Accidents Occurred (2005 - 2007)', fontsize=16)
    plt.gca().invert_yaxis()  # Đảo ngược trục y để hiển thị từ cao xuống thấp
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.show()

visualize_conditions(condition_agg_pd)
```

Hình 2.15: Đoạn mã Thống kê những điều kiện ảnh hưởng tới vụ tai nạn từ năm 2005 đến 2007



Hình 2.16:Kết quả thống kê những điều kiện ảnh hưởng tới vụ tai nạn từ năm 2005 đến 2007

Từ kết quả ta nhận thấy:

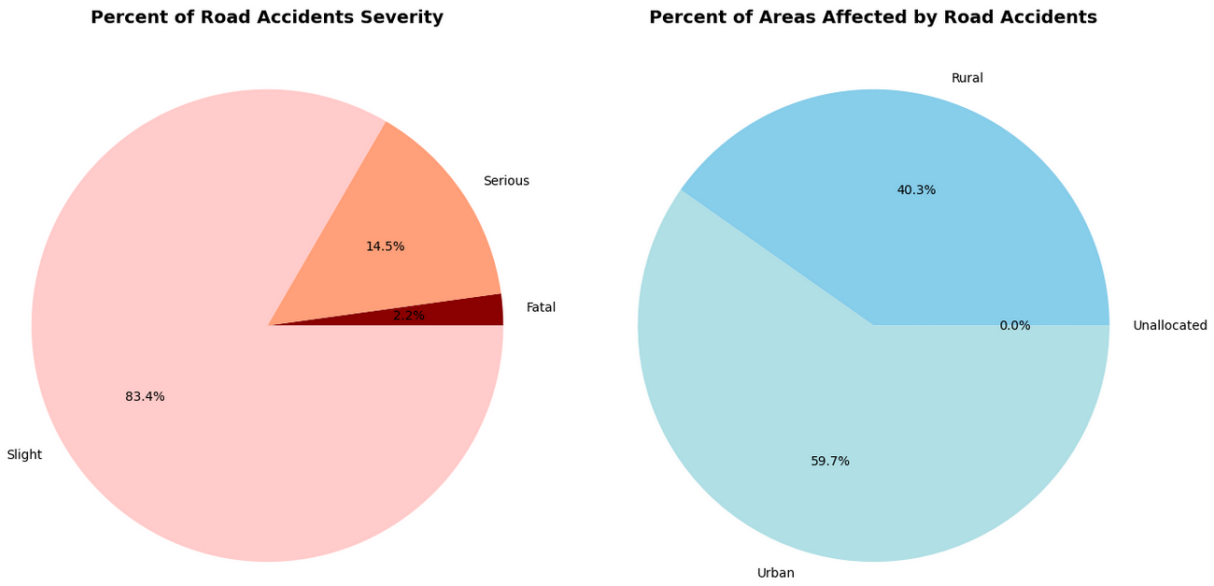
- Từ biểu đồ trên, hầu hết các vụ tai nạn giao thông xảy ra trong các điều kiện không ảnh hưởng đến đường, phương tiện hoặc tài xế, đơn giản vì phần lớn các vụ tai nạn được cho là xảy ra trong điều kiện thời tiết tốt, không có gió lớn, ban ngày với đèn đường hoạt động và trên mặt đường khô ráo.
- Ngược lại, hầu hết các vụ tai nạn giao thông không xảy ra khi có lũ lụt, trời tối hoặc có tuyết rơi kèm theo gió lớn.

2.4.4. Thống kê những khu vực nào thường xảy ra tai nạn và mức độ nghiêm trọng của các vụ tai nạn ở Anh từ năm 2005 đến 2007

```
severity_agg_sql = spark.sql("""
    SELECT
        CASE
            WHEN Accident_Severity = 1 THEN 'Fatal'
            WHEN Accident_Severity = 2 THEN 'Serious'
            WHEN Accident_Severity = 3 THEN 'Slight'
        END AS Accident_Severity,
        SUM(Number_of_Casualties) AS total_casualties
    FROM road_accidents
    GROUP BY Accident_Severity
    ORDER BY total_casualties
""")

area_agg_sql = spark.sql("""
    SELECT
        CASE
            WHEN Urban_or_Rural_Area = 1 THEN 'Urban'
            WHEN Urban_or_Rural_Area = 2 THEN 'Rural'
            WHEN Urban_or_Rural_Area = 3 THEN 'Unallocated'
        END AS Area_Type,
        SUM(Number_of_Casualties) AS total_casualties
    FROM road_accidents
    GROUP BY Urban_or_Rural_Area
    ORDER BY total_casualties
""")
```

```
severity_agg_pd = severity_agg_sql.toPandas()
area_agg_pd = area_agg_sql.toPandas()

labels1 = severity_agg_pd["Accident_Severity"]
values1 = severity_agg_pd["total_casualties"]

labels2 = area_agg_pd["Area_Type"]
values2 = area_agg_pd["total_casualties"]

fig, axes = plt.subplots(1, 2, figsize=(14, 7))

axes[0].pie(values1, labels=labels1, autopct='%1.1f%%', colors=['#8b0000', '#ffa07a', '#ffcccb'])
axes[0].set_title('Percent of Road Accidents Severity', fontsize=14, weight='bold')

axes[1].pie(values2, labels=labels2, autopct='%1.1f%%', colors=['#4682b4', '#87ceeb', '#b0e0e6'])
axes[1].set_title('Percent of Areas Affected by Road Accidents', fontsize=14, weight='bold')

plt.tight_layout()
plt.show()
```

Hình 2.17: Đoạn mã Thống kê những khu vực nào thường xảy ra tai nạn và mức độ nghiêm trọng của các vụ tai nạn ở Anh từ năm 2005 đến 2007



Hình 2.18: Kết quả Thống kê những khu vực nào thường xảy ra tai nạn và mức độ nghiêm trọng của các vụ tai nạn ở Anh từ năm 2005 đến 2007

Từ kết quả trên ta nhận thấy:

- Số vụ tai nạn giao thông xảy ra ở khu vực đô thị (Urban) cao hơn so với ở khu vực nông thôn (Rural).
- Phần lớn các vụ tai nạn giao thông chỉ gây thương tích nhẹ (Slight) cho nạn nhân, chỉ một tỷ lệ rất nhỏ các vụ tai nạn là gây tử vong (Fatal).

2.5. Tiền xử lý dữ liệu

2.5.1. Đếm số lượng giá trị Null ở từng thuộc tính trong dữ liệu

```
null_counts = []
for col in data.columns:
    null_count = data.select(F.sum(F.col(col).isNull().cast("int"))).first()[0]
    null_counts.append(Row(Column=col, Null_Count=null_count))

null_counts_df = spark.createDataFrame(null_counts)
null_counts_df.show(40)
```

Hình 2.19: Đoạn mã đếm số lượng giá trị Null ở từng thuộc tính trong dữ liệu

Column	Null_Count
Accident_Index	0
Location_Easting_...	101
Location_Northing...	101
Longitude	101
Latitude	101
Police_Force	0
Accident_Severity	0
Number_of_Vehicles	0
Number_of_Casualties	0
Date	0
Day_of_Week	0
Time	67
Local_Authority_(...	0
Local_Authority_(...	0
1st_Road_Class	0
1st_Road_Number	0
Road_Type	0
Speed_limit	0
Junction_Detail	570011
Junction_Control	236945
2nd_Road_Class	0
2nd_Road_Number	0
Pedestrian_Crossi...	17
Pedestrian_Crossi...	34
Light_Conditions	0
Weather_Conditions	20
Road_Surface_Cond...	662
Special_Condition...	11
Carriageway_Hazards	23
Urban_or_Rural_Area	0
Did_Police_Office...	2375
LSOA_of_Accident_...	47511
Year	0

Hình 2.20: Kết quả đếm số lượng giá trị Null ở từng thuộc tính trong dữ liệu

Ta nhận thấy có 3 cột có dữ liệu Null nhiều nhất là “Junction_Detail”, “Junction_Control”, “LSOA_of_Accident_Location”.

Ta tiến hành bỏ đi những cột trên và bỏ đi những hàng có chứa giá trị Null trong các cột của dữ liệu

```
data = data.drop("Junction_Detail", "Junction_Control", "LSOA_of_Accident_Location")
data = data.na.drop()
```

2.5.2. Chọn ra những cột có giá trị None lớn hơn 100000

```
none_columns = []

for col in data.columns:
    none_count = data.filter(F.col(col) == "None").count()
    if none_count >= 100000:
        none_columns.append(col)

print("Columns with more than 100,000 null values:", none columns)
```

Hình 2.21: Đoạn mã chọn ra những cột có giá trị None lớn hơn 100000

```
Columns with more than 100,000 null values: ['Special_Conditions_at_Site', 'Carriageway_Hazards']
```

Hình 2.22: Kết quả chọn ra những cột có giá trị None lớn hơn 100000

Ta tiến hành bỏ đi những cột có nhiều giá None trên

```
data = data.drop("Special_Conditions_at_Site", "Carriageway_Hazards")
```

2.5.3. Chuẩn hóa dữ liệu bằng phương pháp Min-Max Normalization

Chọn cột "Date" và "Number_of_Casualties" để huấn luyện mô hình và tổng hợp theo từng ngày

```
data = data.withColumn("Date", to_date(col("Date"), "dd/MM/yyyy"))

df_uk_lstm = data.select('Date', 'Number_of_Casualties')

df_uk_agg = df_uk_lstm.groupBy('Date').agg(
    F.sum('Number_of_Casualties').alias('sum_Casualties')
)
df_uk_agg_sorted = df_uk_agg.orderBy('Date')

df_uk_agg_sorted.show(10)
```

Hình 2.23: Đoạn mã chuẩn hóa dữ liệu bằng phương pháp Min-Max Normalization (1)

```
+----------+--------------+
|      Date|sum_Casualties|
+----------+--------------+
|2005-01-01|           441|
|2005-01-02|           460|
|2005-01-03|           408|
|2005-01-04|           584|
|2005-01-05|           665|
|2005-01-06|           699|
|2005-01-07|           643|
|2005-01-08|           589|
|2005-01-09|           505|
|2005-01-10|           684|
+----------+--------------+
only showing top 10 rows
```

Hình 2.24:Kết quả đoạn mã chuẩn hóa dữ liệu bằng phương pháp Min-Max Normalization

Chia tập dữ liệu thành 2 phần Train, Test với tỉ lệ 80,20

```
total_count = df_uk_agg_sorted.count()

train_count = int(total_count * 0.8)

train_data = df_uk_agg_sorted.limit(train_count)
test_data1 = df_uk_agg_sorted.subtract(train_data)
```

Hình 2.25: Chuẩn hóa dữ liệu bằng phương pháp Min-Max Normalization (2)

Tạo cột Vector Features và chuẩn hóa sum_Casualties về khoảng [0,1] bằng phương pháp Min-Max Normalization

```
vector_assembler = VectorAssembler(inputCols=["sum_Casualties"], outputCol="Features")
df_with_features = vector_assembler.transform(df_uk_agg_sorted)

min_max = df_with_features.select(
    F.min("sum_Casualties").alias("min_value"),
    F.max("sum_Casualties").alias("max_value")
).first()

min_value = min_max["min_value"]
max_value = min_max["max_value"]

df_normalized = df_with_features.withColumn(
    "value_normalized",
    (col("sum_Casualties") - min_value) / (max_value - min_value)
)

assembler = VectorAssembler(inputCols=["value_normalized"], outputCol="Scaled_Features")
df_vectorized = assembler.transform(df_normalized)

normalized_df = df_vectorized.drop("value_normalized")
normalized_df.show(3)
```

Hình 2.26: Đoạn mã chuẩn hóa dữ liệu bằng phương pháp Min-Max Normalization (3)

```
+----------+--------------+--------+--------------------+
|      Date|sum_Casualties|Features|     Scaled_Features|
+----------+--------------+--------+--------------------+
|2005-01-01|           441| [441.0]||[0.21145374449339...|
|2005-01-02|           460| [460.0]||[0.23237885462555...|
|2005-01-03|           408| [408.0]||[0.1751101321585903]|
+----------+--------------+--------+--------------------+
only showing top 3 rows
```

Hình 2.27:Kết quả đoạn mã chuẩn hóa dữ liệu bằng phương pháp Min-Max Normalization (1)

2.5.4. Thực hiện phân vùng dữ liệu theo năm và chia lag cho dữ liệu

```
normalized_df = normalized_df.withColumn("Year", year("Date"))

windowSpec = Window.partitionBy("Year").orderBy("Date")

def create_lagged_features(df, lag_count=50):
    for i in range(1, lag_count + 1):
        df = df.withColumn(f"lag_{i}", F.lag(df["Scaled_Features"], i).over(windowSpec))
    return df

lagged_df = create_lagged_features(normalized_df)
lagged_df = lagged_df.na.drop()  # Loại bỏ các giá trị NaN
```

Hình 2.28: Đoạn mã Thực hiện phân vùng dữ liệu theo năm và chia lag cho dữ liệu (1)

```
total_count = lagged_df.count()

train_count = int(total_count * 0.8)

lagged_train_data = lagged_df.limit(train_count)
lagged_test_data = lagged_df.subtract(lagged_train_data)
```

Hình 2.29 Đoạn mã Thực hiện phân vùng dữ liệu theo năm và chia lag cho dữ liệu (2)

CHƯƠNG 3. CƠ SỞ LÝ THUYẾT

3.1. Mô hình thuật toán Long short-term memory

3.1.1. LSTM là gì?

Thuật toán Long Short-Term Memory (LSTM) là một loại Mạng nơ-ron hồi quy (RNN) có khả năng học và ghi nhớ thông tin theo thời gian.

Nó có một ô nhớ cho phép lưu trữ và truy xuất thông tin theo thời gian. Mặt khác, RNN truyền thống có bộ nhớ hạn chế và chỉ có thể lưu trữ dữ liệu trong một khoảng thời gian giới hạn. Do đó, LSTM phù hợp hơn với các nhiệm vụ đòi hỏi khả năng nhớ lại và áp dụng kiến thức từ các đầu vào trước đó [3].

3.1.2. Xây dựng mô hình LSTM như thế nào?

Hàm Sigmoid dùng để biến đầu vào x thành giá trị trong khoảng [0, 1]

$$\sigma(x) = 1/\left(1 + e^{-x}\right)$$

Hình 3.1: Công thức tính hàm Sigmoid

$$\sigma\prime(x) = \sigma(x)(1 - \sigma(x))$$

Hình 3.2: Công thức tính đạo hàm Sigmoid

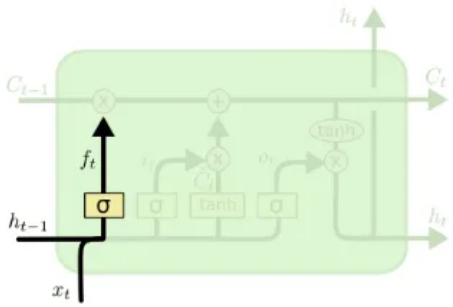
Hàm Tanh dùng để biến đầu vào x thành giá trị trong khoảng [-1, 1]

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Hình 3.3: Công thức tính hàm Tanh

$$tanh'\left(x\right) = 1 - tanh^{2}\left(x\right)$$

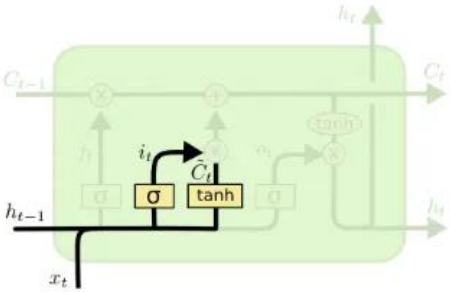
Hình 3.4: Công thức tính đạo hàm Tanh



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Hình 3.5: Tại sao chúng ta cần nó?

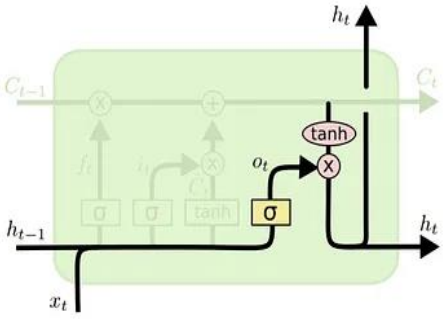
Cổng quên kiểm soát lượng thông tin từ bước thời gian trước được lưu giữ trong bước thời gian hiện tại.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Hình 3.6: Tại sao chúng ta cần nó?

Cổng đầu vào này cho biết lượng thông tin cần thêm để tạo ra trạng thái ô hiện tại sau khi lưu giữ thông tin trạng thái ô trước đó.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

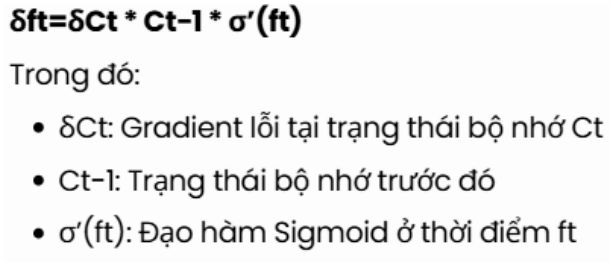
Hình 3.7: Tại sao chúng ta cần nó?

Cổng ra điều khiển lượng thông tin từ trạng thái ô được sử dụng để tạo ra đầu ra tại bước thời gian hiện tại, đầu ra trước đó, trạng thái ô trước đó.

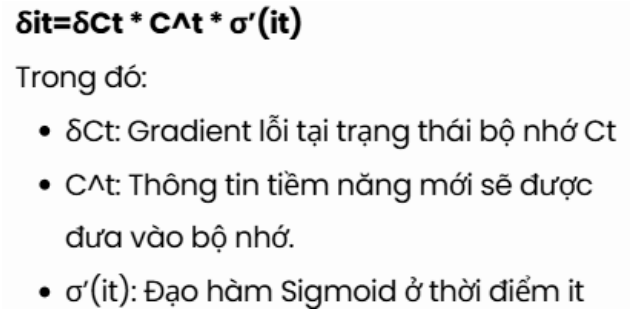
Cả ba cổng này đều là các thành phần cơ bản của LSTM, điều chỉnh luồng dữ liệu vào và ra khỏi ô nhớ. Và cuối cùng, trạng thái ô là một vectơ biểu diễn "bộ nhớ" của mạng LSTM; nó chứa thông tin từ cả bước thời gian trước đó và bước thời gian hiện tại [4].

Mỗi cổng sẽ quyết định phần nào của dữ liệu cũ hơn phải bị quên, phần nào của dữ liệu mới hơn phải được ghi nhớ và phần nào của bộ nhớ phải được cung cấp tương ứng.

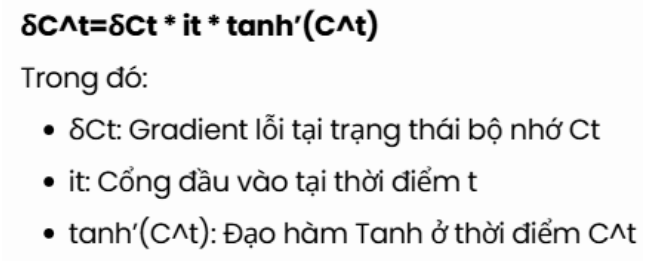
Quy trình lan truyền ngược là quy trình tính gradient của các tham số thông qua Backpropagation Through Time (BPTT). Điều này giúp tối ưu hóa trọng số của mô hình trong quá trình học.



Hình 3.8: Công thức tính gradient cho cổng forget



Hình 3.9: Công thức tính gradient cho cổng input



Hình 3.10: Công thức tính gradient cho cổng cell state

$$\delta ot = \delta ht * \tanh(Ct) * \sigma'(ot)$$

Trong đó:

- δht: Gradient lỗi tại trạng thái bộ nhớ ht
- Tanh(Ct): Hàm kích hoạt Tanh ở Ct
- σ'(ot): Đạo hàm Sigmoid ở thời điểm ot

Hình 3.11: Công thức tính gradient cho cổng output

$$\partial W / \partial Loss = \delta gate * [ht\_prev, xt]T$$

Trong đó:

- δgate: Gradient lỗi ở các cổng
- [ht_prev,xt]T: Ma trận chuyển vị kết hợp giữa ht_prev và xt

Hình 3.12: Công thức tính gradient cho Weight của các cổng

$$\partial b / \partial Loss = \delta gate$$

Trong đó:

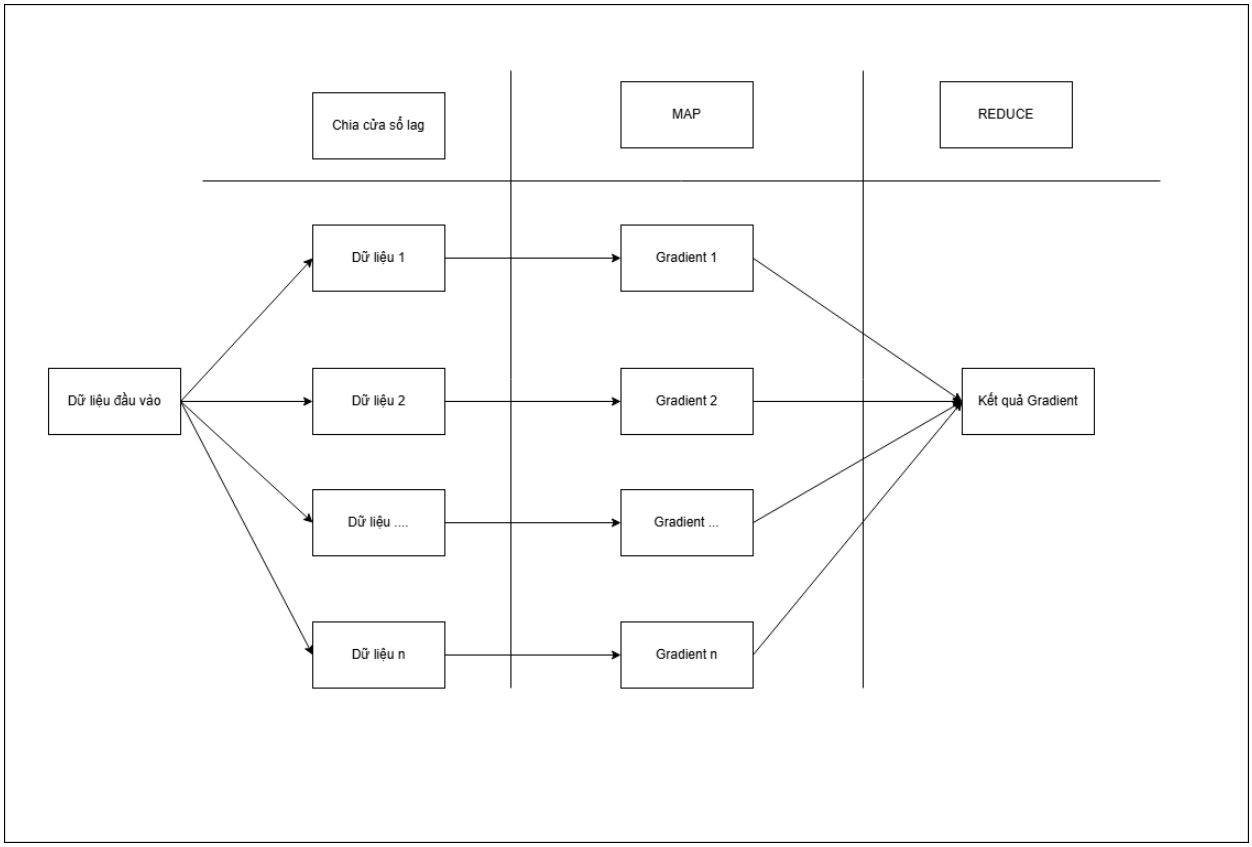
- δgate: Gradient lỗi ở các cổng

Hình 3.13: Công thức tính gradient bias của các cổng

3.1.3. Tại sao chúng ta cần sử dụng Mô hình LSTM?

Một trong những lợi ích chính của LSTM là khả năng xử lý sự phụ thuộc dài hạn. RNN truyền thống gặp khó khăn với thông tin được phân tách theo các khoảng thời gian dài, tuy nhiên LSTM có thể nhớ lại và sử dụng thông tin từ các đầu vào trước đó. Cho phép mô hình hiểu và ghi nhớ thông tin quan trọng từ quá khứ khi dự đoán tương lai. Điều này làm cho LSTM phù hợp với các ứng dụng như dự báo chuỗi thời gian, dịch máy, nhận dạng giọng nói và nhiều tác vụ khác liên quan đến dữ liệu chuỗi. Hơn nữa, LSTM có thể xử lý khối lượng dữ liệu lớn, khiến chúng trở nên lý tưởng cho các ứng dụng dữ liệu lớn

3.1.4. Sơ đồ song song hóa giải thuật



Hình 3.14: Sơ đồ song song hóa giải thuật

Khi chia cửa sổ lag: Dữ liệu ban đầu đọc từ csv sẽ được Spark phân tán tự động và được phân vùng theo cột Year khi chia cửa sổ lag trên các **worker nodes.**

Huấn luyện mô hình:

- Với giai đoạn Map: Áp dụng lên từng hàng trong dữ liệu. Quá trình diễn ra độc lập và song song cho từng hàng trong từng phân vùng
- Với giai đoạn Reduce: Tổng hợp các kết quả Gradient của từng phân vùng thành một kết quả duy nhất

CHƯƠNG 4. GIẢI THUẬT KHAI THÁC DỮ LIỆU

4.1. Áp dụng giải thuật trên Apache Spark

Bước 1.1: Khởi tạo lớp LSTMModel gồm các tham số: in_dim, hidden_dim, learn_rate và num_epochs

```
class LSTMModel(Params):
    in_dim = Param(Params._dummy(), "in_dim", "Input dimension of the LSTM cell")
    hidden_dim = Param(Params._dummy(), "hidden_dim", "Hidden dimension of the LSTM cell")
    learn_rate = Param(Params._dummy(), "learn_rate", "Learning rate")
    num_epochs = Param(Params._dummy(), "num_epochs", "Number of epochs")

    def __init__(self, in_dim=None, hidden_dim=None, out_dim=None, learn_rate=None, num_epochs=None):
        super(LSTMModel, self).__init__()
        self._setDefault(in_dim=50, hidden_dim=128, learn_rate=0.001, num_epochs=10)
        self.out_dim = out_dim
        self.lstm_weights = None
        self.lstm_weights_broadcast = None

        if in_dim is not None:
            self._set(in_dim=in_dim)
        if hidden_dim is not None:
            self._set(hidden_dim=hidden_dim)
        if learn_rate is not None:
            self._set(learn_rate=learn_rate)
        if num_epochs is not None:
            self._set(num_epochs=num_epochs)
```

Hình 4.1: Đoạn mã áp dụng giải thuật trên Apache Spark (1)

Bước 1.2: Khởi tạo các trọng số và bias của các cổng

```
def initialize_lstm_weights(self):
    hidden_units = self.getOrDefault(self.hidden_dim)
    input_units = self.getOrDefault(self.in_dim)
    output_units = self.out_dim

    def generate_weights(rows, cols):
        return np.random.normal(size=(rows, cols))

    def generate_biases(size):
        return np.zeros(shape=(size, 1))

    input_weights = generate_weights(hidden_units, input_units + hidden_units)
    input_bias = generate_biases(hidden_units)

    forget_weights = generate_weights(hidden_units, input_units + hidden_units)
    forget_bias = generate_biases(hidden_units)

    output_weights = generate_weights(hidden_units, input_units + hidden_units)
    output_bias = generate_biases(hidden_units)

    cell_weights = generate_weights(hidden_units, input_units + hidden_units)
    cell_bias = generate_biases(hidden_units)

    output_layer_weights = generate_weights(output_units, hidden_units)
    output_layer_bias = generate_biases(output_units)

    return input_weights, input_bias, forget_weights, forget_bias, output_weights, output_bias, cell_weights, cell_bias, output_layer_weigh
```

Hình 4.2: Đoạn mã áp dụng giải thuật trên Apache Spark (1.2)

Bước 1.3: Khởi tạo các hàm Sigmoid, Tanh, đạo hàm Sigmoid và đạo hàm Tanh

```
def activation_sigmoid(self, input_value):
    return 1/(1 + np.exp(-input_value))

def activation_sigmoid_derivative(self, sigmoid_output):
    return np.multiply(sigmoid_output, np.subtract(1, sigmoid_output))

def activation_tanh(self, input_value):
    return np.tanh(input_value)

def activation_tanh_derivative(self, tanh_output):
    return np.subtract(1, np.square(tanh_output))
```

Hình 4.3: Đoạn mã áp dụng giải thuật trên Apache Spark (1.3)

Bước 1.4: Thực hiện lan truyền xuôi trong mô hình LSTM

```
def lstm_forward(self, xt, ht_prev, ct_prev, input_weights, input_bias, forget_weights, forget_bias, output_weights, output_bias, cell_weigh
    concat = np.vstack((ht_prev, xt))

    it = self.activation_sigmoid(np.dot(input_weights, concat) + input_bias)
    ft = self.activation_sigmoid(np.dot(forget_weights, concat) + forget_bias)
    ot = self.activation_sigmoid(np.dot(output_weights, concat) + output_bias)

    ct_tilde = np.tanh(np.dot(cell_weights, concat) + cell_bias)
    ct = ft * ct_prev + it * ct_tilde
    ht = ot * np.tanh(ct)

    return ht, ct, it, ft, ot, ct_tilde
```

Hình 4.4: Đoạn mã áp dụng giải thuật trên Apache Spark (1.4)

Bước 1.5: Thực hiện quá trình lan truyền ngược trong LSTM

```
def lstm_backward(self, dht, dct, xt, ht, ct, ht_prev, ct_prev, it, ft, ot, ct_tilde, input_weights, input_bias, forget_weights, forget_bi
    concat = np.vstack((ht_prev, xt))

    dot = dht * np.tanh(ct) * self.activation_sigmoid_derivative(ot)
    d_output_weights = np.dot(dot, concat.T)
    d_output_bias = dot

    dct += dht * ot * self.activation_tanh_derivative(np.tanh(ct))

    dft = dct * ct_prev * self.activation_sigmoid_derivative(ft)
    d_forget_weights = np.dot(dft, concat.T)
    d_forget_bias = dft

    dit = dct * ct_tilde * self.activation_sigmoid_derivative(it)
    d_input_weights = np.dot(dit, concat.T)
    d_input_bias = dit

    dct_tilde = dct * it * self.activation_tanh_derivative(ct_tilde)
    d_cell_weights = np.dot(dct_tilde, concat.T)
    d_cell_bias = dct_tilde
    return d_input_weights, d_input_bias, d_forget_weights, d_forget_bias, d_output_weights, d_output_bias, d_cell_weights, d_cell_bias
```

Hình 4.5: Đoạn mã áp dụng giải thuật trên Apache Spark (1.5)

Bước 1.6: Thực hiện quá trình map từng dòng của dữ liệu để tìm gradient

```
def map_gradient(self, row, lstm_weights):
    xt = np.array(row[4:], dtype=np.float32).reshape(self.getOrDefault(self.in_dim), 1)
    yt = np.array([row[1]], dtype=np.float32).reshape(self.out_dim, 1)

    input_weights, input_bias, forget_weights, forget_bias, output_weights, output_bias, cell_weights, cell_bias, output_layer_weights, outp

    ht_prev = np.zeros((self.getOrDefault(self.hidden_dim), 1))
    ct_prev = np.zeros((self.getOrDefault(self.hidden_dim), 1))

    ht, ct, it, ft, ot, ct_tilde = self.lstm_forward(xt, ht_prev, ct_prev, input_weights, input_bias, forget_weights, forget_bias, output_we
    output = np.dot(output_layer_weights, ht) + output_layer_bias

    dout = output - yt
    d_output_layer_weights = np.dot(dout, ht.T)
    d_output_layer_bias = dout
    dht = np.dot(output_layer_weights.T, dout)
    dct = np.zeros_like(ct)

    d_input_weights, d_input_bias, d_forget_weights, d_forget_bias, d_output_weights, d_output_bias, d_cell_weights, d_cell_bias = self.lstm

    return [(d_input_weights, d_input_bias), (d_forget_weights, d_forget_bias), (d_output_weights, d_output_bias), (d_cell_weights, d_cell_b
```

Hình 4.6: Đoạn mã áp dụng giải thuật trên Apache Spark (1.6)

Bước 1.7: Thực hiện cập nhật lại các trọng số và bias dựa trên gradient theo cơ chế gradient descent

```
def _update_weights(self, gradients):
    input_weights, input_bias, forget_weights, forget_bias, output_weights, output_bias, cell_weights,
    cell_bias, output_layer_weights, output_layer_bias = self.lstm_weights

    d_input_weights, d_input_bias = gradients[0]
    d_forget_weights, d_forget_bias = gradients[1]
    d_output_weights, d_output_bias = gradients[2]
    d_cell_weights, d_cell_bias = gradients[3]
    d_output_layer_weights, d_output_layer_bias = gradients[4]

    input_weights -= self.getOrDefault(self.learn_rate) * d_input_weights
    input_bias -= self.getOrDefault(self.learn_rate) * d_input_bias
    forget_weights -= self.getOrDefault(self.learn_rate) * d_forget_weights
    forget_bias -= self.getOrDefault(self.learn_rate) * d_forget_bias
    output_weights -= self.getOrDefault(self.learn_rate) * d_output_weights
    output_bias -= self.getOrDefault(self.learn_rate) * d_output_bias
    cell_weights -= self.getOrDefault(self.learn_rate) * d_cell_weights
    cell_bias -= self.getOrDefault(self.learn_rate) * d_cell_bias
    output_layer_weights -= self.getOrDefault(self.learn_rate) * d_output_layer_weights
    output_layer_bias -= self.getOrDefault(self.learn_rate) * d_output_layer_bias

    self.lstm_weights = (input_weights, input_bias, forget_weights, forget_bias, output_weights, output_bias, cell_weights,
                         cell_bias, output_layer_weights, output_layer_bias)
```

Hình 4.7: Đoạn mã áp dụng giải thuật trên Apache Spark (1.7)

Bước 1.8: Tổng hợp lại các gradient của mỗi hàng bằng phép cộng

```
def reduce_gradient(self, g1, g2):
    return [(a + c, b + d) for (a, b), (c, d) in zip(g1, g2)]
```

Hình 4.8: Đoạn mã áp dụng giải thuật trên Apache Spark (1.8)

Bước 1.9: Thực hiện dự đoán tập lag_test và 90 ngày tiếp theo sau khi đã huấn luyện mô hình

```
def predict(self, row, lstm_weights):
    xt = np.array(row[1][1:], dtype=np.float32).reshape(self.getOrDefault(self.in_dim), 1)

    ht_prev = np.zeros((self.getOrDefault(self.hidden_dim), 1))
    ct_prev = np.zeros((self.getOrDefault(self.hidden_dim), 1))

    input_weights, input_bias, forget_weights, forget_bias, output_weights, output_bias, cell_weights, cell_bias, output_layer_weights,

    ht, ct, _, _, _, _ = self.lstm_forward(xt, ht_prev, ct_prev, input_weights, input_bias, forget_weights, forget_bias, output_weights,
    output = np.dot(output_layer_weights, ht) + output_layer_bias

    return (row[0], float(output[0][0]), row[2])

def predict_next90(self, xt, lstm_weights):
    ht_prev = np.zeros((self.getOrDefault(self.hidden_dim), 1))
    ct_prev = np.zeros((self.getOrDefault(self.hidden_dim), 1))

    input_weights, input_bias, forget_weights, forget_bias, output_weights, output_bias, cell_weights, cell_bias, output_layer_weights,

    ht, ct, _, _, _, _ = self.lstm_forward(xt, ht_prev, ct_prev, input_weights, input_bias, forget_weights, forget_bias, output_weights,
    output = np.dot(output_layer_weights, ht) + output_layer_bias

    return (float(output[0][0]))
```

Hình 4.9: Đoạn mã áp dụng giải thuật trên Apache Spark (1.9)

Bước 1.10: Thực hiện huấn luyện mô hình với mỗi lần lặp epoch và cập nhật lại trọng số dựa vào gradient theo mỗi lần lặp

```
def train(self, lagged_train_data, lstm_weights):
    self.lstm_weights = lstm_weights

    for epoch in range(self.getOrDefault(self.num_epochs)):
        gradients = lagged_train_data.rdd \
            .map(lambda row: self.map_gradient(row, self.lstm_weights)) \
            .reduce(self.reduce_gradient)

        self._update_weights(gradients)

        print(f"Epoch {epoch+1}/{self.getOrDefault(self.num_epochs)} completed")
```

Hình 4.10: Đoạn mã áp dụng giải thuật trên Apache Spark (1.10)

Bước 2: Hàm khởi tạo mô hình LSTM

```
def create_lstm_model1(in_dim, hidden_dim, lr, epoch):
    return LSTMModel(
        in_dim=in_dim,
        hidden_dim=hidden_dim,
        learn_rate=lr,
        num_epochs=epoch,
        out_dim=1
    )
```

Hình 4.11: Đoạn mã áp dụng giải thuật trên Apache Spark (2)

Bước 3: Thực hiện dự đoán mô hình trên tập lag_test bằng cách map qua từng dòng dữ liệu và dự đoán trả về kết quả DataFrame

```
def evaluate_model(model, test_data, lstm_weights):
    test_data = test_data.select(
        col("Date"),
        array(col("Scaled_Features"), *[col(f"lag_{i}") for i in range(1, 51)]).alias("Features"),
        col("sum_Casualties").alias("Label")
    )
    return test_data.rdd \
        .map(lambda row: model.predict(row, lstm_weights)) \
        .toDF(["Date", "Prediction", "Actual"])
```

Hình 4.12: Đoạn mã áp dụng giải thuật trên Apache Spark (3)

Bước 4: Tính toán các độ đo RMSE, R2 dựa trên dự đoán tập lag_test

```
def calculate_metrics(predictions):
    rmse = predictions.selectExpr("sqrt(mean(power(Prediction - Actual, 2))) as rmse").first()[0]
    mean_actual = predictions.select(avg("Actual")).first()[0]
    r2 = predictions.selectExpr(f"1 - sum(power(Prediction - Actual, 2)) / sum(power(Actual - {mean_actual}, 2)) as r2").first()[0]
    return rmse, r2

def print_evaluation_metrics(rmse, r2):
    print(f"The RMSE is: {rmse}")
    print(f"The R2 is : {r2}")
```

Hình 4.13: Đoạn mã áp dụng giải thuật trên Apache Spark (4)

Bước 5: Hàm train_and_evaluate1 bao gồm tất cả quy trình chạy thuật toán gồm khởi tạo mô hình, khởi tạo trọng số, huấn luyện mô hình, dự đoán tập lag_test và dự đoán 90 ngày tiếp theo

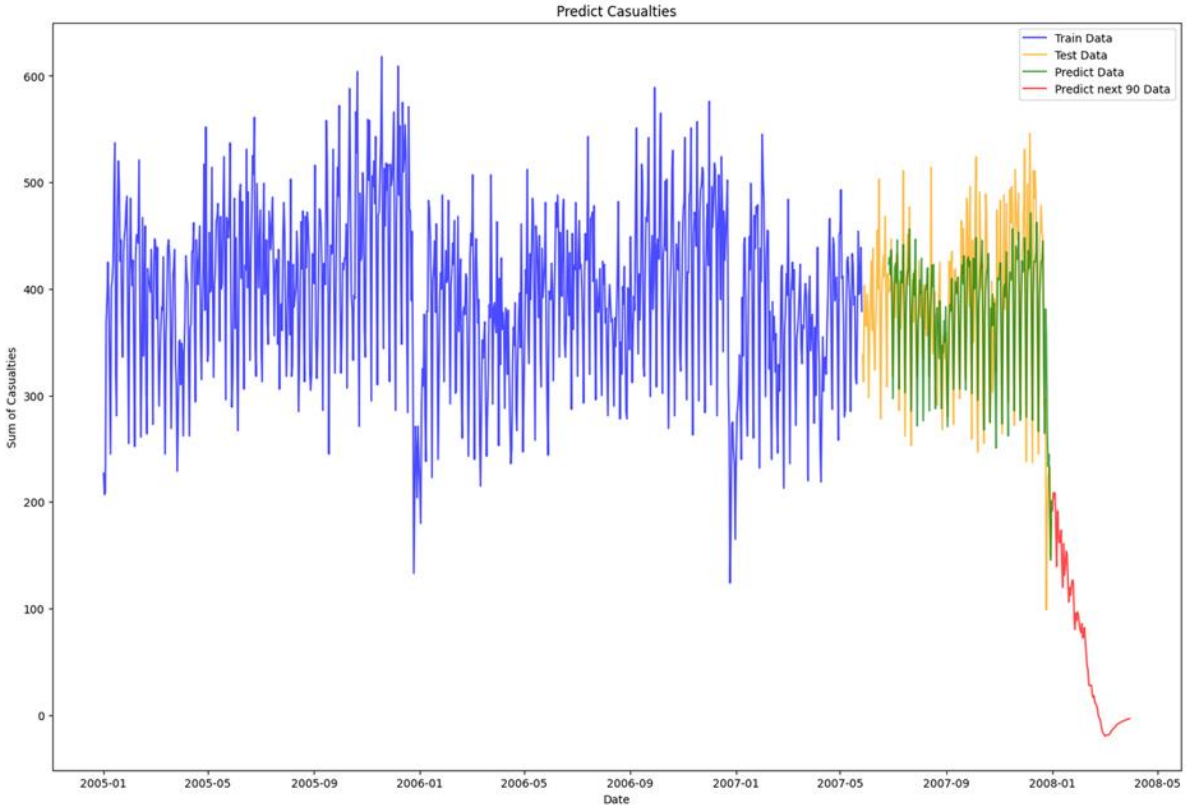
```python
def train_and_evaluate1(lagged_train_data, lagged_test_data, in_dim, hidden_dim, lr, epoch, n_step):
    lag_train = lagged_train_data.drop("Year")
    lag_test = lagged_test_data.drop("Year")

    lstm_model = create_lstm_model1(in_dim, hidden_dim, lr, epoch)
    lstm_weights = lstm_model.initialize_lstm_weights()
    lstm_model.train(lag_train, lstm_weights)

    test_predictions = evaluate_model(lstm_model, lag_test, lstm_weights)

    last_row = lagged_test_data.orderBy(F.col("Date").desc()).limit(1)

    last_features1 = last_row.select([*["Scaled_Features"] + [f"lag_{i}" for i in range(1, 50)]]).first()

    last_date = last_row.select("Date").first()[0]

    current_date = datetime.strptime(str(last_date), "%Y-%m-%d")

    schema = StructType([
        StructField("Date", StringType(), True),
        StructField("Prediction", FloatType(), True)
    ])

    predictions_df = spark.createDataFrame([], schema)

    xt = np.array([last_features1[col] for col in ["Scaled_Features"] + [f"lag_{i}" for i in range(1, 50)]], dtype=np.float32).reshape(50, 1)

    for step in range(n_step):
        new_prediction = lstm_model.predict_next90(xt,lstm_weights)

        new_prediction_scaled = (new_prediction - min_value) / (max_value - min_value)

        xt = np.insert(xt, 0, new_prediction_scaled, axis=0)
        xt = np.delete(xt, -1, axis=0)

        current_date += timedelta(days=1)
        new_date = current_date.strftime("%Y-%m-%d")

        print(f"Predict {new_date}: {new_prediction}")

        new_row = spark.createDataFrame([(new_date, new_prediction)], schema)
        predictions_df = predictions_df.union(new_row)
    return test_predictions, predictions_df
```

Hình 4.14: Đoạn mã áp dụng giải thuật trên Apache Spark (5)

Bước 6: Chuyển dữ liệu sang Pandas và thực hiện trực quan hóa dữ liệu

```python
test_predictions_pd = test_prediction.select("Date", "Prediction").toPandas()
next90_predict_pd = prediction_next90.select("Date", "Prediction").toPandas()
train_pandas = train_data.select("Date", "sum_Casualties").toPandas()
test_pandas = test_data1.select("Date", "sum_Casualties").toPandas()

train_pandas['Date'] = pd.to_datetime(train_pandas['Date'])
test_pandas['Date'] = pd.to_datetime(test_pandas['Date'])
test_predictions_pd['Date'] = pd.to_datetime(test_predictions_pd['Date'])
next90_predict_pd['Date'] = pd.to_datetime(next90_predict_pd['Date'])

plt.figure(figsize=(18, 12))
plt.plot(train_pandas['Date'], train_pandas['sum_Casualties'], label='Train Data', color='blue', alpha=0.7)
plt.plot(test_pandas['Date'], test_pandas['sum_Casualties'], label='Test Data', color='orange', alpha=0.7)
plt.plot(test_predictions_pd['Date'], test_predictions_pd['Prediction'], label='Predict Data', color='green', alpha=0.7)
plt.plot(next90_predict_pd['Date'], next90_predict_pd['Prediction'], label='Predict next 90 Data', color='red', alpha=0.7)

plt.xlabel("Date")
plt.ylabel("Sum of Casualties")
plt.title("Predict Casualties")
plt.legend()

plt.show()
```

Hình 4.15: Đoạn mã áp dụng giải thuật trên Apache Spark (6)



Hình 4.16: Kết quả đoạn mã áp dụng giải thuật trên Apache Spark (6)

CHƯƠNG 5. KẾT QUẢ ĐẠT ĐƯỢC

5.1. Phát biểu kết quả

Thực hiện huấn luyện mô hình và dự đoán 90 ngày tiếp theo

```
in_dim = 50
hidden_dim = 256
lr = 0.00001
epoch = 2000
n_step = 90

test_prediction, prediction_next90 = train_and_evaluate1(lagged_train_data, lagged_test_data, in_dim, hidden_dim, lr, epoch, n_step)
```

5.2. So sánh và đánh giá

5.2.1. RMSE

RMSE cung cấp một cái nhìn tổng thể về sự chính xác của mô hình, phản ánh mức độ mà dự đoán của mô hình chệch so với thực tế.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n} \varepsilon_t^2}{n}} = \sqrt{\frac{\sum_{t=1}^{n} \left(Y_t - \hat{Y}_t\right)^2}{n}}$$

Hình 5.1: Công thức tính RMSE

Trong đó:

- yi là giá trị thực tế.
- y^i là giá trị dự đoán.
- n là số lượng mẫu

5.2.2. R2

R^2 giúp nhận diện nhanh chóng liệu mô hình có hoạt động tốt không. Một giá trị R^2 gần 1 cho thấy mô hình có khả năng dự báo chính xác, trong khi giá trị R^2 gần 0 có thể cho thấy mô hình không giải thích được nhiều sự thay đổi trong dữ liệu.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Hình 5.2: Công thức tính R2

Trong đó:

- yi là giá trị thực tế.
- y^i là giá trị dự đoán.
- Y-i là giá trị trung bình của các giá trị thực tế.

5.2.3. Đánh giá

```
test_rmse, test_r2 = calculate_metrics(test_prediction)

print("Testing metrics:")
print evaluation metrics(test rmse, test r2)
```

Hình 5.3: Đoạn mã đánh giá thuật toán

```
Testing metrics:
The RMSE is: 91.76363061761475
The R2 is : 0.3711740588660011
```

Hình 5.4:Kết quả đánh giá thuật toán

Để đánh giá hiệu quả của mô hình, chúng ta sử dụng các chỉ số đánh giá như RMSE (Root Mean Squared Error) và R^2 (Coefficient of Determination). Dưới đây là kết quả của các chỉ số sau khi thực hiện dự báo trên tập kiểm tra:

- **RMSE (Root Mean Squared Error)**: 50.158
- **R^2 (Coefficient of Determination)**: 0.602

Giá trị RMSE là 91.76, cho thấy mức độ sai lệch trung bình giữa giá trị dự đoán và thực tế.

R^2 = 0.371 cho thấy mô hình không quá mạnh trong việc dự đoán, vì tỷ lệ R^2 khá thấp

RMSE cung cấp cái nhìn tổng thể về sự chính xác của mô hình, cho thấy mức độ chệch lệch giữa các dự đoán và giá trị thực tế. Giá trị RMSE càng thấp, mô hình càng chính xác. Trong khi đó, chỉ số R^2 giúp đánh giá khả năng giải thích sự biến động của dữ liệu. Một giá trị R^2 gần 1 cho thấy mô hình có khả năng dự báo chính xác, trong khi giá trị R^2 gần 0 có thể chỉ ra rằng mô hình chưa giải thích tốt sự biến đổi trong dữ liệu.

5.3. Ưu điểm

Mô hình LSTM (Long Short-Term Memory) được ứng dụng trong nghiên cứu này đã cho thấy những ưu điểm rõ rệt trong việc dự báo số lượng thương vong ở Anh từ năm 2005 đến 2007. Cụ thể, LSTM có khả năng học và duy trì thông tin qua nhiều thời kỳ trong dữ liệu, giúp mô hình hiểu rõ mối quan hệ giữa các yếu tố qua thời gian. Điều này đặc biệt quan trọng khi dự báo những sự kiện có tính chuỗi thời gian, như số lượng thương vong, có sự phụ thuộc vào các yếu tố trong quá khứ. Mô hình LSTM có thể xử lý tốt các vấn đề như sự thay đổi đột ngột trong dữ liệu, sự xuất hiện của các đỉnh dữ liệu không thể dự đoán một cách dễ dàng với các mô hình thông thường.

- **Cấu trúc đơn giản và dễ hiểu:** Mô hình LSTM có cấu trúc rõ ràng với các thành phần cơ bản như cell state, input gate, output gate và forget gate. Việc này giúp các nhà nghiên cứu và lập trình viên có thể dễ dàng hiểu và triển khai mô hình vào các bài toán chuỗi thời gian mà không cần phải có kiến thức quá chuyên sâu về học sâu.
- **Khả năng học mối quan hệ dài hạn:** LSTM có khả năng lưu trữ và duy trì thông tin trong thời gian dài, giúp mô hình hiểu rõ các mối quan hệ phức tạp trong dữ liệu chuỗi thời gian. Điều này giúp LSTM vượt trội trong việc giải quyết các vấn đề phụ thuộc lâu dài mà các mô hình khác khó có thể học được.
- **Giảm hiện tượng vanishing gradient:** Một trong những điểm mạnh của LSTM là khả năng ngăn ngừa hiện tượng vanishing gradient, một vấn đề phổ biến trong các mô hình học sâu khác. Điều này giúp LSTM có thể học từ dữ liệu dài hạn mà không bị mất mát thông tin qua nhiều bước huấn luyện.
- **Có thể mở rộng với các tập dữ liệu lớn:** LSTM có thể dễ dàng mở rộng và xử lý các tập dữ liệu lớn nhờ vào khả năng học hiệu quả từ các chuỗi thời gian dài mà không gặp phải vấn đề về quá tải bộ nhớ hoặc hiệu suất. Việc áp dụng các kỹ thuật phân tán hoặc tối ưu hóa phần cứng như GPU có thể giúp tăng tốc quá trình huấn luyện.
- **Dễ dàng tích hợp với các mô hình khác:** LSTM có thể được kết hợp với các mô hình học máy khác như mạng nơ-ron tích chập (CNN) hoặc mạng nơ-ron đối kháng (GAN) để giải quyết các bài toán phức tạp hơn, đặc biệt là khi cần kết hợp thông tin chuỗi thời gian với các loại dữ liệu khác.

- **Có nhiều phiên bản và cải tiến:** Các phiên bản mở rộng của LSTM như GRU (Gated Recurrent Unit) hay các biến thể của LSTM như BiLSTM (Bidirectional LSTM) được phát triển để cải thiện hiệu suất và khả năng dự báo trong các bài toán đa dạng hơn. Các phiên bản này giúp LSTM trở nên linh hoạt và mạnh mẽ hơn trong các tình huống thực tế.

5.4. Hạn chế

Mặc dù LSTM có nhiều ưu điểm, nhưng việc sử dụng mô hình này cũng gặp phải một số hạn chế. Thứ nhất, LSTM yêu cầu một lượng dữ liệu lớn và chất lượng cao để huấn luyện mô hình hiệu quả. Nếu dữ liệu đầu vào thiếu hụt hoặc không đầy đủ, mô hình có thể không hoạt động tốt và dẫn đến kết quả không chính xác. Thứ hai, việc huấn luyện mô hình LSTM tốn khá nhiều tài nguyên tính toán và thời gian. Điều này có thể trở thành một yếu tố hạn chế khi triển khai mô hình trong môi trường với yêu cầu thời gian thực hoặc khi tài nguyên tính toán hạn chế.

Một hạn chế nữa là LSTM có thể gặp khó khăn trong việc dự báo các hiện tượng không có sự liên kết chặt chẽ với các yếu tố lịch sử trong dữ liệu, hoặc khi có sự biến động mạnh mẽ mà mô hình không thể nắm bắt được ngay từ các chuỗi thời gian trước đó.

5.5. Hướng phát triển

Mặc dù LSTM (Long Short-Term Memory) đã chứng tỏ sự hiệu quả trong việc xử lý các bài toán chuỗi thời gian dài hạn, nhưng cũng không thiếu những thách thức cần phải giải quyết. Các nghiên cứu và phát triển liên tục đang hướng đến việc cải thiện khả năng ứng dụng của LSTM trong các bài toán phức tạp hơn, đặc biệt là với các tập dữ liệu lớn hoặc có tính biến động mạnh. Dưới đây là một số hướng phát triển của LSTM:

- **Xử lý dữ liệu thưa thớt:** LSTM có thể gặp khó khăn khi đối mặt với các tập dữ liệu thưa thớt, nơi thông tin thiếu hụt có thể gây khó khăn trong việc học. Để cải thiện, các nghiên cứu hiện nay đang tập trung vào việc kết hợp LSTM với các kỹ thuật học máy khác như GAN (Generative Adversarial Networks) hoặc các phương pháp sinh mẫu để tăng cường độ chính xác khi làm việc với dữ liệu thiếu thông tin.

- **Kỹ thuật giảm độ phức tạp:** Mặc dù LSTM đã giúp giải quyết vấn đề vanishing gradient, nhưng mô hình này vẫn có thể gặp phải vấn đề khi số lượng dữ liệu quá lớn hoặc các chuỗi dài quá phức tạp. Do đó, các phiên bản cải tiến của LSTM như GRU (Gated Recurrent Unit) hoặc các phương pháp tăng cường khác đang được nghiên cứu để giảm độ phức tạp tính toán mà vẫn giữ được khả năng học tốt từ dữ liệu dài hạn.
- **Kết hợp với các mô hình khác:** LSTM có thể được kết hợp với các mô hình học máy khác để xử lý các bài toán phức tạp hơn. Ví dụ, kết hợp LSTM với mô hình CNN (Convolutional Neural Networks) để xử lý các chuỗi dữ liệu không chỉ là văn bản mà còn là dữ liệu hình ảnh hoặc video, giúp khai thác được thông tin từ nhiều nguồn dữ liệu khác nhau một cách hiệu quả.
- **Phát triển các phiên bản LSTM nâng cao:** Các phiên bản LSTM như BiLSTM (Bidirectional LSTM) hoặc Attention-based LSTM đang được nghiên cứu để cải thiện hiệu suất và khả năng học của mô hình. BiLSTM có thể học từ dữ liệu ở cả hai chiều (trái sang phải và phải sang trái), trong khi các mô hình Attention giúp mô hình tập trung vào các phần quan trọng trong chuỗi, từ đó tăng cường khả năng dự đoán và xử lý thông tin.
- **Ứng dụng trong nhiều lĩnh vực khác nhau:** LSTM không chỉ dừng lại ở các bài toán chuỗi thời gian truyền thống mà còn được ứng dụng rộng rãi trong các lĩnh vực khác như nhận dạng giọng nói, phân tích cảm xúc trong văn bản, dự báo tài chính, y tế, và xử lý ngữ nghĩa tự nhiên. Các nghiên cứu tiếp tục mở rộng khả năng của LSTM trong các ứng dụng này bằng cách kết hợp các kỹ thuật tiên tiến và tối ưu hóa mô hình.

Như vậy, LSTM không ngừng phát triển với các cải tiến về hiệu suất, khả năng ứng dụng và kết hợp với các công nghệ khác để giải quyết những bài toán khó khăn hơn, từ đó mở rộng phạm vi ứng dụng trong nhiều lĩnh vực khác nhau.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] World Health Organization, "Global status report on road safety 2018," 17 6 2018. [Online]. Available: https://www.who.int/publications/i/item/9789241565684.

[2] Department for Transport, "Reported road casualties Great Britain, annual report: 2020," 30 9 2021. [Online]. Available: https://www.gov.uk/government/statistics/reported-road-casualties-great-britain-annual-report-2020.

[3] S. Saxena, "What is LSTM? Introduction to Long Short-Term Memory," 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.

[4] Sumanth, "LSTM : Why do we need it?," 5 6 2021. [Online]. Available: sumanthai1993.medium.com/lstm-why-do-we-need-it-8503d277c7a3.

[5] I. N. Oporto, "1.5 Million UK Traffic Accidents EDA," 2022. [Online]. Available: https://www.kaggle.com/code/isidronavarrooporto/1-5-million-uk-traffic-accidents-eda/input?fbclid=IwY2xjawHKESNleHRuA2FlbQIxMAABHeZbV6i_L0TLac5mrL6oStAeXLK9jV51KbQOPhcUPVZpXaB2huwOoMvBJw_aem_JPj37UvurM_O-RWvZ-GAwg.