4.2 Machine Learning dengan PySpark dan HDFS

4.2.1 ETL Data

```
(base) hduser@hadoop-master:~$ wget https://raw.githubusercontent.com/kmdavidds/datasets/refs/heads/main/heart_disease_uci.csv
--2025-06-10 15:39:50--  https://raw.githubusercontent.com/kmdavidds/datasets/refs/heads/main/heart_disease_uci.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 79346 (77K) [text/plain]
Saving to: 'heart_disease_uci.csv'

heart_disease_uci.csv               100%[===================================================================================>]  77.49K

2025-06-10 15:39:51 (2.02 MB/s) - 'heart_disease_uci.csv' saved [79346/79346]
```

wget https://raw.githubusercontent.com/kmdavidds/datasets/refs/heads/main/heart_disease_uci.csv

Gambar 4.2.1.1: Mengunduh Dataset

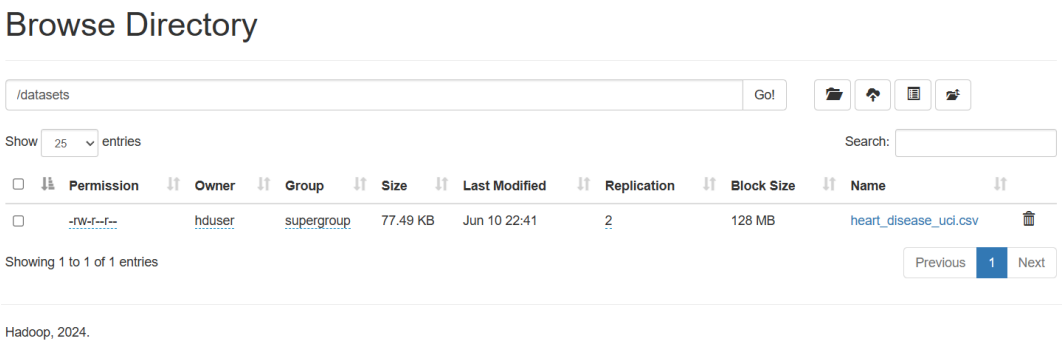
```
(base) hduser@hadoop-master:~$ hdfs dfs -mkdir /datasets
(base) hduser@hadoop-master:~$ hdfs dfs -put ./heart_disease_uci.csv /datasets/heart_disease_uci.csv
(base) hduser@hadoop-master:~$ hdfs dfs -ls /datasets
Found 1 items
-rw-r--r--   2 hduser supergroup      79346 2025-06-10 15:41 /datasets/heart_disease_uci.csv
```

hdfs dfs -mkdir /datasets

hdfs dfs -put ./heart_disease_uci.csv /datasets/heart_disease_uci.csv

hdfs dfs -ls /datasets

Gambar 4.2.1.2: Load Dataset ke HDFS



(in browser there is the dataset visible)

Gambar 4.2.1.3: Check Dataset di HDFS

4.2.2 Preprocessing dan Pipelinenya

```
import findspark
findspark.init()

from pyspark.sql import SparkSession
from pyspark import SparkContext
import pyspark.sql.functions as F
from pyspark.sql.functions import when, col
from pyspark.sql.types import StringType

# Buat Spark Session
spark = SparkSession.builder \
    .appName("JupyterLab-Test") \
    .master("spark://hadoop-master:7077") \
    .getOrCreate()
```

import findspark

findspark.init()

from pyspark.sql import SparkSession

from pyspark import SparkContext

import pyspark.sql.functions as F

from pyspark.sql.functions import when, col

from pyspark.sql.types import StringType

Buat Spark Session

spark = SparkSession.builder \

.appName("JupyterLab-Test") \

.master("spark://hadoop-master:7077") \

.getOrCreate()

Gambar 4.2.2.1: Mulai Spark Session

```
df = spark.read.format("csv").load("hdfs://hadoop-master:9000/datasets/heart_disease_uci.csv", header=True, inferSchema=True)
```

```
df.show(5)
```

```
+---+---+------+---------+---------------+--------+----+-----+--------------+------+-----+-------+-----------+---+-----------------+---+
| id|age|   sex|  dataset|             cp|trestbps|chol|  fbs|       restecg|thalch|exang|oldpeak|      slope| ca|             thal|num|
+---+---+------+---------+---------------+--------+----+-----+--------------+------+-----+-------+-----------+---+-----------------+---+
|  1| 63|  Male|Cleveland| typical angina|     145| 233| true|lv hypertrophy|   150|false|    2.3|downsloping|  0|     fixed defect|  0|
|  2| 67|  Male|Cleveland|   asymptomatic|     160| 286|false|lv hypertrophy|   108| true|    1.5|       flat|  3|           normal|  2|
|  3| 67|  Male|Cleveland|   asymptomatic|     120| 229|false|lv hypertrophy|   129| true|    2.6|       flat|  2|reversable defect|  1|
|  4| 37|  Male|Cleveland|    non-anginal|     130| 250|false|        normal|   187|false|    3.5|downsloping|  0|           normal|  0|
|  5| 41|Female|Cleveland|atypical angina|     130| 204|false|lv hypertrophy|   172|false|    1.4|  upsloping|  0|           normal|  0|
+---+---+------+---------+---------------+--------+----+-----+--------------+------+-----+-------+-----------+---+-----------------+---+
only showing top 5 rows
```

```
df = spark.read.format("csv").load("hdfs://hadoop-master:9000/datasets/heart_disease_uci.csv", header=True, inferSchema=True)

df.show(5)
```

Gambar 4.2.2.2: Load CSV Dataset

```
df.printSchema()

print((df.count(), len(df.columns)))
```

```
root
 |-- id: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- sex: string (nullable = true)
 |-- dataset: string (nullable = true)
 |-- cp: string (nullable = true)
 |-- trestbps: integer (nullable = true)
 |-- chol: integer (nullable = true)
 |-- fbs: boolean (nullable = true)
 |-- restecg: string (nullable = true)
 |-- thalch: integer (nullable = true)
 |-- exang: boolean (nullable = true)
 |-- oldpeak: double (nullable = true)
 |-- slope: string (nullable = true)
 |-- ca: integer (nullable = true)
 |-- thal: string (nullable = true)
 |-- num: integer (nullable = true)

(920, 16)
```

df.printSchema()

print((df.count(), len(df.columns)))

Gambar 4.2.2.3: Check Schema dari Dataset

```
from pyspark.sql.functions import when, count, col, lit, round

total_rows = df.count()

missing_percentage = df.select([
    round((count(when(col(c).isNull(), c)) / lit(total_rows)) * 100, 2).alias(c)
    for c in df.columns
]).show()
```

```
+---+---+---+-------+---+--------+----+----+-------+------+-----+-------+-----+-----+-----+---+
| id|age|sex|dataset| cp|trestbps|chol| fbs|restecg|thalch|exang|oldpeak|slope|   ca| thal|num|
+---+---+---+-------+---+--------+----+----+-------+------+-----+-------+-----+-----+-----+---+
|0.0|0.0|0.0|    0.0|0.0|    6.41|3.26|9.78|   0.22|  5.98| 5.98|   6.74|33.59|66.41|52.83|0.0|
+---+---+---+-------+---+--------+----+----+-------+------+-----+-------+-----+-----+-----+---+
```

```
total_rows = df.count()


missing_percentage = df.select([
    round((count(when(col(c).isNull(), c)) / lit(total_rows)) * 100, 2).alias(c)
    for c in df.columns
]).show()
```

Gambar 4.2.2.4: Check Missing Data secara persentase

```
df = df.withColumn('num', when(col('num') > 0, 1).otherwise(0))
```

df = df.withColumn('num', when(col('num') > 0, 1).otherwise(0))

Gambar 4.2.2.5: Ubah kolom target menjadi binary class (0 atau 1)

```
df = df.withColumn('fbs', col('fbs').cast(StringType()))
df = df.withColumn('exang', col('exang').cast(StringType()))
```

df = df.withColumn('fbs', col('fbs').cast(StringType()))

df = df.withColumn('exang', col('exang').cast(StringType()))

Gambar 4.2.2.6: Ubah tipe data boolean ke string

```
df = df.drop('id', 'dataset')
df = df.drop('ca', 'thal')
```

df = df.drop('id', 'dataset') # id and source dataset not needed for classification

df = df.drop('ca', 'thal') # over 50% missing data

Gambar 4.2.2.7: Hapus kolom irrelevant dan kolom high missing percentage

```
numerical_cols = ['age', 'trestbps', 'chol', 'thalch', 'oldpeak']
categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope']
```

numerical_cols = ['age', 'trestbps', 'chol', 'thalch', 'oldpeak']

categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope']

Gambar 4.2.2.8: Buat list kolom numerical dan categorical

```
train, test = df.randomSplit([0.7, 0.3], seed=42)
```

train, test = df.randomSplit([0.7, 0.3], seed=42)

Gambar 4.2.2.9: Bagi dataset 7:3 dengan seed 42

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import (VectorAssembler, StandardScaler, StringIndexer,
                                OneHotEncoder, Imputer)
from pyspark.ml.classification import LinearSVC, RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
```

from pyspark.ml import Pipeline

from pyspark.ml.feature import (VectorAssembler, StandardScaler, StringIndexer,

OneHotEncoder, Imputer)

from pyspark.ml.classification import LinearSVC, RandomForestClassifier

from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator

Gambar 4.2.2.10: Import bagian library pyspark

```
def create_preprocessing_pipeline():
    stages = []

    imputer = Imputer(
        inputCols=numerical_cols,
        outputCols=numerical_cols,
        strategy="mean"
    )
    stages.append(imputer)

    numerical_assembler = VectorAssembler(
        inputCols=numerical_cols,
        outputCol='numerical_features'
    )
    stages.append(numerical_assembler)

    scaler = StandardScaler(
        inputCol='numerical_features',
        outputCol='scaled_numerical_features',
        withStd=True,
        withMean=True
    )
    stages.append(scaler)
```

```
    indexers = []
    for col_name in categorical_cols:
        indexer = StringIndexer(
            inputCol=col_name,
            outputCol=f"{col_name}_index",
            handleInvalid="keep"
        )
        indexers.append(indexer)
    stages.extend(indexers)

    encoders = []
    for col_name in categorical_cols:
        encoder = OneHotEncoder(
            inputCol=f"{col_name}_index",
            outputCol=f"{col_name}_encoded"
        )
        encoders.append(encoder)
    stages.extend(encoders)
```

```
    categorical_feature_cols = [f"{col}_encoded" for col in categorical_cols]
    categorical_assembler = VectorAssembler(
        inputCols=categorical_feature_cols,
        outputCol='categorical_features'
    )
    stages.append(categorical_assembler)

    final_assembler = VectorAssembler(
        inputCols=['scaled_numerical_features', 'categorical_features'],
        outputCol='features'
    )
    stages.append(final_assembler)

    return Pipeline(stages=stages)
```

def create_preprocessing_pipeline():

stages = []

```
imputer = Imputer(
    inputCols=numerical_cols,
    outputCols=numerical_cols,
    strategy="mean"
)
stages.append(imputer)

numerical_assembler = VectorAssembler(
    inputCols=numerical_cols,
    outputCol='numerical_features'
)
stages.append(numerical_assembler)

scaler = StandardScaler(
    inputCol='numerical_features',
    outputCol='scaled_numerical_features',
    withStd=True,
    withMean=True
)
stages.append(scaler)

indexers = []
for col_name in categorical_cols:
    indexer = StringIndexer(
        inputCol=col_name,
```

```
        outputCol=f"{col_name}_index",
        handleInvalid="keep"
    )
    indexers.append(indexer)
stages.extend(indexers)

encoders = []
for col_name in categorical_cols:
    encoder = OneHotEncoder(
        inputCol=f"{col_name}_index",
        outputCol=f"{col_name}_encoded"
    )
    encoders.append(encoder)
stages.extend(encoders)

categorical_feature_cols = [f"{col}_encoded" for col in categorical_cols]
categorical_assembler = VectorAssembler(
    inputCols=categorical_feature_cols,
    outputCol='categorical_features'
)
stages.append(categorical_assembler)

final_assembler = VectorAssembler(
    inputCols=['scaled_numerical_features', 'categorical_features'],
    outputCol='features'
)
```

stages.append(final_assembler)

return Pipeline(stages=stages)

Gambar 4.2.2.11: Pipeline Preprocessing

4.2.3 Pembuatan Model dan Pipelinenya

```
def create_model_pipeline(model_type):
    preprocessing_pipeline = create_preprocessing_pipeline()

    stages = preprocessing_pipeline.getStages().copy()

    if model_type == 'svm':
        model = LinearSVC(
            featuresCol="features",
            labelCol="num",
            maxIter=100
        )
    elif model_type == 'rf':
        model = RandomForestClassifier(
            featuresCol="features",
            labelCol="num",
            numTrees=100,
            seed=42
        )
    else:
        raise ValueError("model_type must be 'svm' or 'rf'")

    stages.append(model)

    return Pipeline(stages=stages)
```

def create_model_pipeline(model_type):

preprocessing_pipeline = create_preprocessing_pipeline()

stages = preprocessing_pipeline.getStages().copy()

```
if model_type == 'svm':
    model = LinearSVC(
        featuresCol="features",
        labelCol="num",
        maxIter=100
    )
elif model_type == 'rf':
    model = RandomForestClassifier(
        featuresCol="features",
        labelCol="num",
        numTrees=100,
        seed=42
    )
else:
    raise ValueError("model_type must be 'svm' or 'rf'")


stages.append(model)


return Pipeline(stages=stages)
```

Gambar 4.2.3.1: Pipeline Model

```
svm_pipeline = create_model_pipeline('svm')
svm_model = svm_pipeline.fit(train)
svm_predictions = svm_model.transform(test)
```

```
25/06/10 17:01:16 WARN InstanceBuilder: Failed
```

svm_pipeline = create_model_pipeline('svm')

svm_model = svm_pipeline.fit(train)

svm_predictions = svm_model.transform(test)

Gambar 4.2.3.2: Pembuatan Model SVM

```
rf_pipeline = create_model_pipeline('rf')
rf_model = rf_pipeline.fit(train)
rf_predictions = rf_model.transform(test)
```

rf_pipeline = create_model_pipeline('rf')

rf_model = rf_pipeline.fit(train)

rf_predictions = rf_model.transform(test)

Gambar 4.2.3.3: Pembuatan Model Random Forest

4.2.4 Evaluasi Model

```
def evaluate_model(predictions):
    multiclass_metrics = ['accuracy', 'weightedPrecision', 'weightedRecall', 'f1']
    binary_metrics = ['areaUnderROC', 'areaUnderPR']

    results = {}

    for metric in multiclass_metrics:
        evaluator = MulticlassClassificationEvaluator(
            labelCol="num",
            predictionCol="prediction",
            metricName=metric
        )
        results[metric] = evaluator.evaluate(predictions)

    for metric in binary_metrics:
        evaluator = BinaryClassificationEvaluator(
            labelCol="num",
            rawPredictionCol="rawPrediction",
            metricName=metric
        )
        results[metric] = evaluator.evaluate(predictions)

    return results
```

Gambar 4.2.4.1: Fungsi Evaluasi Model

```
print("SVM Evaluation:")
svm_results = evaluate_model(svm_predictions)
for metric, value in svm_results.items():
    print(f"{metric}: {value:.4f}")
```

```
SVM Evaluation:

accuracy: 0.8186
weightedPrecision: 0.8184
weightedRecall: 0.8186
f1: 0.8185
areaUnderROC: 0.8980
areaUnderPR: 0.9089
```

print("SVM Evaluation:")

svm_results = evaluate_model(svm_predictions)

for metric, value in svm_results.items():

print(f"{metric}: {value:.4f}")

Gambar 4.2.4.1: Hasil Evaluasi Model SVM

```
print("Random Forest Evaluation:")
rf_results = evaluate_model(rf_predictions)
for metric, value in rf_results.items():
    print(f"{metric}: {value:.4f}")
```

```
Random Forest Evaluation:

accuracy: 0.8228
weightedPrecision: 0.8226
weightedRecall: 0.8228
f1: 0.8226
areaUnderROC: 0.9009
areaUnderPR: 0.9112
```

print("Random Forest Evaluation:")

rf_results = evaluate_model(rf_predictions)

for metric, value in rf_results.items():

print(f"{metric}: {value:.4f}")

Gambar 4.2.4.2: Hasil Evaluasi Model Random Forest