

Click to verify




```
>>> from pyspark.ml.linalg import Vectors>>> df = spark.createDataFrame([... (1.0, Vectors.dense(1.0)),... (0.0, Vectors.sparse(1, [1, 1])), ["label", "features"])]>>> dt = DecisionTreeRegressor(maxDepth=2)>>> dt.setVarianCol("variance")DecisionTreeRegressor...>>> model = dt.fit(df)>>> model.getVarianceCol()'variance'>>>
model.setLeafCol('leafId')DecisionTreeRegressionModel...>>> model.depth1>>> model.numNodes3>>> model.featureImportancesSparseVector(1, [0: 1.0])>>> model.numFeatures1>>> test0 = spark.createDataFrame([Vectors.dense(1.0)], ["features"])>>> model.predict(test0.head(),features)0.0>>> result =
model.transform(test0).head()>>> result.pdiction0.0>>> model.predictLeaf(test0.head(),features)0.0>>> result.leafId0.0>>> test1 = spark.createDataFrame([Vectors.sparse(1, [0], [1, 0]),], ["features"])>>> model.transform(test1).head(),prediction1.0>>> dttr path = temp_path + "/dtr">>> dt.save(dttr_path)>>> dt2 =
DecisionTreeRegressor.load(dttr_path)>>> dt2.getMaxDepth(2)>>> model_path = temp_path + "/dtr_model">>> model.save(model_path)>>> model2 = DecisionTreeRegressionModel.load(model_path)>>> model.numNodes == model2.numNodesTrue>>> model.depth == model2.depthTrue>>> model.transform(test1).head(),variance0.0>>>
model.transform(test0).take(1) == model2.transform(test0).take(1)True>>> df3 = spark.createDataFrame([... (1.0, 0.2, Vectors.dense(1.0)),... (1.0, 0.8, Vectors.dense(1.0)),... (0.0, 1.0, Vectors.sparse(1, [1, 1])), ["label", "weight", "features"])]>>> dt3 = DecisionTreeRegressor(maxDepth=2, weightCol="weight", varianceCol="variance")>>> model3 =
findspark.init()/findspark.find()import pyspark.find()The next step is to create a Spark session.import matplotlib.pyplot as pltimport numpy as npimport pandas as pdfrom numpy import polyfitfrom pyspark.sql import SparkSessionfrom pyspark import SparkContext,SparkConf,SQLContextconf =
SparkConf().setMaster('local').setAppName('ML_learning')sc = SparkContext(conf=conf)sqlContext = SQLContext(sc)Loading the data which is a csv file in this case.data = sqlContext.read.csv(path='C:\AAI\week 3\Small_Car_Data.csv', header = True, inferSchema = True)data.show((small cars dataset)Now, import the required modules for
implementing linear regression.from pyspark.ml.feature import VectorAssemblerfrom pyspark.ml.regression import LinearRegressionfrom pyspark.ml.evaluation import RegressionEvaluatorInitially performing linear regression on single variable and plotting the result for better understanding.Splitting the data for training and testing purposes.data2
=data.select(data.Displacement,data.Horsepower.alias('label'))train_test = data2.randomSplit([0.9,0.1])Vector Assembler is used for creating a vector of all the individual features.assembler=VectorAssembler().setInputCols([Displacement',]).setOutputCol('label')train01 = assembler.transform(train01) = assembler.transform(train02) =
train01.select("features","label")train02.show(5)Features: Vector of all the individual features.Label: Target Variable.In this case since initially we are building a uni variate linear regression model, only one feature Displacement is converted to a vector.Now, building the model.lr = LinearRegression(model = lr.fit(train02)Testing our model against
unseen data.test01 = assembler.transform(test02)test02 = test01.select('features', 'label')test03 = model.transform(test02)test03.show(5)Plotting the linear fit line.import chart_studio.plotly as pyimport plotly.graph_objects as gofig = go.Figure()fig.add_trace(go.Scatter(x=x, y=y, mode='markers', name='Original Test', ))fig.add_trace(go.Scatter(x=x,
y=y_pred, name='Predicted' ))fig.update_layout(title="Linear Regression", axis_title="Displacement", yaxi_title="Horse Power", font=dict(family="Courier New, monospace", size=18, color="#7f7f7f" ))fig.show()Evaluation of model.evaluator = RegressionEvaluator(print(evaluator.evaluate(test03,(evaluator.metricName:
"r2"))))print(evaluator.evaluate(test03,{evaluator.metricName: "mse"}))print(evaluator.evaluate(test03,{evaluator.metricName: "rmse"}))print(evaluator.evaluate(test03,{evaluator.metricName: "mae"}))Now, lets start building multivariate Linear Regression and Decision Tree model.Selecting the columns relevant to model
building_small_cars=data.select(data._c0,data.Cylinders,data.Displacement,data.Manufacturer,data.Manufacturer,data.Horsepower,data.Acceleration)small_cars.show()Before building the model, some pre-processing of the dataset is required. The variables such as ORIGIN and MANUFACTURER have categorical values which
needs to be converted.from pyspark.ml.feature import StringIndexerindexer = StringIndexer(inputCol="Origin", outputCol="Origin_cat")indexed = indexer.fit(small_cars).transform(small_cars)origin_cat=indexed.select(indexed._c0,indexed.Origin_cat)indexer_1 = StringIndexer(inputCol="Manufacturer", outputCol="Man_cat")indexed_1 =
indexer_1.fit(small_cars).transform(small_cars)man_cat=indexed_1.select(indexed_1._c0,indexed_1.Manufacturer, 'Origin')inner_join_1.show(5)Splitting the data for
training and testing.train_test = inner_join_1.randomSplit([0.9,0.1])Using Vector Assembler to create a vector for all the individual features.assembler=VectorAssembler().setInputCols(['Displacement', 'Cylinders', 'Model_Year', 'Weight', 'Man_cat', 'Origin_cat'])\setOutputCol('features')train_a =
assembler.transform(train)train_b=train_a.select("features",train_a.Horsepower.alias('label'))train_b.show(truncate=False)lr = LinearRegression(model = lr.fit(train_b)test_a = assembler.transform(test)test_b = test_a.select('features', test_a.Horsepower.alias('label'))test_c = model.transform(test_b)test_c.show(truncate=False)Evaluating the
model.evaluator = RegressionEvaluator(print(evaluator.evaluate(test_c,{evaluator.metricName: "mse"}))print(evaluator.evaluate(test_c,{evaluator.metricName: "rmse"}))print(evaluator.evaluate(test_c,{evaluator.metricName: "mae"}))Building a Decision Tree Model.from
pyspark.ml.regression import DecisionTreeRegressordt = DecisionTreeRegressor(model = dt.fit(train_b)test_dt = model.transform(test_b)test_dt.show(truncate=False)Evaluating the model.evaluator = RegressionEvaluator(print(evaluator.evaluate(test_dt,(evaluator.metricName: "r2"))))print(evaluator.evaluate(test_dt,(evaluator.metricName:
"mse"))print(evaluator.evaluate(test_dt,(evaluator.metricName: "rmse"))print(evaluator.evaluate(test_dt,(evaluator.metricName: "mae"))))This is how to build a simple Linear Regression and Decision Tree model in PySpark.Thanks PySpark MLlib library provides a GBRegressor model to implement gradient-boosted tree regression method.
Gradient tree boosting is an ensemble of decision trees model to solve regression and classification tasks in machine learning. Improving the weak learners by different set of train data is the main concept of this model. In this tutorial, we'll briefly learn how to fit and predict regression data by using PySpark GBRegressor in Python. The tutorial
covers:Preparing the dataPrediction and accuracy checkVisualizing the results Source code listing We'll start by loading the required libraries from pyspark import SparkContextfrom pyspark.sql import SQLContextfrom pyspark.ml.feature import VectorAssemblerfrom pyspark.ml.regression import GBRegressorfrom
pyspark.ml.evaluation import RegressionEvaluatorimport matplotlib.pyplot as pltfrom sklearn.datasets import load_bostonimport pandas as pdPreparing the data We use Boston Housing Price dataset as a target regression data and we can easily load it from sklearn.datasets module. Below code shows how to load dataset and transform it into the
pandas data frame type.boston = load_boston()df_boston = pd.DataFrame(boston.data,columns=boston.feature_names)df_boston['target'] = pd.Series(boston.target)print(df_boston.head())Next, we'll define SqlConext and create data frame by using df_boston.data.sc = SparkContext().getOrCreate(sqlContext = SQLContext(sc) data =
sqlContext.createDataFrame(df_boston)print(data.printSchema())root |-- CRIM: double (nullable = true) |-- ZN: double (nullable = true) |-- INDUS: double (nullable = true) |-- NOX: double (nullable = true) |-- RM: double (nullable = true) |-- AGE: double (nullable = true) |-- DIS: double (nullable = true) |-- RAD: double
(nullable = true) |-- TAX: double (nullable = true) |-- PTRATIO: double (nullable = true) |-- B: double (nullable = true) |-- LSTAT: double (nullable = true) |-- target: double (nullable = true)To combine all feature data and separate 'label' data in a dataset, we use VectorAssembler.features = boston.feature_names.tolist() va =
VectorAssembler(inputCols=features, outputCol='features') va.df = va.transform(data)va.df = va.df.select(['features', 'target'])va.df.show(3)+-----+ features|target|+-----+ [0.00632,18.0,2.3,...] 24.0][[0.02731,0.0,7.07...] 21.6][[0.02729,0.0,7.07...] 34.7]+-----+only showing top 3 rowsNext, we'll split data
into the train and test parts.(train_test) = va.df.randomSplit([0.8, 0.2])Prediction and Accuracy Check Next, we'll define the regressor model by using the GBRegressorclass. Here, we can change the parameters according to data content.gbtr = GBRegressor(featuresCol='features', labelCol='target', maxIter=10)gbtr = gbtr.fit(train) After training
the model, we'll predict test data and check the accuracy metrics.mdata = gbtr.transform(test)mdata.show(3)rmse=RegressionEvaluator(labelCol="target", predictionCol="prediction", metricName="rmse")rmse=rmse.evaluate(mdata)mae=RegressionEvaluator(labelCol="target", predictionCol="prediction",
metricName="mae")mae=mae.evaluate(mdata)r2=RegressionEvaluator(labelCol="target", predictionCol="prediction", metricName="r2")r2=r2.evaluate(mdata)print("RMSE: ", rmse)print("MAE: ", mae)print("R-squared: ", r2)+-----+ features|target| prediction|+-----+ [0.02729,0.0,7.07...] 34.7]
33.78460260004924][0.09744,0.0,5.96...] 20.0][21.706837297006956][0.80271,0.0,8.14...] 20.2][18.51482812579027]+-----+only showing top 3 rowsRMSE: 4.41607675529045MAE: 2.826016527055411R-squared: 0.7808863473079463 To visualize the original and predicted data, we can use 'matplotlib' library. We'll
extract those data from the 'mdata' object.x_ax = range(0, mdata.count())y_pred=mdata.select("prediction").collect() plt.plot(x_ax, y_orig, label='original')plt.plot(x_ax, y_pred, label='predicted')plt.title("Boston test and predicted data")plt.xlabel('X-axis')plt.ylabel('Y-axis')plt.legend(loc='best',fancybox=True,
shadow=True)plt.grid(True)plt.show() If you do new executions of your code, do not forget to close the spark context session. In this tutorial, we've briefly learned how to fit and predict regression data by using PySpark GBRegressor model in Python. The full source code is listed below. Source code listingfrom pyspark import SparkContextfrom
pyspark.sql import SQLContextfrom pyspark.ml.feature import VectorAssemblerfrom pyspark.ml.regression import GBRegressorfrom pyspark.ml.evaluation import RegressionEvaluatorimport matplotlib.pyplot as pltfrom sklearn.datasets import load_bostonimport pandas as pd boston = load_boston()df_boston =
pd.DataFrame(boston.data,columns=boston.feature_names)df_boston['target'] = pd.Series(boston.target)print(df_boston.head()) sc = SparkContext().getOrCreate(sqlContext = SQLContext(sc) data = sqlContext.createDataFrame(df_boston)print(data.printSchema()) features = boston.feature_names.tolist() va = VectorAssembler(inputCols =
features, outputCol='features') va.df = va.transform(data)va.df = va.df.select(['features', 'target'])va.df.show(3) (train_test) = va.df.randomSplit([0.8, 0.2]) gbtr = GBRegressor(featuresCol='features', labelCol='target', maxIter=10)gbtr = gbtr.fit(train) mdata = gbtr.transform(test)mdata.show(3) rmse=RegressionEvaluator(labelCol="target", predictionCol="prediction",
metricName="rmse")rmse=rmse.evaluate(mdata)mae=RegressionEvaluator(labelCol="target", predictionCol="prediction", metricName="mae")mae=mae.evaluate(mdata)r2=RegressionEvaluator(labelCol="target", predictionCol="prediction", metricName="r2")r2=r2.evaluate(mdata) print("RMSE: ", rmse)print("MAE: ",
mae)print("R-squared: ", r2) x_ax = range(0, mdata.count()) y_pred=mdata.select("prediction").collect() y_orig=mdata.select("target").collect() plt.plot(x_ax, y_orig, label="original")plt.plot(x_ax, y_pred, label="predicted")plt.title("Boston test and predicted data")plt.xlabel('X-axis')plt.ylabel('Y-axis')plt.legend(loc='best',fancybox=True,
shadow=True)plt.grid(True)plt.show() sc.stop()References:PySpark Gradient-boosted tree regression PySpark MLlib API provides a DecisionTreeClassifier model to implement classification with decision tree method. A decision tree method is one of the well known and powerful supervised machine learning algorithms that can be used for
classification and regression tasks. It is a tree-like, top-down flow learning method to extract rules from the training data. The branches of the tree are based on certain decision outcomes. In this tutorial, we'll briefly learn how to fit and classify data by using PySpark DecisionTreeClassifier. The tutorial covers:Preparing the dataPrediction and
accuracy checkSource code listing We'll start by loading the required libraries for this tutorial.from pyspark import SparkContextfrom pyspark.sql import SQLContextfrom pyspark.ml.classification import DecisionTreeClassifierfrom pyspark.ml.evaluation import MulticlassClassificationEvaluatorfrom pyspark.ml.feature import VectorAssemblerfrom
sklearn.metrics import confusion_matrixfrom sklearn.datasets import load_irisimport pandas as pdPreparing the data We use Iris dataset to perform classification and it can be easily loaded from the Scikit-learn dataset module. Below code explains how to load dataset and transform it into the pandas data frame type.iris = load_iris()df_iris =
pd.DataFrame(iris.data, columns=iris.feature_names)df_iris['label'] = pd.Series(iris.target)print(df_iris.head()) sepal length (cm) ... petal width (cm) label0 5.1 3.5 ... 0.2 01 4.9 3.0 ... 0.2 02 4.7 3.2 ... 0.2 03 4.6 3.1 ... 0.2 04 5.0 3.6 ... 0.2 0Next, we'll define SqlConext and create data frame by using df_iris.data.sc =
SparkContext().getOrCreate(sqlContext = SQLContext(sc) data = sqlContext.createDataFrame(df_iris)print(data.printSchema())root |-- sepal length (cm): double (nullable = true) |-- sepal width (cm): double (nullable = true) |-- petal length (cm): double (nullable = true) |-- petal width (cm): double (nullable = true) |-- label: long (nullable = true)To
combine all feature data and separate 'label' data in a dataset, we use VectorAssembler.features = iris.feature_names va = VectorAssembler(inputCols = features, outputCol='features') va.df = va.transform(data)va.df = va.df.select(['features', 'label'])va.df.show(3) (train_test) = va.df.randomSplit([0.8, 0.2]) dte = DecisionTreeClassifier(featuresCol="features", labelCol="label")dte = dte.fit(train) pred = dte.transform(test)pred.show(3)+-----+ features|label|+-----+ [5.1,3.5,1.4,0.2] 0][
4.9,3.0,1.4,0.2] 0][[4.7,3.2,1.3,0.2] 0]+-----+only showing top 3 rowsNext, we'll split data into the train and test parts.(train_test) = va.df.randomSplit([0.8, 0.2]) Prediction and Accuracy Check Next, we'll define the decision tree classifier model by using the DecisionTreeClassifierclass and fit model on train data. We can predict test data
by using transfrom() method.dtc = DecisionTreeClassifier(featuresCol="features", labelCol="label")dte = dtc.fit(train) pred = dtc.transform(test)pred.show(3)+-----+ features|label| rawPrediction| probability|prediction|+-----+ [5.1,3.5,1.4,0.2] 0][
1.0,0.0,0.0] 0.0][[4.6,3.1,1.5,0.2] 0][[35.0,0.0,0.0] 0][[1.0,0.0,0.0] 0][[5.0,3.6,1.4,0.2] 0][[35.0,0.0,0.0] 0][[1.0,0.0,0.0] 0]+-----+only showing top 3 rows After training the model, we'll predict test data and check the accuracy metrics. Here, we can use MulticlassClassificationEvaluator to check the accuracy.
Confusion matrix can be created by using confusion_matrix function of sklearn.metrics module.evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", acc = evaluator.evaluate(pred)print("Prediction Accuracy: ", acc) y_pred=pred.select("prediction").collect() y_orig=pred.select("label").collect() cm = confusion_matrix(y_orig,
y_pred)print("Confusion Matrix:")print(cm)Prediction Accuracy: 1.0Confusion Matrix:[13 0 0][ 0 5 0][ 0 0 10]Finally, we'll stop spark context session. In this tutorial, we've briefly learned how to fit and classify data by using PySpark DecisionTreeClassifier class. The full source code is listed below. Source code listingfrom pyspark import
SparkContextfrom pyspark.sql import SQLContextfrom pyspark.ml.classification import DecisionTreeClassifierfrom pyspark.ml.evaluation import MulticlassClassificationEvaluatorfrom pyspark.ml.feature import VectorAssemblerfrom sklearn.metrics import confusion_matrixfrom sklearn.datasets import load_irisimport pandas as pd iris =
load_iris()df_iris = pd.DataFrame(iris.data, columns=iris.feature_names)df_iris['label'] = pd.Series(iris.target)print(df_iris.head()) sc = SparkContext().getOrCreate(sqlContext = SQLContext(sc) data = sqlContext.createDataFrame(df_iris)print(data.printSchema()) features = iris.feature_names va = VectorAssembler(inputCols = features,
outputCol='features') va.df = va.transform(data)va.df = va.df.select(['features', 'label'])va.df.show(3) (train_test) = va.df.randomSplit([0.8, 0.2]) dte = DecisionTreeClassifier(featuresCol="features", labelCol="label")dte = dte.fit(train) pred = dte.transform(test)pred.show(3)
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")acc = evaluator.evaluate(pred)print("Prediction Accuracy: ", acc) y_pred=pred.select("prediction").collect() y_orig=pred.select("label").collect() cm = confusion_matrix(y_orig, y_pred)print("Confusion Matrix:")print(cm) sc.stop()References:PySpark Decision tree classifier
Decision trees are a popular family of classification and regression methods. More information about the spark.ml implementation can be found further in the section on decision trees.ExamplesThe following examples load a dataset in LibSVM format, split it into training and test sets, train on the held-out test
set. We use a feature transformer to index categorical features, adding metadata to the DataFrame which the Decision Tree algorithm can recognize.import org.apache.spark.ml.Pipelineimport org.apache.spark.ml.evaluation.RegressionEvaluatorimport org.apache.spark.ml.feature.VectorIndexerimport
org.apache.spark.ml.regression.DecisionTreeRegressionModelimport org.apache.spark.ml.regression.DecisionTreeRegressor// Load the data stored in LIBSVM format as a DataFrame.val data = spark.read.format("libsvm").load("/file:/opt/spark/data/mllib/sample_libsvm_data.txt")// Automatically identify categorical features, and index them.// Here,
we treat features with > 4 distinct values as continuous.val featureIndexer = new VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCategories(4).fit(data)// Split the data into training and test sets (30% held out for testing).val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))// Train a DecisionTree
model.val dt = new DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("indexedFeatures")// Chain indexer and tree in a Pipeline.val pipeline = new Pipeline().setStages(Array(featureIndexer, dt))// Train model. This also runs the indexer.val model = pipeline.fit(trainingData)// Make predictions.val predictions = model.transform(testData)//
Select example rows to display.predictions.select("prediction", "label", "features").show(5)// Select (prediction, true label) and compute test error.val evaluator = new RegressionEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("rmse")val rmse = evaluator.evaluate(predictions)println(s"Root Mean Squared Error
(RMSE) on test data = $rmse")val treeModel = model.stages(1).asInstanceOf[DecisionTreeRegressionModel]println(s"Learned regression tree model: ${treeModel.toDebugString}")/*Output: +-----+ [prediction|label] features|+-----+ +-----+ [0.0] 0.0|[692,123,124,125...]| 0.0] 0.0|[692,124,125,126...]| 0.0] 0.0|[692,124,125,126...]| 0.0] 0.0|[692,126,127,128...]| 0.0] 0.0|[692,126,127,128...]| 0.0] 0.0|[692,127,128,129...]| 0.0] 0.0|[692,127,128,129...]| 0.0] 0.0|[692,128,129,130...]| 0.0] 0.0|[692,128,129,130...]| 0.0] 0.0|[692,129,130,131...]| 0.0] 0.0|[692,129,130,131...]| 0.0] 0.0|[692,130,131,132...]| 0.0] 0.0|[692,130,131,132...]| 0.0] 0.0|[692,131,132,133...]| 0.0] 0.0|[692,131,132,133...]| 0.0] 0.0|[692,132,133,134...]| 0.0] 0.0|[692,132,133,134...]| 0.0] 0.0|[692,133,134,135...]| 0.0] 0.0|[692,133,134,135...]| 0.0] 0.0|[692,134,135,136...]| 0.0] 0.0|[692,134,135,136...]| 0.0] 0.0|[692,135,136,137...]| 0.0] 0.0|[692,135,136,137...]| 0.0] 0.0|[692,136,137,138...]| 0.0] 0.0|[692,136,137,138...]| 0.0] 0.0|[692,137,138,139...]| 0.0] 0.0|[692,137,138,139...]| 0.0] 0.0|[692,138,139,140...]| 0.0] 0.0|[692,138,139,140...]| 0.0] 0.0|[692,139,140,141...]| 0.0] 0.0|[692,139,140,141...]| 0.0] 0.0|[692,140,141,142...]| 0.0] 0.0|[692,140,141,142...]| 0.0] 0.0|[692,141,142,143...]| 0.0] 0.0|[692,141,142,143...]| 0.0] 0.0|[692,142,143,144...]| 0.0] 0.0|[692,142,143,144...]| 0.0] 0.0|[692,143,144,145...]| 0.0] 0.0|[692,143,144,145...]| 0.0] 0.0|[692,144,145,146...]| 0.0] 0.0|[692,144,145,146...]| 0.0] 0.0|[692,145,146,147...]| 0.0] 0.0|[692,145,146,147...]| 0.0] 0.0|[692,146,147,148...]| 0.0] 0.0|[692,146,147,148...]| 0.0] 0.0|[692,147,148,149...]| 0.0] 0.0|[692,147,148,149...]| 0.0] 0.0|[692,148,149,150...]| 0.0] 0.0|[692,148,149,150...]| 0.0] 0.0|[692,149,150,151...]| 0.0] 0.0|[692,149,150,151...]| 0.0] 0.0|[692,150,151,152...]| 0.0] 0.0|[692,150,151,152...]| 0.0] 0.0|[692,151,152,153...]| 0.0] 0.0|[692,151,152,153...]| 0.0] 0.0|[692,152,153,154...]| 0.0] 0.0|[692,152,153,154...]| 0.0] 0.0|[692,153,154,155...]| 0.0] 0.0|[692,153,154,155...]| 0.0] 0.0|[692,154,155,156...]| 0.0] 0.0|[692,154,155,156...]| 0.0] 0.0|[692,155,156,157...]| 0.0] 0.0|[692,155,156,157...]| 0.0] 0.0|[692,156,157,158...]| 0.0] 0.0|[692,156,157,158...]| 0.0] 0.0|[692,157,158,159...]| 0.0] 0.0|[692,157,158,159...]| 0.0] 0.0|[692,158,159,160...]| 0.0] 0.0|[692,158,159,160...]| 0.0] 0.0|[692,159,160,161...]| 0.0] 0.0|[692,159,160,161...]| 0.0] 0.0|[692,160,161,162...]| 0.0] 0.0|[692,160,161,162...]| 0.0] 0.0|[692,161,162,163...]| 0.0] 0.0|[692,161,162,163...]| 0.0] 0.0|[692,162,163,164...]| 0.0] 0.0|[692,162,163,164...]| 0.0] 0.0|[692,163,164,165...]| 0.0] 0.0|[692,163,164,165...]| 0.0] 0.0|[692,164,165,166...]| 0.0] 0.0|[692,164,165,166...]| 0.0] 0.0|[692,165,166,167...]| 0.0] 0.0|[692,165,166,167...]| 0.0] 0.0|[692,166,167,168...]| 0.0] 0.0|[692,166,167,168...]| 0.0] 0.0|[692,167,168,169...]| 0.0] 0.0|[692,167,168,169...]| 0.0] 0.0|[692,168,169,170...]| 0.0] 0.0|[692,168,169,170...]| 0.0] 0.0|[692,169,170,171...]| 0.0] 0.0|[692,169,170,171...]| 0.0] 0.0|[692,170,171,172...]| 0.0] 0.0|[692,170,171,172...]| 0.0] 0.0|[692,171,172,173...]| 0.0] 0.0|[692,171,172,173...]| 0.0] 0.0|[692,172,173,174...]| 0.0] 0.0|[692,172,173,174...]| 0.0] 0.0|[692,173,174,175...]| 0.0] 0.0|[692,173,174,175...]| 0.0] 0.0|[692,174,175,176...]| 0.0] 0.0|[692,174,175,176...]| 0.0] 0.0|[692,175,176,177...]| 0.0] 0.0|[692,175,176,177...]| 0.0] 0.0|[692,176,177,178...]| 0.0] 0.0|[692,176,177,178...]| 0.0] 0.0|[692,177,178,179...]| 0.0] 0.0|[692,177,178,179...]| 0.0] 0.0|[692,178,179,180...]| 0.0] 0.0|[692,178,179,180...]| 0.0] 0.0|[692,179,180,181...]| 0.0] 0.0|[692,179,180,181...]| 0.0] 0.0|[692,180,181,182...]| 0.0] 0.0|[692,180,181,182...]| 0.0] 0.0|[692,181,182,183...]| 0.0] 0.0|[692,181,182,183...]| 0.0] 0.0|[692,182,183,184...]| 0.0] 0.0|[692,182,183,184...]| 0.0] 0.0|[692,183,184,185...]| 0.0] 0.0|[692,183,184,185...]| 0.0] 0.0|[692,184,185,186...]| 0.0] 0.0|[692,184,185,186...]| 0.0] 0.0|[692,185,186,187...]| 0.0] 0.0|[692,185,186,187...]| 0.0] 0.0|[692,186,187,188...]| 0.0] 0.0|[692,186,187,188...]| 0.0] 0.0|[692,187,188,189...]| 0.0] 0.0|[692,187,188,189...]| 0.0] 0.0|[692,188,189,190...]| 0.0] 0.0|[692,188,189,190...]| 0.0] 0.0|[692,189,190,191...]| 0.0] 0.0|[692,189,190,191...]| 0.0] 0.0|[692,190,191,192...]| 0.0] 0.0|[692,190,191,192...]| 0.0] 0.0|[692,191,192,193...]| 0.0] 0.0|[692,191,192,193...]| 0.0] 0.0|[692,192,193,194...]| 0.0] 0.0|[692,192,193,194...]| 0.0] 0.0|[692,193,194,195...]| 0.0] 0.0|[692,193,194,195...]| 0.0] 0.0|[692,194,195,196...]| 0.0] 0.0|[692,194,195,196...]| 0.0] 0.0|[692,195,196,197...]| 0.0] 0.0|[692,195,196,197...]| 0.0] 0.0|[692,196,197,198...]| 0.0] 0.0|[692,196,197,198...]| 0.0] 0.0|[692,197,198,199...]| 0.0] 0.0|[692,197,198,199...]| 0.0] 0.0|[692,198,199,200...]| 0.0] 0.0|[692,198,199,200...]| 0.0] 0.0|[692,199,200,201...]| 0.0] 0.0|[692,199,200,201...]| 0.0] 0.0|[692,200,201,202...]| 0.0] 0.0|[692,200,201,202...]| 0.0] 0.0|[692,201,202,203...]| 0.0] 0.0|[692,201,202,203...]| 0.0] 0.0|[692,202,203,204...]| 0.0] 0.0|[692,202,203,204...]| 0.0] 0.0|[692,203,204,205...]| 0.0] 0.0|[692,203,204,205...]| 0.0] 0.0|[692,204,205,206...]| 0.0] 0.0|[692,204,205,206...]| 0.0] 0.0|[692,205,206,207...]| 0.0] 0.0|[692,205,206,207...]| 0.0] 0.0|[692,206,207,208...]| 0.0] 0.0|[692,206,207,208...]| 0.0] 0.0|[692,207,208,209...]| 0.0] 0.0|[692,207,208,209...]| 0.0] 0.0|[692,208,209,210...]| 0.0] 0.0|[692,208,209,210...]| 0.0] 0.0|[692,209,210,211...]| 0.0] 0.0|[692,209,210,211...]| 0.0] 0.0|[692,210,211,212...]| 0.0] 0.0|[692,210,211,212...]| 0.0] 0.0|[692,211,212,213...]| 0.0] 0.0|[692,211,212,213...]| 0.0] 0.0|[692,212,213,214...]| 0.0] 0.0|[692,212,213,214...]| 0.0] 0.0|[692,213,214,215...]| 0.0] 0.0|[692,213,214,215...]| 0.0] 0.0|[692,214,215,216...]| 0.0] 0.0|[692,214,215,216...]| 0.0] 0.0|[692,215,216,217...]| 0.0] 0.0|[692,215,216,217...]| 0.0] 0.0|[692,216,217,218...]| 0.0] 0.0|[692,216,217,218...]| 0.0] 0.0|[692,217,218,219...]| 0.0] 0.0|[692,217,218,219...]| 0.0] 0.0|[692,218,219,220...]| 0.0] 0.0|[692,218,219,220...]| 0.0] 0.0|[692,219,220,221...]| 0.0] 0.0|[692,219,220,221...]| 0.0] 0.0|[692,220,221,222...]| 0.0] 0.0|[692,220,221,222...]| 0.0] 0.0|[692,221,222,223...]| 0.0] 0.0|[692,221,222,223...]| 0.0] 0.0|[692,222,223,224...]| 0.0] 0.0|[692,222,223,224...]| 0.0] 0.0|[692,223,224,225...]| 0.0] 0.0|[692,223,224,225...]| 0.0] 0.0|[692,224,225,226...]| 0.0] 0.0|[692,224,225,226...]| 0.0] 0.0|[692,225,226,227...]| 0.0] 0.0|[692,225,226,227...]| 0.0] 0.0|[692,226,227,228...]| 0.0] 0.0|[692,226,227,228...]| 0.0] 0.0|[692,227,228,229...]| 0.0] 0.0|[692,227,228,229...]| 0.0] 0.0|[692,228,229,230...]| 0.0] 0.0|[692,228,229,230...]| 0.0] 0.0|[692,229,230,231...]| 0.0] 0.0|[692,229,230,231...]| 0.0] 0.0|[692,230,231,232...]| 0.0] 0.0|[692,230,231,232...]| 0.0] 0.0|[692,231,232,233...]| 0.0] 0.0|[692,231,232,233...]| 0.0] 0.0|[692,232,233,234...]| 0.0] 0.0|[692,232,233,234...]| 0.0] 0.0|[692,233,234,235...]| 0.0] 0.0|[692,233,234,235...]| 0.0] 0.0|[692,234,235,236...]| 0.0] 0.0|[692,234,235,236...]| 0.0] 0.0|[692,235,236,237...]| 0.0] 0.0|[692,235,236,237...]| 0.0] 0.0|[692,236,237,238...]| 0.0] 0.0|[692,236,237,238...]| 0.0] 0.0|[692,237,238,239...]| 0.0] 0.0|[692,237,238,239...]| 0.0] 0.0|[692,238,239,240...]| 0.0] 0.0|[692,238,239,240...]| 0.0] 0.0|[692,239,240,241...]| 0.0] 0.0|[692,239,240,241...]| 0.0] 0.0|[692,240,241,242...]| 0.0] 0.0|[692,240,241,242...]| 0.0] 0.0|[692,241,242,243...]| 0.0] 0.0|[692,241,242,243...]| 0.0] 0.0|[692,242,243,244...]| 0.0] 0.0|[692,242,243,244...]| 0.0] 0.0|[692,243,244,245...]| 0.0] 0.0|[692,243,244,245...]| 0.0] 0.0|[692,244,245,246...]| 0.0] 0.0|[692,244,245,246...]| 0.0] 0.0|[692,245,246,247...]| 0.0] 0.0|[692,245,246,247...]| 0.0] 0.0|[692,246,247,248...]| 0.0] 0.0|[692,246,247,248...]| 0.0] 0.0|[692,247,248,249...]| 0.0] 0.0|[692,247,248,249...]| 0.0] 0.0|[692,248,249,250...]| 0.0] 0.0|[692,248,249,250...]| 0.0] 0.0|[692,249,250,251...]| 0.0] 0.0|[692,249,250,251...]| 0.0] 0.0|[692,250,251,252...]| 0.0] 0.0|[692,250,251,252...]| 0.0] 0.0|[692,251,252,253...]| 0.0] 0.0|[692,251,252,253...]| 0.0] 0.0|[692,252,253,254...]| 0.0] 
```