TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
ĐẠI HỌC QUỐC GIA HÀ NỘI

BÁO CÁO BÀI TẬP LỚN

Ứng dụng Naive Bayes và SVM trong phân tích cảm xúc văn bản lớn trên nền tảng Hadoop và Spark

---

Sinh viên :
23020384 Nguyễn Đình Khải
23020374 Đoàn Quang Huy

Giảng viên :
TS. Trần Hồng Việt

Tháng 5, 2025

Mục lục

Danh sách hình vẽ

Danh sách bảng

Bảng phân chia công việc

Công việc	**Người thực hiện**
1. Xây dựng cấu trúc dự án	Nguyễn Đình Khải
2. Viết code phần Hadoop (Naive Bayes và SVM)	Nguyễn Đình Khải
3. Viết code phần Spark (Naive Bayes và SVM)	Đoàn Quang Huy
4. Thiết kế bố cục và hình thức báo cáo	Nguyễn Đình Khải
5. Viết báo cáo phần Giới thiệu chung và Cơ sở lý thuyết	Nguyễn Đình Khải
6. Viết báo cáo phần Tổng quan về Big Data	Đoàn Quang Huy
7. Viết báo cáo phần Hadoop (Naive Bayes và SVM)	Nguyễn Đình Khải
8. Viết báo cáo phần Spark (Naive Bayes và SVM)	Đoàn Quang Huy
9. Viết báo cáo phần kết quả và thảo luận	Nguyễn Đình Khải
10. Viết báo cáo phần kết luận và hướng phát triển	Đoàn Quang Huy
11. Thiết kế cấu trúc slide thuyết trình	Đoàn Quang Huy
12. Làm slide phần Hadoop (Naive Bayes và SVM)	Nguyễn Đình Khải
13. Làm slide phần Spark (Naive Bayes và SVM)	Đoàn Quang Huy

Tóm tắt

Trong bối cảnh bùng nổ dữ liệu văn bản từ các nền tảng mạng xã hội, diễn đàn và hệ thống phản hồi khách hàng, việc phân tích cảm xúc văn bản trở thành một công cụ quan trọng giúp doanh nghiệp và tổ chức hiểu rõ hơn về tâm lý và hành vi người dùng. Tuy nhiên, xử lý và phân tích khối lượng lớn dữ liệu văn bản đòi hỏi các giải pháp tính toán hiệu quả và có khả năng mở rộng.

Dự án này tập trung vào việc ứng dụng hai thuật toán học máy phổ biến là **Naive Bayes (NB)** và **Support Vector Machine (SVM)** trong phân tích cảm xúc văn bản lớn, triển khai trên hai nền tảng xử lý phân tán mạnh mẽ là **Hadoop MapReduce** và **Apache Spark**. Mục tiêu là xây dựng một hệ thống có khả năng xử lý hiệu quả dữ liệu văn bản lớn, đồng thời so sánh hiệu suất và độ chính xác giữa các mô hình và nền tảng.

Quá trình thực hiện bao gồm việc thu thập và tiền xử lý dữ liệu văn bản, trích xuất đặc trưng, huấn luyện và đánh giá mô hình NB và SVM trên cả hai nền tảng MapReduce và Spark. Các chỉ số đánh giá như độ chính xác, thời gian xử lý và khả năng mở rộng được sử dụng để so sánh hiệu suất của các mô hình.

Các kết quả thu được từ dự án này cung cấp cái nhìn tổng quan về hiệu quả của việc kết hợp các thuật toán học máy với các nền tảng xử lý dữ liệu lớn trong phân tích cảm xúc văn bản, đồng thời đề xuất hướng phát triển ứng dụng mô hình học sâu và xử lý dữ liệu thời gian thực trong tương lai.

Bạn có thể truy cập mã nguồn trên GitHub tại đây : Text-Sentiment-Classification-Hadoop-Spark

Chương 1

Giới thiệu chung

1.1 Bối cảnh

Trong thời đại công nghệ số hiện nay, khối lượng dữ liệu văn bản phát sinh từ các nền tảng như mạng xã hội, diễn đàn trực tuyến và hệ thống phản hồi khách hàng đã và đang tăng với tốc độ ngày càng nhanh chóng. Việc phân tích cảm xúc từ dữ liệu văn bản trở thành một công cụ quan trọng để doanh nghiệp và tổ chức nắm bắt suy nghĩ, hành vi và xu hướng của người dùng. Tuy nhiên, bài toán phân tích cảm xúc văn bản đặt ra nhiều thách thức khi phải xử lý dữ liệu lớn trong thời gian ngắn, đòi hỏi hệ thống có khả năng mở rộng và tính toán hiệu quả.
Song song với sự phát triển của các thuật toán học máy như Naive Bayes và Support Vector Machine (SVM), các nền tảng xử lý dữ liệu lớn như Hadoop MapReduce và Apache Spark ngày càng được ứng dụng rộng rãi trong lĩnh vực phân tích dữ liệu. Việc kết hợp các mô hình học máy truyền thống với nền tảng tính toán phân tán mở ra hướng đi mới trong việc xây dựng các hệ thống phân tích cảm xúc hiệu quả và linh hoạt.

1.2 Mục tiêu và phạm vi

Mục tiêu của dự án là xây dựng và so sánh hiệu quả hai mô hình phân tích cảm xúc văn bản sử dụng Naive Bayes và SVM trên hai nền tảng xử lý phân tán : Hadoop MapReduce và Apache Spark. Cụ thể, nhóm thực hiện :

1. Xây dựng pipeline xử lý dữ liệu văn bản lớn cho hai thuật toán Naive Bayes và SVM.
2. Triển khai các pipeline trên MapReduce và Spark để so sánh hiệu quả về độ chính xác và thời gian xử lý.
3. Phân tích, đánh giá kết quả thực nghiệm và đưa ra nhận xét về ưu, nhược điểm của từng mô hình trên từng nền tảng.

Phạm vi dự án tập trung vào việc hướng dẫn sử dụng MapReduce và Spark cho việc xử lý và phân tích dữ liệu văn bản lớn có nhãn (labelled data), sử dụng tập dữ liệu được chuẩn hóa để huấn luyện và đánh giá mô hình. Dự án không đề cập đến việc xử lý dữ liệu không có nhãn hay ứng dụng học sâu (deep learning).

1.3 Đóng góp chính

Dự án mang lại một số đóng góp chính như sau :

1. Triển khai và so sánh hai mô hình học máy kinh điển (Naive Bayes và SVM) trong bài toán phân tích cảm xúc văn bản lớn.

2. Tích hợp và đánh giá khả năng xử lý song song, phân tán của hai nền tảng Hadoop MapReduce và Apache Spark trong bài toán thực tế.
3. Cung cấp cái nhìn trực quan và định lượng về hiệu quả giữa các mô hình và nền tảng, từ đó làm cơ sở cho các nghiên cứu và ứng dụng tiếp theo trong lĩnh vực xử lý ngôn ngữ tự nhiên và phân tích dữ liệu lớn.

Chương 2

Cơ sở lý thuyết

2.1 Phân tích cảm xúc văn bản

Phân tích cảm xúc văn bản (Text Sentiment Analysis) là quá trình tự động xác định thái độ, quan điểm hoặc cảm xúc của người viết khi bày tỏ ý kiến về một chủ thể nào đó trong văn bản. Các bước chính thường bao gồm :

1. **Tiền xử lý văn bản** :
   - Loại bỏ dấu câu, số, ký tự đặc biệt.
   - Chuyển toàn bộ ký tự về dạng thường (lowercase).
   - Tách từ, gỡ bỏ từ dừng (stop words), thực hiện stemming/lemmatization.
2. **Trích xuất đặc trưng** :
   - Mã hóa bằng Bag-of-Words (BoW) hoặc TF–IDF.
   - Sử dụng các mô hình nhúng từ (Word Embeddings) như Word2Vec, GloVe.
3. **Phân loại cảm xúc** :
   - Nhị phân (positive/negative) hoặc đa lớp (positive, neutral, negative).
   - Áp dụng các thuật toán học máy truyền thống (Naive Bayes, SVM) hoặc mô hình học sâu (CNN, RNN).
4. **Đánh giá** :
   - Các chỉ số : Accuracy, Precision, Recall, F1-score.
   - Ma trận nhầm lẫn (Confusion Matrix).

2.2 Thuật toán Naive Bayes

1. Nguyên lý cơ bản

Naive Bayes là một bộ phân loại xác suất dựa trên định lý Bayes với giả thiết độc lập (naive) giữa các đặc trưng. Cho một văn bản d được biểu diễn bởi vector đặc trưng $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, xác suất thuộc về lớp C_k được tính theo công thức :

$$P(C_k \mid \mathbf{x}) = \frac{P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)}{P(\mathbf{x})}$$

Trong đó :

- $P(C_k)$ là xác suất tiên nghiệm của lớp C_k.
- $P(x_i \mid C_k)$ là xác suất điều kiện của đặc trưng x_i khi biết lớp C_k.
- Mẫu được gán cho lớp có giá trị $P(C_k \mid \mathbf{x})$ lớn nhất.

2. Ưu nhược điểm

Ưu điểm :

- Đơn giản, dễ triển khai.
- Hiệu quả trên dữ liệu nhỏ hoặc trung bình.
- Tốc độ huấn luyện và phân loại nhanh.

Nhược điểm :

- Giả thiết độc lập giữa các từ không luôn đúng trong thực tế.
- Nhạy cảm với đặc trưng hiếm (rare features).

2.3 Thuật toán Support Vector Machines

1. Nguyên lý cơ bản

Support Vector Machine (SVM) là thuật toán phân loại tìm siêu phẳng (hyperplane) tối ưu để tách hai lớp dữ liệu sao cho khoảng cách (margin) giữa siêu phẳng và hai lớp là lớn nhất. Đối với dữ liệu tuyến tính, bài toán tối ưu :

$$\begin{aligned} &\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{thoả mãn} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

Trong đó $\mathbf{w}$ và b xác định siêu phẳng, $y_i \in \{-1, +1\}$ là nhãn lớp.

2. Kernel Trick

Khi dữ liệu không tách tuyến tính, SVM sử dụng hàm kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ để ánh xạ lên không gian đặc trưng cao chiều :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j),$$

các kernel phổ biến : polynomial, RBF (Gaussian), sigmoid.

3. Ưu nhược điểm

Ưu điểm :

- Hiệu quả trong không gian đặc trưng cao chiều.
- Ổn định với overfitting nhờ tối ưu margin.

Nhược điểm :

- Thời gian huấn luyện lâu với tập dữ liệu lớn.
- Cần tinh chỉnh tham số kernel và hệ số điều chỉnh C.

2.4 Các công cụ xử lý phân tán Big Data

2.4.1 Hadoop MapReduce

Tổng quan

- **Định nghĩa** : MapReduce là một framework xử lý dữ liệu phân tán, cho phép xử lý song song trên các cụm máy tính lớn, dựa trên mô hình lập trình "Map" và "Reduce".
- **Ra mắt** : Được Google giới thiệu trong một bài báo năm 2004, sau đó Apache Hadoop triển khai phiên bản mã nguồn mở.
- **Mục đích** : Xử lý dữ liệu lớn, chịu lỗi, và có khả năng mở rộng trên hàng nghìn node.

Cách hoạt động

- **Map Phase** : Dữ liệu đầu vào được chia thành các phần nhỏ (splits), mỗi phần được xử lý song song bởi các tác vụ Map, tạo ra các cặp key-value trung gian.
- **Shuffle and Sort** : Các cặp key-value được sắp xếp và phân nhóm theo key, sau đó chuyển đến giai đoạn Reduce.
- **Reduce Phase** : Các cặp key-value được xử lý để tạo ra kết quả cuối cùng.
- **JobTracker và TaskTracker** (Hadoop 1.x) : JobTracker quản lý công việc, TaskTracker thực thi các tác vụ Map và Reduce.
- **YARN** (Hadoop 2.x trở lên) : Thay thế JobTracker/TaskTracker bằng ResourceManager và NodeManager để quản lý tài nguyên và lập lịch.

Thành phần chính

- **HDFS (Hadoop Distributed File System)** : Hệ thống tệp phân tán lưu trữ dữ liệu, làm nền tảng cho MapReduce.
- **MapReduce API** : Hỗ trợ viết chương trình bằng Java, Python (qua Hadoop Streaming), hoặc các ngôn ngữ khác.
- **InputFormat và OutputFormat** : Xác định cách dữ liệu được đọc và ghi (TextInputFormat, SequenceFileInputFormat, v.v.).
- **Combiner** : Tối ưu hóa bằng cách giảm dữ liệu trung gian trước khi gửi đến Reduce.
- **Partitioner** : Quyết định cách phân phối dữ liệu trung gian đến các Reducer.

Ứng dụng thực tế

- **Xử lý dữ liệu lớn** : Phân tích log, dữ liệu web, dữ liệu giao dịch trong tài chính, thương mại điện tử.
- **ETL (Extract, Transform, Load)** : Chuẩn hóa và chuyển đổi dữ liệu cho kho dữ liệu.
- **Phân tích dữ liệu** : Tạo báo cáo, xử lý dữ liệu thống kê, hoặc tìm kiếm văn bản.

Ưu điểm

- **Khả năng mở rộng** : Có thể xử lý dữ liệu petabyte trên hàng nghìn node.

- **Chịu lỗi** : Tự động khôi phục khi node bị lỗi nhờ cơ chế sao chép dữ liệu trong HDFS.
- **Đơn giản** : Mô hình MapReduce dễ hiểu, chỉ yêu cầu hai hàm chính (Map và Reduce).
- **Tích hợp** : Hoạt động tốt với HDFS, HBase, Hive, Pig.

Hạn chế

- **Hiệu suất** : Chậm hơn so với các công nghệ như Apache Spark do phụ thuộc vào lưu trữ đĩa (disk-based) thay vì bộ nhớ (in-memory).
- **Phức tạp** : Viết chương trình MapReduce bằng Java có thể dài dòng và phức tạp.
- **Không phù hợp với thời gian thực** : Không hỗ trợ tốt xử lý dữ liệu luồng (streaming).
- **Lập trình cấp thấp** : Yêu cầu nhà phát triển xử lý nhiều chi tiết kỹ thuật so với các công cụ cấp cao như Spark SQL.

Triển khai, cộng đồng và công cụ hỗ trợ

- **Môi trường triển khai** : Chạy trên cụm Hadoop, hỗ trợ bởi YARN, có thể tích hợp với AWS EMR, Google Cloud Dataproc, Azure HDInsight.
- **Công cụ liên quan** :
  - **Hive** : Chuyển truy vấn SQL thành các job MapReduce.
  - **Pig** : Ngôn ngữ cấp cao để viết pipeline xử lý dữ liệu.
  - **Oozie** : Lập lịch và quản lý các job MapReduce.
- **Cài đặt** : Yêu cầu cấu hình HDFS và YARN, thường phức tạp trong môi trường lớn.
- **Mã nguồn mở** : Được duy trì bởi Apache Software Foundation với cộng đồng lớn.

2.4.2 Apache Spark

Tổng quan

- **Định nghĩa** : Apache Spark là một hệ thống tính toán phân tán dựa trên mô hình xử lý dữ liệu song song, tập trung vào hiệu suất và khả năng mở rộng.
- **Ra mắt** : Được phát triển tại UC Berkeley AMPLab vào năm 2009, trở thành dự án mã nguồn mở của Apache vào năm 2010.
- **Đặc điểm chính** :
  - **Tốc độ cao** : Nhờ cơ chế tính toán trong bộ nhớ (in-memory computing), Spark nhanh hơn nhiều so với Hadoop MapReduce.
  - **Dễ sử dụng** : Hỗ trợ nhiều ngôn ngữ lập trình như Scala, Java, Python, R (qua API PySpark, SparkR).
  - **Khả năng mở rộng** : Có thể chạy trên hàng nghìn node trong cụm máy chủ.
  - **Tích hợp** : Tích hợp tốt với Hadoop HDFS, Apache Cassandra, HBase, và các hệ thống lưu trữ khác.

Thành phần cốt lõi

Spark cung cấp một bộ công cụ tích hợp trong hệ sinh thái của nó :

- **Spark Core** : Lõi của Spark, cung cấp các chức năng cơ bản như lập lịch tác vụ, quản lý bộ nhớ, và xử lý dữ liệu phân tán.
- **Spark SQL** : Hỗ trợ truy vấn dữ liệu có cấu trúc và bán cấu trúc bằng SQL hoặc DataFrame/DataSet API.
- **Spark Streaming** : Xử lý dữ liệu thời gian thực (real-time) theo luồng (stream processing).
- **MLlib** : Thư phối học máy (machine learning) với các thuật toán phân tích dữ liệu.
- **GraphX** : Thư viện xử lý đồ thị và tính toán đồ thị phân tán.

Cách hoạt động

- **RDD (Resilient Distributed Dataset)** : Đơn vị dữ liệu cơ bản của Spark, là tập hợp dữ liệu phân tán có khả năng chịu lỗi và xử lý song song.
- **DAG (Directed Acyclic Graph)** : Spark sử dụng DAG để tối ưu hóa kế hoạch thực thi các tác vụ.
- **Cơ chế in-memory** : Dữ liệu được lưu trữ chủ yếu trong RAM, giảm thời gian truy xuất so với lưu trữ trên đĩa.
- **SparkContext** : Điểm khởi tạo để kết nối ứng dụng với cụm Spark.
- **Driver và Executors** : Driver quản lý luồng công việc, trong khi Executors thực thi các tác vụ trên các node.

Ứng dụng thực tế

- **Xử lý big data** : Phân tích dữ liệu lớn trong tài chính, y tế, thương mại điện tử.
- **Học máy** : Xây dựng mô hình dự đoán, phân loại, khuyến nghị.
- **Xử lý luồng dữ liệu** : Phân tích log thời gian thực, giám sát hệ thống.
- **Phân tích dữ liệu** : Báo cáo kinh doanh, xử lý dữ liệu từ IoT, khoa học dữ liệu.

Ưu điểm

- Hiệu suất cao nhờ tính toán trong bộ nhớ.
- Hỗ trợ đa ngôn ngữ, dễ tiếp cận cho nhà phát triển.
- Hệ sinh thái phong phú, tích hợp nhiều công cụ.
- Khả năng xử lý cả dữ liệu theo lô (batch) và luồng (streaming).

Hạn chế

- Yêu cầu tài nguyên lớn (RAM, CPU) để đạt hiệu suất tối ưu.
- Cấu hình cụm phức tạp trong môi trường lớn.
- Không phù hợp cho các tác vụ yêu cầu xử lý giao dịch thời gian thực với độ trễ cực thấp.

So sánh với Hadoop và MapReduce

- **Tốc độ** : Spark nhanh hơn Hadoop MapReduce nhờ tính toán trong bộ nhớ.
- **Dễ sử dụng** : API của Spark thân thiện hơn so với MapReduce.
- **Xử lý luồng** : Spark Streaming linh hoạt hơn Hadoop.
- **Phụ thuộc** : Spark có thể tích hợp với HDFS, nhưng không yêu cầu Hadoop.

- **Hệ sinh thái** : Spark có các thư viện tích hợp (MLlib, GraphX), trong khi MapReduce cần các công cụ bổ sung như Hive, Pig.

Cộng đồng và triển khai

- **Mã nguồn mở** : Được hỗ trợ bởi cộng đồng lớn và Apache Software Foundation.
- **Triển khai** : Có thể chạy trên local, standalone cluster, hoặc trên các nền tảng như AWS, Google Cloud, Azure, Kubernetes, YARN, Mesos.
- **Công cụ quản lý** : Databricks (dựa trên Spark), Apache Zeppelin, Jupyter Notebook.

Chương 3

Triển khai và thực nghiệm

3.1 Tập dữ liệu

Tập dữ liệu được sử dụng trong dự án là tập **Sentiment140**, bao gồm **1.6 triệu dòng tweet** được gán nhãn cảm xúc. Mỗi dòng có định dạng dạng CSV gồm 3 trường :

- ID của tweet,
- Nhãn cảm xúc (positive hoặc negative),
- Nội dung tweet.

Ví dụ :

```
0,Sentiment140,I must think about positive..
```

Dữ liệu được chia thành 10 tập con có kích thước từ **100.000 đến 1.000.000 tweet**, được lưu tại thư mục `input/`. Trong đó :

- Đối với Hadoop : các tệp có tên `train#` và `test#` (# từ 1 đến 10).
- Đối với Spark : các tệp có tên `spark_input_#` (# từ 1 đến 10).

Trước khi thực thi, hệ thống yêu cầu khởi động HDFS và YARN :

```
start-dfs.sh
start-yarn.sh
```

Chuẩn bị dữ liệu đầu vào
Dữ liệu cần được tải lên HDFS để Hadoop có thể xử lý :

```
hdfs dfs -mkdir -p /user/username/input_text_sentiment
hdfs dfs -put input/ /user/username/input_text_sentiment/
```

3.2 Môi trường và công cụ phát triển

Dự án được phát triển và triển khai trên các công nghệ mã nguồn mở phổ biến :

- **Apache Hadoop** : để xây dựng mô hình MapReduce và xử lý dữ liệu phân tán.
- **Apache Spark** : để xây dựng mô hình xử lý song song trên bộ nhớ.
- **Ngôn ngữ Java** : dùng để triển khai các ứng dụng MapReduce trên Hadoop.
- **Ngôn ngữ Python** : dùng để triển khai các ứng dụng trên Spark.

3.3 Triển khai Naive Bayes trên Hadoop MapReduce

3.3.1 Biên dịch và thực thi chương trình

Thư mục chứa mã nguồn Naive Bayes nằm tại `Hadoop/NB`. Cần tạo thư mục chứa file biên dịch, biên dịch mã Java và tạo file JAR như sau :

```
cd Hadoop/NB
mkdir NB_classes
javac -classpath "$(yarn classpath)" -d NB_classes NB.java
jar -cvf NB.jar -C NB_classes/ .
```

Sau khi biên dịch thành công, thực thi chương trình bằng lệnh :

```
hadoop jar NB.jar NB \
  /input_text_sentiment/train# \
  /input_text_sentiment/test# \
  training_split testing_split
```

`train#` và `test#` là các file dữ liệu đã tải lên HDFS từ thư mục `input/`. Các tham số `training_split` và `testing_split` là số byte để chia nhỏ file dữ liệu cho các `Mapper`.

3.3.2 Cấu trúc chương trình

Chương trình được triển khai dựa trên mô hình lập trình MapReduce của Hadoop, bao gồm hai giai đoạn chính : huấn luyện (training) và kiểm tra (testing). Mỗi giai đoạn được chia thành các lớp thành phần có vai trò riêng biệt như sau :

- **Lớp chính :** `NB`
  Là lớp chính chứa định nghĩa cho các Mapper và Reducer trong cả hai giai đoạn training và testing. Ngoài ra, lớp này còn định nghĩa các bộ đếm (counters) toàn cục để thu thập các thông số thống kê cần thiết cho việc tính toán xác suất trong mô hình Naive Bayes.

Giai đoạn Huấn luyện (Training)

- **Mapper :** `Map_Training`
  Chức năng :
  - Đọc từng dòng dữ liệu đầu vào (mỗi dòng là một tweet).
  - Làm sạch nội dung tweet (xóa link, hashtag, số, dấu câu,...).
  - Phân tách văn bản thành các từ và gắn nhãn tương ứng với cảm xúc (POSITIVE hoặc NEGATIVE).
  - Tăng các bộ đếm thống kê số tweet và số từ theo từng nhãn.
  - Đầu ra : `<word, sentiment>`
- **Reducer :** `Reduce_Training`
  Chức năng :
  - Nhận vào một từ và danh sách nhãn tương ứng từ các tweet chứa từ đó.
  - Đếm số lần từ xuất hiện trong các tweet có nhãn POSITIVE và NEGATIVE.
  - Tăng bộ đếm số lượng đặc trưng (features).
  - Đầu ra : `<word, pos_count@neg_count>`

Giai đoạn Kiểm tra (Testing)

- **Mapper :** `Map_Testing`
  Chức năng :
  - Nạp các thông số thống kê từ giai đoạn training : số lượng tweet, từ, xác suất lớp,...
  - Nạp mô hình Naive Bayes từ đầu ra của Reducer trong bước training.
  - Tính xác suất của mỗi tweet thuộc lớp POSITIVE và NEGATIVE bằng công thức Naive Bayes (có sử dụng Laplace smoothing).
  - Dự đoán nhãn cho mỗi tweet dựa trên xác suất cao hơn.
  - So sánh với nhãn thực tế để cập nhật các bộ đếm : `TRUE_POSITIVE`, `FALSE_POSITIVE`, `TRUE_NEGATIVE`, `FALSE_NEGATIVE`.
- **Reducer (nếu có) :** Không cần Reducer trong pha kiểm tra vì mỗi mapper đã tự thực hiện dự đoán độc lập.

Quy trình tổng quát

1. Chạy MapReduce job để huấn luyện mô hình từ tập dữ liệu đầu vào.
2. Xuất mô hình ra file (`training/`).
3. Chạy MapReduce job để kiểm tra mô hình trên dữ liệu kiểm tra.
4. Dựa vào các bộ đếm kết quả để tính các chỉ số như độ chính xác, độ nhạy, độ đặc hiệu,...

3.4 Triển khai SVM trên Hadoop MapReduce

Trong phần này, chúng tôi triển khai một mô hình SVM (Support Vector Machine) tuyến tính đơn giản sử dụng MapReduce trong Hadoop. Việc triển khai bao gồm hai giai đoạn chính : huấn luyện (training) và kiểm tra (testing), được thực hiện lặp lại trong nhiều epoch.

3.4.1 Biên dịch và thực thi chương trình

Thư mục chứa mã nguồn Naive Bayes nằm tại `Hadoop/SVM`. Cần tạo thư mục chứa file biên dịch, biên dịch mã Java và tạo file JAR như sau :

```
cd Hadoop/SVM
mkdir SVM_classes
javac -classpath "$(yarn classpath)" -d SVM_classes SVM.java
jar -cvf SVM.jar -C SVM_classes/ .
```

Sau khi biên dịch thành công, thực thi chương trình bằng lệnh :

```
hadoop jar SVM.jar SVM \
  /input_text_sentiment/train# \
  /input_text_sentiment/test# \
  training_split testing_split
```

`train#` và `test#` là các file dữ liệu đã tải lên HDFS từ thư mục `input/`. Các tham số `training_split` và `testing_split` là số byte để chia nhỏ file dữ liệu cho các `Mapper`.

3.4.2 Cấu trúc tổng thể chương trình

Chương trình chính được viết trong lớp `SVM.java` với các phần cấu thành sau :

- **Các lớp phụ trợ như :** `Mapper`, `Reducer`.
- **Các tham số huấn luyện :** Gồm số epoch (`EPOCHS = 5`), learning rate ban đầu (`INITIAL_LR = 0.1`) và hệ số điều chuẩn (`REGULARIZATION = 0.01`).
- **Tiền xử lý dữ liệu :** Sử dụng biểu thức chính quy để loại bỏ URL và ký tự không phải chữ cái.

3.4.3 Giai đoạn huấn luyện

Quá trình huấn luyện được lặp lại qua nhiều epoch. Tại mỗi epoch, mô hình SVM được cập nhật trọng số dựa trên gradient của hàm mất mát hinge loss.

- **Mapper :** Được định nghĩa trong lớp `Map_Training`. Mapper này :
  - Tải trọng số từ mô hình của epoch trước đó (nếu có).
  - Tiền xử lý văn bản đầu vào.
  - Tính tích vô hướng giữa vector đặc trưng và vector trọng số.
  - Nếu mẫu sai (margin < 1), cập nhật gradient dựa trên nhãn và đặc trưng.
- **Reducer :** Trong lớp `Reduce_Training`, các trọng số cập nhật được cộng dồn lại từ nhiều Mapper để cho ra mô hình của epoch hiện tại.

3.4.4 Giai đoạn kiểm tra (Testing)

Sau khi kết thúc huấn luyện, mô hình tại epoch cuối cùng được sử dụng để kiểm tra tập dữ liệu mới. Quá trình này được thực hiện bởi Mapper `Map_Testing` :

- Tải mô hình từ epoch cuối cùng.
- Tiền xử lý văn bản và tính điểm dự đoán (*score*).
- Gán nhãn dự đoán dựa vào dấu của điểm số (`score` $\geq$ `0` $\Rightarrow$ dương).
- Ghi nhận thông tin thống kê như TP, FP, TN, FN bằng Hadoop Counters.

3.5 Triển khai pipeline Naive Bayes và SVM trên Apache Spark

3.5.1 Khởi tạo môi trường Spark

- Tạo `SparkConf` và `SparkContext` để thiết lập ứng dụng Spark với tên "Naive Bayes" (nếu thuật toán được sử dụng là Naive Bayes) hoặc "Support Vector Machines" (nếu thuật toán là SVM).
- Tạo `SparkSession` để làm việc với DataFrame.

3.5.2 Đọc và tiền xử lý dữ liệu

- Đọc dữ liệu từ tệp CSV trên HDFS bằng `sc.textFile`.
- Áp dụng hàm `split_csv` để tách các dòng thành các cột, xử lý trường hợp có nhiều cột bằng cách nối các cột từ cột thứ 4 trở đi.
- Sử dụng hàm `clean_text` để làm sạch văn bản tweet : loại bỏ URL, hashtag, ký tự đặc biệt, số, chuyển về chữ thường và chuẩn hóa khoảng trắng.
- Tạo DataFrame từ RDD với các cột `label` (sentiment) và `tweet` (văn bản đã làm sạch).

3.5.3 Xây dựng pipeline xử lý đặc trưng

- **Tokenization** : Sử dụng `Tokenizer` để chia tweet thành danh sách các từ, lưu vào cột `words`.
- **HashingTF** : Chuyển danh sách từ thành vector đặc trưng thô (`rawFeatures`) bằng cách ánh xạ từ vào không gian vector cố định.
- **IDF** : Áp dụng Inverse Document Frequency để tính trọng số TF-IDF cho các đặc trưng, tạo cột `features`.

3.5.4 Chia dữ liệu và huấn luyện mô hình

- Chia tập dữ liệu thành tập huấn luyện (75%) và tập kiểm tra (25%) với seed cố định.
- Huấn luyện mô hình Naive Bayes hoặc SVM trên tập huấn luyện sử dụng `Naive Bayes` hoặc `LinearSVC` từ `pyspark.ml.classification`.

3.5.5 Dự đoán, đánh giá và đo thời gian thực thi

- Dự đoán trên tập kiểm tra để tạo cột `prediction`.
- Chuyển kết quả dự đoán và nhãn thực tế thành RDD để tính toán các chỉ số đánh giá.
- Sử dụng `MulticlassMetrics` để tính ma trận nhầm lẫn, độ chính xác (accuracy) và F1-score.
- Ghi nhận thời gian bắt đầu và kết thúc để tính thời gian thực thi của toàn bộ pipeline.

3.5.6 Kết thúc

- In ra ma trận nhầm lẫn, độ chính xác, F1-score và thời gian thực thi.
- Đóng `SparkContext` bằng `sc.stop()`.

Tóm lại, pipeline này thực hiện quy trình phân loại cảm xúc (sentiment analysis) trên dữ liệu tweet bằng cách làm sạch văn bản, chuyển đổi thành đặc trưng TF-IDF, huấn luyện mô hình Naive Bayes, và đánh giá hiệu suất trên Spark.

3.6 Thiết kế chỉ số đánh giá

Sau khi hoàn tất kiểm tra, mô hình xuất ra các chỉ số đánh giá gồm :

- **Độ chính xác (Accuracy) :**

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Độ chính xác theo lớp dương (Precision) :**

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Độ bao phủ (Recall) :**

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score :**

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Tất cả các giá trị thống kê được thu thập từ Hadoop Counters và in ra sau khi kết thúc chương trình. Việc thực thi toàn bộ pipeline MapReduce cho phép xử lý dữ liệu văn bản lớn một cách hiệu quả trên cụm Hadoop phân tán.

Chương 4

Kết quả và thảo luận

4.1 So sánh về hiệu năng giữa Naive Bayes và SVM

Trong thí nghiệm này, hai mô hình học máy gồm **Naive Bayes** và **Support Vector Machines (SVM)** được triển khai trên nền tảng **Hadoop MapReduce** và **Apache Spark**. Cả hai thuật toán đều được áp dụng trên cùng một tập dữ liệu huấn luyện và kiểm thử để đảm bảo tính khách quan trong việc đánh giá hiệu năng.

Bảng 4.1 – So sánh kết quả giữa Naive Bayes và SVM trên Hadoop

Chỉ số	**Naive Bayes**	**SVM**
Thời gian thực thi (giây)	481.10	1656.86
Độ chính xác (Accuracy)	0.6440	0.7688
F1-Score	0.7311	0.8060
Độ chính xác dự đoán (Precision)	0.6599	0.7988
Khả năng thu hồi (Recall)	0.8195	0.8133

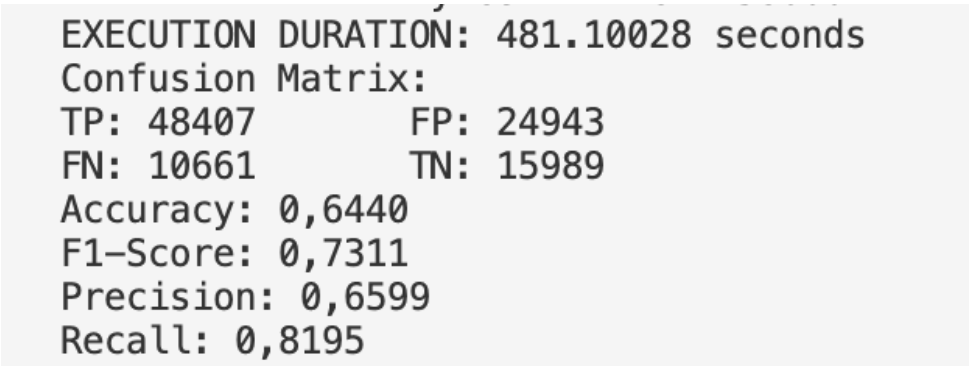
Bảng 4.2 – So sánh kết quả giữa Naive Bayes và SVM trên Spark

Chỉ số	**Naive Bayes**	**SVM**
Thời gian thực thi (giây)	72.05	78.78
Độ chính xác (Accuracy)	0.7310	0.7615
F1-Score	0.7324	0.7581

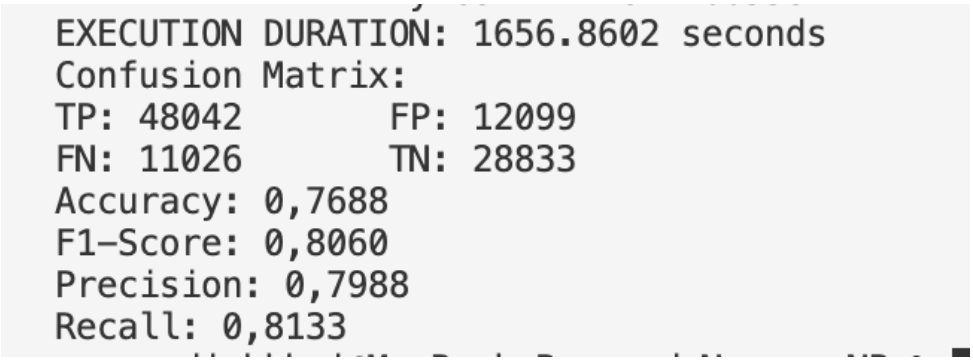
Từ bảng 4.1, ta có thể thấy, mặc dù **Naive Bayes** có thời gian thực thi nhanh hơn đáng kể khi chạy trên **MapReduce** (chỉ mất khoảng 481 giây so với 1656 giây của **SVM**), nhưng hiệu quả phân loại của **SVM** lại cao hơn rõ rệt về mọi mặt : độ chính xác, precision, F1-score. Chỉ riêng recall của **Naive Bayes** là cao hơn một chút.
Trong khi đó, bảng 4.2 cho ta thấy khoảng cách về thời gian huấn luyện giữa hai thuật toán này đã thu hẹp đáng kể khi chạy trên Spark (72 giây cho **Naive Bayes** và 78 giây cho **SVM**). **SVM** vẫn cho ra kết quả chính xác cao hơn, tuy nhiên mọi khoảng cách đã bị thu hẹp đáng kể.
Qua thực nghiệm trên, ta có thể rút ra nhận xét về hai thuật toán Naive Bayes và SVM như sau :

- Naive Bayes có tốc độ thực thi nhanh hơn hẳn SVM. Điều này được lý giải là vì Naive Bayes có độ phức tạp tính toán thấp hơn, dựa trên giả định độc lập giữa các đặc trưng, trong khi SVM cần tính toán các vector hỗ trợ và tối ưu hóa lề, đặc biệt tốn kém với dữ liệu lớn.

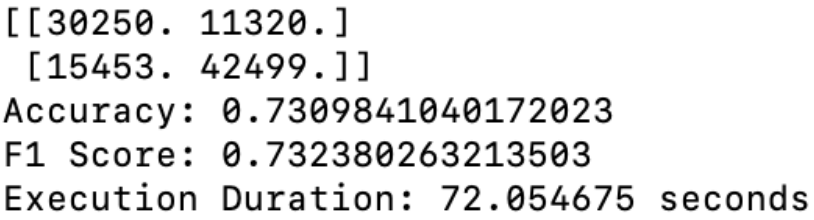
- SVM dù sở hữu tốc độ thực thi chậm hơn nhưng độ chính xác cho ra cao hơn so với Naive Bayes. Điều này phản ánh khả năng phân loại tốt hơn của SVM, đặc biệt khi dữ liệu phức tạp và không tuân theo giả định độc lập khi so với Naive Bayes.
- Do đó, Naive Bayes phù hợp với các tác vụ cần tốc độ cao và dữ liệu có giả định độc lập giữa các đặc trưng. SVM phù hợp hơn khi cần độ chính xác cao và dữ liệu phức tạp.



Hình 4.1 – Kết quả của Naive Bayes trên Hadoop



Hình 4.2 – Kết quả của SVM trên Hadoop



Hình 4.3 – Kết quả của Naive Bayes trên Spark

```
[[26684. 14886.]
 [ 8846. 49106.]]
Accuracy: 0.7615401619742369
F1 Score: 0.7581052808073749
Execution Duration: 78.780155 seconds
```

Hình 4.4 – Kết quả của SVM trên Spark

4.2 So sánh về hiệu năng giữa MapReduce và Spark

Hadoop MapReduce có hiệu năng xử lý chậm hơn rất nhiều so với Spark (Naive Bayes chậm hơn 6.7 lần, SVM chậm hơn 21 lần). Điều này là do Hadoop MapReduce sử dụng mô hình xử lý theo batch, ghi và đọc dữ liệu từ đĩa (HDFS) giữa các giai đoạn Map và Reduce, gây ra độ trễ lớn. Đặc biệt với SVM, yêu cầu tính toán phức tạp và lặp lại nhiều lần, thời gian thực thi tăng đáng kể. Trong khi đó, Spark sử dụng mô hình xử lý dựa trên bộ nhớ (in-memory computing), giảm thiểu việc đọc/ghi dữ liệu từ đĩa. Ngoài ra, Spark hỗ trợ xử lý song song hiệu quả hơn, đặc biệt với các thuật toán lặp như SVM.

Việc chạy mô hình trên Spark đem lại độ chính xác cao hơn hoặc tương đương với MapReduce (mặc dù có thấp hơn ở chỉ số F1-Score khi chạy SVM). Điều này cho ta thấy Spark cung cấp môi trường xử lý tốt hơn, giúp các thuật toán tận dụng dữ liệu hiệu quả hơn.

Bên cạnh đó, Spark phù hợp hơn cho các tác vụ học máy trên dữ liệu lớn nhờ khả năng xử lý nhanh và hỗ trợ lặp lại, trong khi Hadoop MapReduce phù hợp hơn cho các tác vụ đơn giản, không yêu cầu lặp nhiều.

Từ đó, ta có thể sử dụng MapReduce cho các hệ thống đã triển khai sẵn HDFS và không yêu cầu tốc độ cao. Còn với trường hợp cần xử lý nhanh, đặc biệt là các thuật toán học máy phức tạp như SVM, Apache Spark sẽ là lựa chọn tối ưu.

Chương 5

Kết luận và hướng phát triển mở rộng

5.1 Tóm tắt kết quả chính

Dự án đã sử dụng hai công cụ xử lý dữ liệu lớn phổ biến hiện nay là **Hadoop MapReduce** và **Apache Spark** để phát triển mô hình **Naive Bayes** và **SVM**, từ đó thực hiện **phân loại cảm xúc văn bản** dựa trên những mô hình đã phát triển.
Kết quả cho thấy **SVM** có **chất lượng phân loại tốt hơn** (Accuracy, F1-Score, Precision), nhưng **Naive Bayes nhanh hơn đáng kể**, đặc biệt trên Hadoop. Trên Spark, khoảng cách hiệu năng thu hẹp, nhưng SVM vẫn nhỉnh hơn về độ chính xác.
Ngoài ra, thông qua dự án, ta có thể khẳng định, **Spark vượt trội** về **tốc độ** và **cải thiện hiệu năng phân loại**, là lựa chọn tốt hơn cho các **tác vụ học máy** trên **dữ liệu lớn**. **Hadoop MapReduce** chậm hơn và phù hợp với các tác vụ **đơn giản hơn**.

5.2 Hướng phát triển mở rộng

5.2.1 Cải thiện thuật toán và mô hình

- **Sử dụng mô hình học sâu (Deep Learning)** : Kết hợp các mô hình như Transformer (BERT, RoBERTa) hoặc mô hình ngôn ngữ lớn (LLMs) để cải thiện độ chính xác trong phân loại cảm xúc, đặc biệt với văn bản tiếng Việt (ví dụ : PhoBERT).
- **Phân loại cảm xúc đa nhãn (Multi-label)** : Thay vì chỉ phân loại tích cực/tiêu cực, mở rộng để nhận diện nhiều cảm xúc phức tạp hơn như vui, buồn, giận dữ, sợ hãi, v.v.
- **Xử lý đa ngôn ngữ** : Xây dựng hệ thống phân loại cảm xúc cho nhiều ngôn ngữ, đặc biệt là các ngôn ngữ ít tài nguyên như tiếng Việt, sử dụng các kỹ thuật transfer learning.

5.2.2 Tối ưu hóa hiệu suất xử lý

- **Tăng cường khả năng xử lý dữ liệu lớn** : Tối ưu hóa pipeline xử lý trên Spark bằng cách sử dụng các kỹ thuật như caching, partitioning hợp lý, hoặc tích hợp Spark Streaming để xử lý dữ liệu theo thời gian thực.
- **Tích hợp với các công cụ lưu trữ hiện đại** : Thay vì chỉ sử dụng HDFS (Hadoop Distributed File System), tích hợp với các hệ thống lưu trữ như Apache Kafka, Amazon S3, hoặc Delta Lake để tăng tính linh hoạt và khả năng mở rộng.
- **Song song hóa mô hình học sâu** : Sử dụng Spark MLlib hoặc các framework như Horovod để huấn luyện các mô hình học sâu trên nhiều node, giảm thời gian huấn luyện với tập dữ liệu lớn.

5.2.3 Xử lý dữ liệu không cấu trúc

- **Kết hợp dữ liệu đa phương thức** : Ngoài văn bản, tích hợp phân tích cảm xúc từ hình ảnh, video, hoặc âm thanh (ví dụ : nhận diện cảm xúc từ giọng nói hoặc biểu cảm khuôn mặt).
- **Làm sạch dữ liệu nâng cao** : Phát triển các kỹ thuật tiền xử lý văn bản tiếng Việt tốt hơn, như xử lý tiếng lóng, từ viết tắt, hoặc lỗi chính tả, sử dụng các công cụ như Underthesea hoặc VnCoreNLP.
- **Phân tích ngữ cảnh** : Xây dựng mô hình hiểu ngữ cảnh sâu hơn (context-aware) để xử lý các câu có tính mỉa mai, ẩn dụ, hoặc văn hóa đặc thù.

5.2.4 Ứng dụng trí tuệ nhân tạo tổng quát (AGI)

- **Tích hợp với hệ thống AI tổng quát** : Sử dụng các mô hình AI lớn như Grok để hỗ trợ phân tích cảm xúc phức tạp hơn hoặc cung cấp phản hồi tương tác dựa trên kết quả phân tích.
- **Tự động hóa quy trình học** : Áp dụng AutoML trên Spark để tự động tối ưu hóa các siêu tham số (hyperparameters) hoặc lựa chọn mô hình phù hợp.

5.2.5 Xử lý thời gian thực

- **Phân tích cảm xúc thời gian thực** : Sử dụng Spark Streaming hoặc Flink để phân tích cảm xúc từ các luồng dữ liệu như mạng xã hội (X, Twitter, v.v.) ngay khi chúng xuất hiện.
- **Cảnh báo theo thời gian thực** : Phát triển hệ thống gửi thông báo hoặc cảnh báo khi phát hiện cảm xúc tiêu cực từ các nguồn dữ liệu như bình luận khách hàng hoặc bài đăng công khai.

Lời cảm ơn

Chúng em xin gửi lời cảm ơn chân thành đến giảng viên TS. Trần Hồng Việt, người đã tận tình giảng dạy và hướng dẫn chúng em trong suốt quá trình học tập môn học. Những kiến thức và kinh nghiệm quý báu mà thầy đã chia sẻ không chỉ giúp chúng em hiểu sâu hơn về kỹ thuật và công nghệ với dữ liệu lớn mà còn truyền cảm hứng để chúng em hoàn thành tốt dự án này.
Chúng em cũng xin cảm ơn Trường Đại Học Công Nghệ và Viện Trí Tuệ Nhân Tạo đã tạo điều kiện để chúng em có cơ hội thực hiện dự án. Những nguồn tài liệu và môi trường học tập lý tưởng đã hỗ trợ chúng em rất nhiều trong quá trình nghiên cứu.
Cuối cùng, xin gửi lời chúc mừng đến các thành viên trong nhóm. Sự hợp tác, nỗ lực và tinh thần trách nhiệm của mọi người là yếu tố quan trọng giúp dự án được hoàn thành đúng thời hạn với chất lượng tốt nhất.
Xin chân thành cảm ơn!

Nguyễn Đình Khải
Đoàn Quang Huy