DA5402: Machine Learning Operations Laboratory

Assignment 9

Nandhakishore C S (DA24M011)

April 25, 2025

Problem Statement

We learned about Apache Spark for parallelization and distributed computing. In the classroom training, we used an Amazon reviews dataset to compute the average star-rating for a Gourmet product. The dataset is already shared with you. Let's spice up the use case and solve it.

Refer to the README file to run the code

Task 1 [30 points]

Let's use pretrained sentiment analysis pipeline to reprocess the review texts from the data file. Getting Started with Sentiment Analysis using Python provides a good introduction to sentiment analysis. Let's use the solution presented in Section 2 (sentiment-analysis pipeline) of that article. Follow the map-reduce paradigm to script the distributed processing of the datafile for sentiment analysis. You will process every record for sentiment classification using the 'pipeline' from the article into POSITIVE or NEGATIVE label. The records should be processed in a parallel processing style across the available CPUs in your machine.

The sentiment analysis can is initialised from transforms library. Usin py spark parallel processing functions the pipeline is used to predict the sentiment. The .txt file is parsed and then converted to rdd schema before doing sentiment analysis.

```python
# Function to parse the text file
def parse_text_file(lines):
    records = []
    current_record = {}
    # Convert lines to list to get length for tqdm
    lines = list(lines)
    logger.debug(f"Parsing {len(lines)} lines")
    for line in tqdm(lines, desc="Parsing text file"):
        line = line.strip()
        if not line:
            if current_record:
                records.append(current_record)
                current_record = {}
            continue
        match = re.match(r"(\w+)/(\w+): (.*)", line)
        if match:
            category, key, value = match.groups()
            if category == "product":
                current_record[f"product_{key}"] = value
            elif category == "review":
                current_record[f"review_{key}"] = value
    if current_record:
        records.append(current_record)
    return records

# Define schema for DataFrame
schema = StructType([
    StructField("product_productId", StringType(), True),
    StructField("product_title", StringType(), True),
    StructField("product_price", StringType(), True),
    StructField("review_userId", StringType(), True),
    StructField("review_profileName", StringType(), True),
    StructField("review_helpfulness", StringType(), True),
    StructField("review_score", StringType(), True),
    StructField("review_time", StringType(), True),
    StructField("review_summary", StringType(), True),
    StructField("review_text", StringType(), True)
])

# Convert RDD to DataFrame
logger.info("Converting RDD to DataFrame")
try:
    df = spark.createDataFrame(parsed_rdd, schema)
except Exception as e:
    logger.error(f"Failed to create DataFrame: {str(e)}")
    spark.stop()
    raise
```

Task 2 [20 points]

We have a rating for each item in the dataset, which needs to discretized in to POSITIVE and NEGATIVE label. Let's use rating ¿= 3.0 as the threshold for becoming POSITIVE. Define a map-reduce logic to compute the Precision and Recall of the sentiment classifier model, assuming that the labels from the dataset are the ground truth. Display the confusion matrix.

Please follow the usual routines in maintaining your code's neatness

The sentiment score column is discretised and the True Positive, True Negative, False Positive, False Negative are calculated using map reduce paradigm.

```
try:
    confusion_rdd = df.rdd.map(map_to_confusion_matrix).reduceByKey(lambda a, b: a + b)
    # Collect results with tqdm
    confusion_counts = {}
    logger.debug("Collecting confusion matrix counts")
    for (predicted, ground_truth), count in tqdm(confusion_rdd.collect(), desc="Collecting confusion matrix counts"):
        confusion_counts[(predicted, ground_truth)] = count
except Exception as e:
    logger.error(f"Failed to compute confusion matrix: {str(e)}")
    spark.stop()
    raise

# Initialize confusion matrix components
tp = confusion_counts.get(("POSITIVE", "POSITIVE"), 0)  # True Positives
tn = confusion_counts.get(("NEGATIVE", "NEGATIVE"), 0)  # True Negatives
fp = confusion_counts.get(("POSITIVE", "NEGATIVE"), 0)  # False Positives
fn = confusion_counts.get(("NEGATIVE", "POSITIVE"), 0)  # False Negatives

# Calculate Precision and Recall
precision = tp / (tp + fp) if (tp + fp) > 0 else 0.0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0.0
```