Machine Learning with Pyspark: Recommender Systems

Key Takeaways

Task 1:
Set up the Project Environment

- **Key Takeaways**:
  - Storing secret keys in a .env file enhances security and reduces the risk of data breaches.
  - Principal Component Analysis (PCA) is crucial for dimensionality reduction, retaining as much information as possible.

Task 2:
Prepare the Dataset

- **Key Takeaways**:
  - concat_ws in PySpark concatenates multiple input columns into a single string column, aiding in effective preprocessing.
  - Combining titles and descriptions before extracting embedding vectors ensures comprehensive product representations.
  - Text embedding vectors encode semantic and contextual meanings, making them versatile for machine learning applications like recommendations.

Task 3:
Cluster Products Using K-means

- **Key Takeaways**:
  - VectorAssembler combines features for machine learning models in PySpark.
  - K-means clustering is an unsupervised algorithm that groups data points based on shared characteristics, making it ideal for product clustering.

Task 4:
Visualize Product Clusters

- **Key Takeaways**:
    - PCA effectively reduces the dimensionality of numerical data while preserving core information.
    - Combining features with a vector assembler is a necessary step before applying PCA for dimensionality reduction.

Task 5:
Highlight Recently Viewed Products

- **Key Takeaways**:
    - Clustering similar products enables tailored recommendations aligned with user preferences.
    - Filtering data to focus on recently viewed products ensures the relevance of analyses and insights to user behavior.
    - In PySpark, multi-RDD (Resilient Distributed Dataset) processing allows parallel handling of data across the cluster. Each RDD represents a batch of data that is processed simultaneously on distributed nodes, enabling efficient and scalable computation. This parallelized processing significantly reduces computation time, especially for large datasets like user activity logs.
    - However, not all operations in PySpark can or should be executed in parallel. For instance, operations like joins require data shuffling between partitions to align keys across RDDs or DataFrames. This shuffling creates network overhead and dependency between distributed nodes, making the operation less suitable for full parallelization.
    - Joins are inherently complex because they need to ensure that data from different partitions is correctly combined. To mitigate performance bottlenecks, techniques like partitioning on the join key can be applied to minimize data movement. Additionally, repartitioning your data to a single partition before performing a join can further streamline the process by centralizing the data, and reducing inter-node communication at the expense of parallelism.

Task 6:

Recommend Products Based on Recently Viewed Items

- **Key Takeaways**:
  - To recommend products based on recently viewed items, we should first identify product clusters. Once clustered, various ranking techniques can be applied to recommend products based on similarity metrics or through random selection.
  - Ranking techniques or similarity metrics applied within product clusters improve recommendation quality.