inventory_demamd_prediction

November 29, 2025

[8]:
```
import sys
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.regression import RandomForestRegressor
from pyspark.sql.functions import col, isnan, when, round
from pyspark.sql.types import DoubleType, FloatType, BooleanType

# ========================================
# CONFIGURATION
# ========================================
CONFIG = {
    "APP_NAME": "Retail_Inventory_RF_Notebook",

    # Paths
    "TRAIN_PATH": "gs://inv-demand-bucket/data/train_2017_ts_train.parquet",
    "TEST_PATH": "gs://inv-demand-bucket/data/train_2017_ts_test.parquet",
    "MODEL_OUTPUT_PATH": "gs://inv-demand-bucket/models",

    # Run Mode (Set False for full 22M row run)
    "TEST_MODE": False,
    "TEST_LIMIT": 50000,
    "NUM_PARTITIONS": 16,

    # Model Hyperparameters
    "NUM_TREES": 50,
    "MAX_DEPTH": 10,
    "SEED": 42
}

# Columns to exclude from features
EXCLUDED_COLS = [
    # Identifiers
    'id', 'date',

    # Target variable duplicates
```

```
    'unit_sales_raw', 'unit_sales',

    # Text versions
    'city', 'state', 'type', 'family',

    # Scaled versions
    'unit_sales_scaled', 'transactions_scaled', 'dcoilwtico_scaled',
    'unit_sales_minmax', 'transactions_minmax', 'dcoilwtico_minmax',

    # Less important temporal
    'year',  # Only 2017 data
    'day', 'day_of_year', 'week_of_year', 'quarter',

    # Redundant rolling features
    'sales_rolling_min_7', 'sales_rolling_max_7',
    'sales_rolling_min_14', 'sales_rolling_max_14',
    'sales_rolling_min_30', 'sales_rolling_max_30',

    # Redundant EWMA
    'sales_ewm_30',

    # Redundant difference features
    'sales_pct_change_1', 'sales_pct_change_7'
]

# Initialize Spark with Legacy Parquet Nanoseconds Config
spark = SparkSession.builder \
    .appName(CONFIG["APP_NAME"]) \
    .config("spark.sql.legacy.parquet.nanosAsLong", "true") \
    .getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

print("Spark Session Initialized with 'nanosAsLong=true'")
```

```
Spark Session Initialized with 'nanosAsLong=true'
```

[9]:
```
# ========================================
# Util functions
# ========================================

def load_data(spark, path, limit=None, repartition_n=16):
    """Loads Parquet data, optionally limits rows, and repartitions."""
    print(f"Loading data from: {path}")
    df = spark.read.parquet(path)

    if limit:
```

```
        print(f"TEST MODE: Limiting to {limit} rows.")
        df = df.limit(limit)

    return df.repartition(repartition_n)

def sanitize_features(df, exclude_cols):
    """
    Cleans features to ensure no Infinity, NaN, or Null values exist.
    Returns: (cleaned_dataframe, list_of_feature_column_names)
    """
    print("Sanitizing data (Removing Infinity/NaN)...")
    feature_cols = [c for c in df.columns if c not in exclude_cols]

    for c_name in feature_cols:
        dtype = df.schema[c_name].dataType

        if isinstance(dtype, (DoubleType, FloatType)):
            # Handle Infinity and NaN for floating points
            df = df.withColumn(c_name,
                when(col(c_name) == float('inf'), 0.0)
                .when(col(c_name) == float('-inf'), 0.0)
                .when(isnan(col(c_name)), 0.0)
                .when(col(c_name).isNull(), 0.0)
                .otherwise(col(c_name))
            )
        elif isinstance(dtype, BooleanType):
            # Cast Booleans to Integers (1/0)
            df = df.withColumn(c_name, col(c_name).cast("integer"))
            df = df.fillna(0, subset=[c_name])
        else:
            # Fill basic nulls for other types
            df = df.fillna(0, subset=[c_name])

    return df, feature_cols
```

[10]:
```
# ========================================
# Model Training - Random Forest Regressor
# ========================================

# 1. Load Data
limit = CONFIG["TEST_LIMIT"] if CONFIG["TEST_MODE"] else None
train_df = load_data(spark, CONFIG["TRAIN_PATH"], limit,␣
 ↪CONFIG["NUM_PARTITIONS"])

# 2. Clean Data & Identify Features
train_df, feature_cols = sanitize_features(train_df, EXCLUDED_COLS)
print(f"Training on {len(feature_cols)} features.")
```

```
# 3. Build Pipeline
assembler = VectorAssembler(
    inputCols=feature_cols,
    outputCol="features",
    handleInvalid="keep"
)

rf = RandomForestRegressor(
    featuresCol="features",
    labelCol="unit_sales",
    numTrees=CONFIG["NUM_TREES"],
    maxDepth=CONFIG["MAX_DEPTH"],
    seed=CONFIG["SEED"]
)

pipeline = Pipeline(stages=[assembler, rf])

# 4. Train
print("Starting Random Forest training...")
model = pipeline.fit(train_df)
print("Training completed.")

# 5. Save Model
print(f"Saving model to {CONFIG['MODEL_OUTPUT_PATH']}...")
model.write().overwrite().save(CONFIG["MODEL_OUTPUT_PATH"])
print("Model saved successfully.")

# 6. Feature Importance Check
print("\n--- Top 10 Features ---")
rf_model = model.stages[-1]
importances = rf_model.featureImportances
feats = sorted(zip(feature_cols, importances), key=lambda x: x[1], reverse=True)
for feat, score in feats[:10]:
    print(f"{feat}: {score:.4f}")
```

```
Loading data from: gs://inv-demand-bucket/data/train_2017_ts_train.parquet
Sanitizing data (Removing Infinity/NaN)…
Training on 44 features.
Starting Random Forest training…

25/11/26 21:54:46 WARN DAGScheduler: Broadcasting large task binary with size
1308.3 KiB
25/11/26 21:55:32 WARN DAGScheduler: Broadcasting large task binary with size
2.4 MiB
25/11/26 21:56:30 WARN DAGScheduler: Broadcasting large task binary with size
4.6 MiB
25/11/26 21:57:47 WARN DAGScheduler: Broadcasting large task binary with size
```

```
1223.4 KiB
25/11/26 21:57:48 WARN DAGScheduler: Broadcasting large task binary with size
8.8 MiB
25/11/26 21:59:29 WARN DAGScheduler: Broadcasting large task binary with size
2.3 MiB

Training completed.
Saving model to gs://inv-demand-bucket/models…


Model saved successfully.

--- Top 10 Features ---
store_item_month_avg_sales: 0.1490
sales_momentum_7: 0.1088
store_item_dow_avg_sales: 0.0876
sales_rolling_mean_7: 0.0834
sales_diff_1: 0.0823
sales_ewm_14: 0.0577
sales_rolling_mean_14: 0.0470
sales_diff_7: 0.0357
sales_lag_1: 0.0302
transactions: 0.0220
```

[11]:
```python
# ========================================
# Model Evaluation
# ========================================

# 1. Load Test Data
# We apply limit here too if TEST_MODE is on
limit = CONFIG["TEST_LIMIT"] if CONFIG["TEST_MODE"] else None
test_df = load_data(spark, CONFIG["TEST_PATH"], limit, CONFIG["NUM_PARTITIONS"])

# 2. Clean Test Data (Must apply same cleaning as training!)
test_df, _ = sanitize_features(test_df, EXCLUDED_COLS)

# 3. Load Model
print(f"Loading model from {CONFIG['MODEL_OUTPUT_PATH']}...")
loaded_model = PipelineModel.load(CONFIG["MODEL_OUTPUT_PATH"])
print("Model loaded.")

# 4. Generate Predictions
print("Running inference...")
predictions = loaded_model.transform(test_df)

# 5. Calculate RMSE
evaluator = RegressionEvaluator(
    labelCol="unit_sales",
```

```
    predictionCol="prediction",
    metricName="rmse"
)
rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

# 6. Show Actual vs Predicted
print("\n--- Sample Predictions ---")
results = predictions.select(
    col("id"),
    col("store_nbr"),
    col("item_nbr"),
    col("unit_sales").alias("Actual_unit_sales"),
    col("prediction").alias("Predicted_unit_sales"),
    round(col("unit_sales") - col("prediction"), 2).alias("Diff")
)
results.show(10)
```

```
Loading data from: gs://inv-demand-bucket/data/train_2017_ts_test.parquet
Sanitizing data (Removing Infinity/NaN)…
Loading model from gs://inv-demand-bucket/models…


Model loaded.
Running inference…


Root Mean Squared Error (RMSE): 13.6885

--- Sample Predictions ---
[Stage 140:============================>                            (6 + 6) / 12]
+---------+---------+--------+------+------------------+-----+
|       id|store_nbr|item_nbr|Actual|         Predicted| Diff|
+---------+---------+--------+------+------------------+-----+
|124792795|        8| 1366212|   8.0| 7.486722209483122| 0.51|
|124046636|        5| 1999114|   1.0|1.2147200753698628|-0.21|
|124585499|        7| 1346628|   5.0| 4.582576117662974| 0.42|
|124687725|        7|  108797|   8.0| 8.024190262625309|-0.02|
|124362990|        6| 2043849|   9.0|   8.8408151944928| 0.16|
|124469174|        5|  222975|   4.0|3.8154554691925715| 0.18|
|124986970|        6| 1157561|  17.0|15.932059339019474| 1.07|
|125196180|        6|  407542|  12.0|  12.0869424219287|-0.09|
|124891849|        8|  205387|   5.0|  5.12315320578576|-0.12|
|124890873|        7| 1390351|   2.0|1.9166414031966408| 0.08|
+---------+---------+--------+------+------------------+-----+
only showing top 10 rows
```

[ ]:

[ ]: