Practical 4

#load csv file

df\_iris = spark.read.format("csv").option("multiline", True).option("header",

True).option("inferschema", True).load("/FileStore/tables/Iris-1.csv")

df\_iris.show()

df\_iris.printSchema()

df\_iris.count()

#to print the summary of the columns

df\_iris.summary().show()

#select specific columns

df\_iris.select("SepalLengthCm","PetalLengthCm").show()

df\_iris.createOrReplaceTempView("irisTab")

df\_iris1 = spark.sql("select \* from irisTab where PetalLengthCm<2.5")

df\_iris1.show()

df\_iris1.count()

df\_iris2 = spark.sql("select \* from irisTab where PetalLengthCm between 2.5 and 4.5")

df\_iris2.show()

df\_iris2.count()

#spark query to see the count of each species

df\_iris.groupby("species").count().show()

df\_iris1=spark.sql("Select species, count(\*) from iristab group by species")

df\_iris1.show()

#get the sepal length and sepal width of the species iris-setosa

df\_iris2=spark.sql("select SepalLengthCm, SepalWidthCm,species from irisTab where species=\"Iris-setosa\"")

df\_iris2.show()

df\_iris2 = spark.sql("select species, min(SepalLengthCm) as min\_sepal\_length from irisTab group by species")

df\_iris2.show()

#find out the average petal length of each species in the iris dataset

df\_iris.groupby("species").mean("PetalLengthCm").show()

#to run multiple aggregates using spark

spark.sql("select species,min(PetalLengthCm) as min\_petal\_len,max(PetalLengthCm) as max\_petal\_len,avg(PetalLengthCm) as avg\_petal\_len from irisTab group by species").show()

#print the petal length and petal width and species where the averagepetal length is more than 4

spark.sql("select species,PetalLengthCm,PetalWidthCm from irisTab group by species,PetalLengthCm,PetalWidthCm having mean(PetalLengthCm)>4.0").show()

#to print the data using a spark command where species iris setosa

df\_iris.where("species==\"Iris-setosa\"").show()

df\_iris2 = df\_iris.withColumnRenamed("PetalLengthCm","petal\_len")

df\_iris2.show()

#conver iris spark dataset to pandas

pd\_iris = df\_iris.toPandas()

pd\_iris.head()

pd\_iris.dtypes

Practical 5

rdd\_iris1 = sc.textFile("/FileStore/tables/Iris-1.csv",4)

rdd\_iris1.collect()

rdd\_iris1.getNumPartitions()

rdd\_iris1.glom().collect()[0]

len(rdd\_iris1.glom().collect()[0])

rdd\_iris1.glom().collect()[1]

len(rdd\_iris1.glom().collect()[1])

len(rdd\_iris1.glom().collect()[2])

len(rdd\_iris1.glom().collect()[3])

rdd\_iris2 = sc.parallelize(rdd\_iris1.collect(),8)

rdd\_iris2.getNumPartitions()

for i in rdd\_iris1.collect():print(i)

rdd\_iris3=rdd\_iris2.map(lambda x:x\*2)

rdd\_iris2.collect()

rdd\_iris3.collect()

rdd\_iris4 = rdd\_iris2.flatMap(lambda x:x.split(","))

rdd\_iris4.collect()

rdd\_iris4.getNumPartitions()

rdd\_iris4.glom().collect()[0]

rdd\_iris4.glom().collect()[1]

rdd\_iris5 = rdd\_iris4.map(lambda x:x\*2)

rdd\_iris5.collect()

## Filtering

rdd\_iris6 = rdd\_iris1.flatMap(lambda x:x.split(",")).filter(lambda x:x=="Iris virginica").map(lambda x:(x,1))

rdd\_iris6.glom().collect()[1]

rdd\_species = rdd\_iris1.flatMap(lambda x:x.split(",")).filter(lambda x:(x=="Iris-virginica" or x=="Iris-setosa" or x=="Iris-versicolor"))

rdd\_species.collect()

rdd\_species1 = rdd\_species.groupBy(lambda x:x).mapValues(lambda x:len(x))

rdd\_species1.collect()

Practical 6

import numpy as np

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score,classification\_report

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import train\_test\_split

df\_train = pd.read\_csv('/home/sumon/Documents/Datasets/titan\_train.csv')

df\_test = pd.read\_csv('/home/sumon/Documents/Datasets/titan\_test.csv')

df\_train.head()

df\_test.head()

df\_train.info()

df\_test.info()

df\_train.isna().sum()

df\_test.isna().sum()

def cleaning\_data(df):

m=np.mean(df['Age'])

df['Age'].fillna(m,inplace=True)

df['Embarked'].fillna('S',inplace=True)

df.drop(['Ticket','Name','Cabin','PassengerId'],axis=1,inplace=True)

m\_fare = np.median(df[df['Fare'].isna()==False]['Fare'])

df['Fare'].fillna(m\_fare,inplace=True)

return df

def encoding\_data(df):

df = pd.get\_dummies(data=df,columns=['Sex','Embarked'])

return df

df\_cleaned\_tr = cleaning\_data(df\_train)

df\_cleaned\_test = cleaning\_data(df\_test)

df\_encode\_tr = encoding\_data(df\_cleaned\_tr)

df\_encode\_test = encoding\_data(df\_cleaned\_test)

logit\_p1 = Pipeline([('scaling',StandardScaler()),

('pca',PCA(n\_components=3)),

('model\_logit',LogisticRegression())])

dtree\_p1 = Pipeline([('scaling',StandardScaler()),

('pca',PCA(n\_components=3)),

('model\_dtree',DecisionTreeClassifier())])

naive\_p1 = Pipeline([('scaling',StandardScaler()),

('pca',PCA(n\_components=3)),

('model\_bayes',GaussianNB())])

x = df\_encode\_tr.drop('Survived',axis=1)

y = df\_encode\_tr['Survived']

x\_tr,x\_test,y\_tr,y\_test = train\_test\_split(x,y,test\_size=0.2,random\_state=100)

my\_pipeline = [logit\_p1,dtree\_p1,naive\_p1]

pipeline\_dict = {0:'Logistic\_Regression',1:'Decision\_Tree',2:'Naive\_Bayes'}

for i in my\_pipeline:

i.fit(x\_tr,y\_tr)

for i,model in enumerate(my\_pipeline):

print(f"{pipeline\_dict[i]}'s training accuracy is : {model.score(x\_tr,y\_tr)}")

for i,model in enumerate(my\_pipeline):

print(f"{pipeline\_dict[i]}'s testing accuracy is : {model.score(x\_test,y\_test)}")

Practical 7

from pyspark.ml import Pipeline

from pyspark.ml.feature import StringIndexer,OneHotEncoder,VectorAssembler

df\_ipl = spark.read.option("inferSchema",True).option("header",True).csv("/FileStore/tables/ind\_ban\_comment.csv")

df\_ipl.show()

df\_ipl.printSchema()

df\_ipl\_mod = StringIndexer(inputCol="Batsman\_Name",outputCol="bt\_idx").fit(df\_ipl).transform(df\_ipl)

df\_ipl\_mod.select("Batsman\_name","bt\_idx").show()

df\_ipl.columns

df\_ipl = df\_ipl.drop(\*["Batsman","Bowler","Id"])

stage2 = StringIndexer(inputCol='Batsman\_Name',outputCol='bm\_index')

stage3 = StringIndexer(inputCol='Bowler\_Name',outputCol='bowler\_index')

stage4 = StringIndexer(inputCols=['bm\_index','bowler\_index'],outputCols=['bm\_ohe','bowler\_ohe'])

stage5 = VectorAssembler(inputCols=['Dismissed','Isball','Isboundary','Over','Runs','bm\_ohe','bowler\_ohe'],outputCol="features")

from pyspark.ml.classification import LogisticRegression

stage6 = LogisticRegression(featuresCol="features",labelCol="Iswicket")

pl = Pipeline(stages=[stage2,stage3,stage4,stage5,stage6])

df\_ipl = df\_ipl.fillna(0)

logit\_model = pl.fit(df\_ipl).transform(df\_ipl)

logit\_model.select("features","Iswicket","prediction").show()

logit\_model.columns

pl1 = Pipeline(stages=[stage2,stage3,stage4])

df\_mod = pl1.fit(df\_ipl).transform(df\_ipl)

df\_mod.show()

df\_vec = VectorAssembler(inputCols=['Dismissed','Isball','Isboundary','Over','Runs','bm\_ohe','bowler\_ohe'],outputCol="features").transform(df\_mod)

df\_vector = df\_vec.select("features","Iswicket")

splits = df\_vector.randomSplit([.7,.3])

df\_train = splits[0]

df\_test = splits[1]

df\_train.show(10)

logit\_mod\_tr = LogisticRegression(featuresCol="features",labelCol="Iswicket").fit(df\_train).transform(df\_train)

logit\_mod\_test = LogisticRegression(featuresCol="features",labelCol="Iswicket").fit(df\_train).transform(df\_test)

logit\_mod\_test.select(["features","Iswicket","prediction"]).show()

from pyspark.ml.evaluation import BinaryClassificationEvaluator

eval = BinaryClassificationEvaluator(labelCol="Iswicket",rawPredictionCol="prediction",metricName="areaUnderROC")

eval.evaluate(logit\_mod\_test)

Practical 8

import pyspark

import pandas as pd

from pyspark import SparkContext,SparkConf

from pyspark.sql import SparkSession

spark\_new = SparkSession.builder.master("local[2]").getOrCreate()

data = spark\_new.read.option("header",True).option("inferSchema",True).csv("/FileStore/tables/USA\_cars\_datasets.csv")

data.show()

data.printSchema()

df\_panda=data.toPandas()

df\_panda.dtypes

df\_panda.columns

df\_panda.isna().sum()

import seaborn as sns

sns.heatmap(df\_panda.corr(), annot=True)

from pyspark.ml.feature import OneHotEncoder

from pyspark.ml.feature import StringIndexer

string\_index=StringIndexer(inputCols=['brand','model','color','state'],outputCols=['brand\_idx','model\_idx','color\_idx','state\_idx'])

data = string\_index.fit(data).transform(data)

data.show()

ohe\_mod=OneHotEncoder(inputCols=['brand\_idx','model\_idx','color\_idx','state\_idx'],outputCols=['brand\_ohe','model\_ohe','color\_ohe','state\_ohe'])

data = ohe\_mod.fit(data).transform(data)

from pyspark.ml.feature import VectorAssembler

v1 = VectorAssembler(inputCols=['brand\_ohe','color\_ohe','state\_ohe','year', 'mileage', 'lot'], outputCol="feature\_vector")

data\_new = v1.transform(data)

data\_new.show()

data\_new.select("feature\_vector").show()

from pyspark.ml.regression import RandomForestRegressor

rf\_mod = RandomForestRegressor(featuresCol="feature\_vector",labelCol="price")

#cross\_validation tuning

from pyspark.ml.tuning import ParamGridBuilder

pm=ParamGridBuilder().addGrid(rf\_mod.numTrees,[100,1500]).build()

from pyspark.ml.tuning import CrossValidator

from pyspark.ml.evaluation import RegressionEvaluator

cf = CrossValidator(estimator = rf\_mod,

estimatorParamMaps = pm,

evaluator = RegressionEvaluator(labelCol = "price"),

numFolds = 3)

#splitting of the data into train and test set

tr\_data,test\_data = data\_new.randomSplit([.75,.25],seed=100)

data\_new.show()

cvmod = cf.fit(tr\_data)

bestModel = cvmod.bestModel

bestModel

pred=cvmod.transform(test\_data)

pred.show()

pred\_mod = pred.select("price","prediction")

pred\_mod.show()

pred = pred.withColumnRenamed("price","label")

pred.show()

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator()

eval\_score = evaluator.evaluate(pred,{evaluator.metricName:'r2'})

eval\_score

Practical 9

import pyspark

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.evaluation import ClusteringEvaluator

import pandas as pd

import seaborn as sns

uberdf\_1 = spark.read.option("header",True).option("inferSchema",True).csv("/FileStore/tables/uber\_data.csv")

uberdf\_1.show()

from pyspark.sql.functions import \*

uberdf\_1.printSchema()

spark.sql("set spark.sql.legacy.timeParserPolicy=LEGACY")

uberdf\_2 = uberdf\_1.withColumn("date\_new", to\_date(col("Date/Time"), "MM/dd/yyyy HH:mm:ss"))

uberdf\_2.show()

from pyspark.sql.functions import month,dayofweek,year

uberdf\_3 = uberdf\_2.withColumn("year", year(col("date\_new")))

uberdf\_3.show()

uberdf\_3.printSchema()

input\_features = ["Lat","Lon"]

va = VectorAssembler(inputCols=input\_features,outputCol="features")

uberdf\_3 = va.transform(uberdf\_3)

uberdf\_3.show()

df = uberdf\_2.toPandas()

df.drop(["Base","Date/Time","date\_new"], axis=1, inplace=True)

df

df.head(20)

from sklearn.cluster import KMeans

wcss=[]

for k in range(1,11):

kmod=KMeans(n\_clusters=k, init="random").fit(df)

wcss.append(kmod.inertia\_)

import matplotlib.pyplot as plt

plt.plot(range(1,11), wcss, marker="o", c="blue", markerfacecolor="red")

training, testing = uberdf\_3.randomSplit([.75,.25])

training.show()

from pyspark.ml.clustering import KMeans

kmod = KMeans(k=3, initMode="k-means||", featuresCol="features", predictionCol="prediction")

kmod = kmod.fit(training)

p = kmod.transform(testing)

p.show()

pl = p.toPandas()

pl["prediction"].value\_counts()

pl.shape

p.createOrReplaceTempView("uber")

df=spark.sql("select prediction, count(\*) from uber group by prediction")

df.show()

Practical 10

!pip install keras

!pip install tensorflow

!pip install elephas

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

from pyspark.ml import Pipeline

from pyspark.mllib.evaluation import MulticlassMetrics

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation

from keras import optimizers, regularizers

from keras.optimizers import Adam

from elephas.ml\_model import ElephasEstimator

df = spark.read.option("inferSchema", True).option("header", True).csv("/FileStore/tables/bank\_new.csv")

df.show(10)

df.printSchema()

df1 = df.toPandas()

dt = df1.dtypes

numeric = dt[dt=='int32'].index

categorical = dt[dt=='object'].index

categorical

cat\_features = dt[dt=="object"].index

cat\_features

num\_features = dt[dt=="int32"].index

num\_features

si = StringIndexer(inputCols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'deposit'], outputCols = ['job\_idx', 'marital\_idx', 'education\_idx', 'default\_idx', 'housing\_idx', 'loan\_idx', 'contact\_idx', 'month\_idx', 'poutcome\_idx', 'label'])

si

df\_si = si.fit(df)

df\_si = df\_si.transform(df)

df\_si.columns

ohe = OneHotEncoder(inputCols = ['job\_idx', 'marital\_idx', 'default\_idx', 'housing\_idx', 'loan\_idx', 'contact\_idx', 'month\_idx', 'poutcome\_idx'], outputCols = ['job\_ohe', 'marital\_ohe', 'default\_ohe', 'housing\_ohe', 'loan\_ohe', 'contact\_ohe', 'month\_ohe', 'poutcome\_ohe'])

df\_si = si.fit(df).transform(df)

df\_si.show()

df\_ohe = ohe.fit(df\_si).transform(df\_si)

df\_ohe.select("job\_ohe", "marital\_ohe").show()

va = VectorAssembler(inputCols = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous', 'job\_ohe', 'marital\_ohe', 'default\_ohe', 'housing\_ohe', 'loan\_ohe', 'contact\_ohe', 'month\_ohe', 'poutcome\_ohe'], outputCol="features")

df\_new = va.transform(df\_ohe)

d1 = df\_new.select("features").toPandas()

d1.head()

from pyspark.sql.functions import rand

df\_new = df\_new.select("features", "label")

df\_new = df\_new.orderBy(rand())

df\_new.show()

df2 = df\_new.toPandas()

df2.head()

df\_new.select("features").show(5)

df\_tr, df\_test = df\_new.randomSplit([.8, .2])

df\_tr.show()

n\_classes = df\_tr.select("label").distinct().count()

n\_classes

input\_dim = len(df\_tr.select("features").first()[0])

input\_dim

model = Sequential()

model.add(Dense(64, input\_shape=(input\_dim,), activity\_regularizer = regularizers.l2(0.01)))

model.add(Activation('relu'))

model.add(Dropout(rate=0.3))

model.add(Dense(64, activity\_regularizer=regularizers.l2(0.01)))

model.add(Activation('relu'))

model.add(Dropout(rate=0.3))

model.add(Dense(n\_classes))

model.add(Activation('sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam')

optimizer\_conf = optimizers.Adam(learning\_rate=0.01)

opt\_conf = optimizers.serialize(optimizer\_conf)

#Initialize SparkML Estimator and get Settings

estimator = ElephasEstimator()

estimator.setFeaturesCol("features")

estimator.setLabelCol("label")

estimator.set\_keras\_model\_config(model.to\_json())

estimator.set\_categorical\_labels(True)

estimator.set\_nb\_classes(n\_classes)

estimator.set\_num\_workers(1)

estimator.set\_epochs(25)

estimator.set\_batch\_size(64)

estimator.set\_verbosity(1)

estimator.set\_validation\_split(0.10)

estimator.set\_optimizer\_config(opt\_conf)

estimator.set\_mode("synchronous")

estimator.set\_loss("binary\_crossentropy")

estimator.set\_metrics(['acc'])

dl\_pipeline = Pipeline(stages=[estimator])

fit\_pipeline = dl\_pipeline.fit(df\_new)

df\_new.printSchema()

pred\_train = fit\_pipeline.transform(df\_new)

pred\_train.show()

pnl\_train = pred\_train.select("label", "prediction")

pnl\_train.printSchema()

pnl\_train.show()

import numpy as np

df = pnl\_train.toPandas()

li\_pred = []

for i in df["prediction"]:

li\_pred.append(np.argmax(i))

df["pred\_new"] = li\_pred

df.head()

from sklearn.metrics import confusion\_matrix, accuracy\_score

print(confusion\_matrix(df["label"], df["pred\_new"]))

accuracy\_score(df["label"], df["pred\_new"])