

Task 1

```
!curl https://raw.githubusercontent.com/mosesyh/de300-2025sp-class/refs/heads/main/agnews_clean.csv -O
```

```

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
             Dload  Upload   Total      Spent      Left     Speed
100 33.2M  100 33.2M    0     0  21.3M      0  0:00:01  0:00:01 --:--:-- 21.3M

```

```

from pyspark.sql import SparkSession

spark = (SparkSession.builder
        .master("local[*]")
        .appName("AG news")
        .getOrCreate()
        )

agnews = spark.read.csv("agnews_clean.csv", inferSchema=True, header=True)

# turning the second column from a string to an array
import pyspark.sql.functions as F
from pyspark.sql.types import ArrayType, StringType
agnews = agnews.withColumn('filtered', F.from_json('filtered', ArrayType(StringType())))

if 'id' not in agnews.columns and '_c0' in agnews.columns:
    agnews = agnews.withColumnRenamed('_c0', 'id')

```

```

# each row contains the document id and a list of filtered words
agnews.show(5, truncate=30)

```

```

+---+-----+
| id|          filtered|
+---+-----+
| 0|[wall, st, bears, claw, bac...|
| 1|[carlyle, looks, toward, co...|
| 2|[oil, economy, cloud, stock...|
| 3|[iraq, halts, oil, exports,...|
| 4|[oil, prices, soar, time, r...|
+---+-----+
only showing top 5 rows

```

```

# Q1
from pyspark.sql import functions as F

# TF Function
def tf_mapper(df):
    """Explode tokens → TF per (doc, term)."""
    exploded = df.select('id', F.explode('filtered').alias('term'))
    term_freq = exploded.groupBy('id', 'term').count() \
        .withColumnRenamed('count', 'term_count')
    doc_len = exploded.groupBy('id').count() \
        .withColumnRenamed('count', 'doc_len')
    return term_freq.join(doc_len, 'id') \
        .withColumn('tf', F.col('term_count') / F.col('doc_len'))

# IDF Function
def idf_mapper(tf_df, n_docs):
    """IDF per term."""
    docs_with_term = (tf_df.select('term', 'id').distinct()
        .groupBy('term').count()
        .withColumnRenamed('count', 'docs_with_term'))
    return docs_with_term.withColumn(
        'idf', F.log(F.lit(n_docs) / F.col('docs_with_term'))
    )

# TF-IDF Function
def tfidf_reduce(tf_df, idf_df):
    """Join TF & IDF → TF-IDF."""
    return (tf_df.join(idf_df.select('term', 'idf'), 'term')
        .withColumn('tfidf', F.col('tf') * F.col('idf'))
        .select('id', 'term', 'tfidf'))

```

```

# Q2
# Calculate tf-idf measure for each row
total_docs = agnews.count()

tf_df = tf_mapper(agnews)
idf_df = idf_mapper(tf_df, total_docs)
tfidf_df = tfidf_reduce(tf_df, idf_df)

tfidf_map_df = (tfidf_df.groupBy('id')
    .agg(F.map_from_entries(
        F.collect_list(F.struct('term', 'tfidf'))
    ).alias('tfidf')))

agnews = agnews.join(tfidf_map_df, 'id') # ← adds the new “tfidf” column

```

```

# Q3
# Print out the tf-idf measure for the first 5 documents.
agnews.select('id', 'tfidf').show(5, truncate=False)

```

```

+---+-----+
|id |tfidf|
+---+-----+
1 |{investment -> 0.1890771769001148, commercial -> 0.2057832028092643, reputation -> 0.2578098186776328, group ->
3 |{exports -> 0.2146590164054526, infrastructure -> 0.22959926718225876, reuters -> 0.15913296762843637, militia
5 |{positive -> 0.18127557126337487, 46 -> 0.2067185029184427, o -> 0.1405921241478995, end -> 0.1131018693698805,
6 |{98 -> 0.24380014644675033, billion -> 0.12463394966614495, money -> 0.32032556569959436, thursday -> 0.0941173
9 |{wall -> 0.48467229409055657, green -> 0.27256812064061997, ultra -> 0.3908380162950787, new -> 0.1003734968095
+---+-----+
only showing top 5 rows

```

Task 2

```
!curl https://raw.githubusercontent.com/mosesyh/de300-2025sp-class/refs/heads/main/w.csv -O
!curl https://raw.githubusercontent.com/mosesyh/de300-2025sp-class/refs/heads/main/bias.csv -O
!curl https://raw.githubusercontent.com/mosesyh/de300-2025sp-class/refs/heads/main/data_for_svm.csv -O
```

```

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %         %         Dload  Upload    Total   Spent    Left   Speed
100 1391 100 1391    0     0   5035      0 --:--:-- --:--:-- --:--:-- 5039
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %         %         Dload  Upload    Total   Spent    Left   Speed
100  22 100  22    0     0    97      0 --:--:-- --:--:-- --:--:--  98
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %         %         Dload  Upload    Total   Spent    Left   Speed
100 61.9M 100 61.9M    0     0 39.7M      0 0:00:01 0:00:01 --:--:-- 39.7M

```

```

# LOAD & PREPARE SVM DATA
from pyspark.sql import functions as F
import pandas as pd
import numpy as np

# Read Data
feature_cols = [f"f{i}" for i in range(64)] + ["label"]
svm_df = (spark.read
          .csv("data_for_svm.csv", header=False, inferSchema=True)
          .toDF(*feature_cols))

# Read w and b
w_vec = pd.read_csv("w.csv", header=None).values.flatten().astype(float) # len = 64
b_val = float(pd.read_csv("bias.csv", header=None).iloc[0, 0])           # scalar

w_bc = spark.sparkContext.broadcast(w_vec)
b_bc = spark.sparkContext.broadcast(b_val)

print(f"Loaded {svm_df.count()} rows")
print(f"w length = {w_vec.shape[0]} (should be 64)")
print(f"b (bias) = {b_val}")

```

```

Loaded 400000 rows
w length = 64 (should be 64)
b (bias) = 0.0001495661647902

```

```
# Q1
# MapReduce functions required to calculate the loss function

def map_hinge(row):
    """
    Map step: for one (x_i, y_i) → (1, hinge_i)
    hinge_i = max(0, 1 - y_i (w·x_i + b))
    Uses the GLOBAL broadcasts w_bc, b_bc defined in Cell A.
    """
    x = np.array([row[f"f{i}"] for i in range(64)], dtype=float)
    y = float(row["label"])
    margin = y * (np.dot(w_bc.value, x) + b_bc.value)
    hinge = max(0.0, 1.0 - margin)
    return (1, hinge)

def red_sum(a, b):
    """Reduce step: add counts and hinge sums."""
    return (a[0] + b[0], a[1] + b[1])
```

```
# Q2
# Create loss function

def loss_SVM(w, b, df, lmbda=1.0):
    """
    Compute  $L(w,b) = \lambda \|w\|^2 + (1/n) \sum \text{hinge}_i$  in pure MapReduce style.
    """
    # fresh broadcasts for the given (w, b)
    w_tmp = spark.sparkContext.broadcast(np.asarray(w, dtype=float))
    b_tmp = spark.sparkContext.broadcast(float(b))

    def hinge_only(row):
        x = np.array([row[f"f{i}"] for i in range(64)], dtype=float)
        y = float(row["label"])
        margin = y * (np.dot(w_tmp.value, x) + b_tmp.value)
        return max(0.0, 1.0 - margin)


    n = df.count()
    hinge_sum = df.rdd.map(hinge_only).sum()

    reg_term = lmbda * float(np.dot(w, w))
    hinge_term = hinge_sum / n

    return reg_term + hinge_term
```

```
# Q3
# Calculating the objective value

loss_value = loss_SVM(w_vec, b_val, svm_df, lmbda=1.0)
print(f"SVM objective L(w,b) = {loss_value:.6f}")
```

 SVM objective $L(w,b) = 1.002940$

```
# Q4
# Design the MapReduce function
def map_predict(row):
    x = np.array([row[f"f{i}"] for i in range(64)], dtype=float)
    score = float(np.dot(w_bc.value, x) + b_bc.value)
    pred = 1 if score >= 0 else -1
    return (pred,)

# Run prediction across the data
pred_rdd = svm_df.rdd.map(map_predict)
```

```
pred_df = pred_rdd.toDF(["y_pred"])
```