

## Link prediction:

### //graph projections for link pred

```
CALL gds.graph.project(
  'linkp',
  'City',
  {
    FLIGHT_TO:{properties:['flightID'],
    orientation: 'UNDIRECTED'
  }
})
```

```
CALL gds.graph.project(
  'fullGraph',
  ['City', 'Operator', 'CrashLocation'],
  {
    FLIGHT_TO: {
      orientation: 'UNDIRECTED',
      properties: ['flightID']
    },
    OPERATED_AT: {},
    CRASHED_At: {properties: ['flightID']}
  }
)
```

### //pipeline1 (link pred pipeline for linkp projection)

```
CALL gds.beta.pipeline.linkPrediction.create('pipe')
```

```
CALL gds.beta.pipeline.linkPrediction.addNodeProperty('pipe', 'fastRP', {
  mutateProperty: 'embedding',
  embeddingDimension: 256,
  randomSeed: 42
})
```

```
CALL gds.beta.pipeline.linkPrediction.addFeature('pipe', 'hadamard', {
```

```
    nodeProperties: ['embedding']
  }) YIELD featureSteps
```

```
CALL gds.beta.pipeline.linkPrediction.configureSplit('pipe', {
  testFraction: 0.25,
  trainFraction: 0.6,
  validationFolds: 3
})
YIELD splitConfig
```

```
CALL gds.beta.pipeline.linkPrediction.addLogisticRegression('pipe')
YIELD parameterSpace
```

```
CALL gds.beta.pipeline.linkPrediction.addRandomForest('pipe', {numberOfDecisionTrees: 20})
YIELD parameterSpace
```

```
CALL gds.alpha.pipeline.linkPrediction.addMLP('pipe',
{hiddenLayerSizes: [16, 8], penalty: 0.5, patience: 5, learningRate: 0.01, classWeights: [0.55,
0.45], focusWeight: {range: [0.0, 0.1]}})
YIELD parameterSpace
```

```
CALL gds.alpha.pipeline.linkPrediction.configureAutoTuning('pipe', {
  maxTrials: 10
}) YIELD autoTuningConfig
```

```
CALL gds.beta.pipeline.linkPrediction.train.estimate('linkp', {
  pipeline: 'pipe',
  modelName: 'lp-pipeline-model',
  targetRelationshipType: 'FLIGHT_TO'
})
YIELD requiredMemory
```

```
CALL gds.beta.pipeline.linkPrediction.train('linkp', {
  pipeline: 'pipe',
  modelName: 'lp-pipeline-model',
  metrics: ['AUCPR', 'OUT_OF_BAG_ERROR'],
```

```

targetRelationshipType: 'FLIGHT_TO',
randomSeed: 42
}) YIELD modelInfo, modelSelectionStats
RETURN
  modelInfo.bestParameters AS winningModel,
  modelInfo.metrics.AUCPR.train.avg AS avgTrainScore,
  modelInfo.metrics.AUCPR.outerTrain AS outerTrainScore,
  modelInfo.metrics.AUCPR.test AS testScore,
  [cand IN modelSelectionStats.modelCandidates | cand.metrics.AUCPR.validation.avg] AS
validationScores

```

winningModel	avgTrainScore	outerTrainScore	testScore	validationScores
<pre> {   "minEpochs": 1,   "maxEpochs": 100,   "FocusWeight": 0.01599103928769201,   "patience": 5,   "tolerance": 0.001,   "learningRate": 0.01,   "hiddenLayerSizes": [     16,     8   ],   "batchSize": 100,   "penalty": 0.5,   "methodName": "MultilayerPerceptron",   "classWeights": [] } </pre>	0.75	0.75	0.7411764705882353	[0.6760152803729936, 0.6532438942858799, 0.6693217991687783, 0.75, 0.5627017363024619, 0.75, 0.75, 0.56232055]

```

CALL gds.beta.pipeline.linkPrediction.predict.stream.estimate('linkp', {
  modelName: 'lp-pipeline-model',
  topN: 5,
  threshold: 0.5
})
YIELD requiredMemory

```

```

CALL gds.beta.pipeline.linkPrediction.predict.stream('linkp', {
  modelName: 'lp-pipeline-model',
  topN: 5,
  threshold: 0.5
})
YIELD node1, node2, probability
RETURN gds.util.asNode(node1).Name AS city1, gds.util.asNode(node2).Name AS city2,
probability
ORDER BY probability DESC, city1

```

```
CALL gds.beta.pipeline.linkPrediction.predict.mutate('linkp', {
  modelName: 'lp-pipeline-model',
  relationshipTypes: ['FLIGHT_TO'],
  mutateRelationshipType: 'FLIGHT_EXHAUSTIVE_PREDICTED',
  threshold: 0.5
}) YIELD relationshipsWritten, samplingStats
```

```
CALL gds.beta.pipeline.linkPrediction.predict.mutate('linkp', {
  modelName: 'lp-pipeline-model',
  relationshipTypes: ['FLIGHT_TO'],
  mutateRelationshipType: 'FLIGHT_APPROX_PREDICTED',
  sampleRate: 0.5,
  topK: 1,
  randomJoins: 2,
  maxIterations: 3,
  // necessary for deterministic results
  concurrency: 1,
  randomSeed: 42
})
YIELD relationshipsWritten, samplingStats
```

### **//pipeline2 (link pred pipeline for fullGraph projection) DOES NOT WORK**

```
CALL gds.beta.pipeline.linkPrediction.create('pipe-with-context')
```

```
CALL gds.beta.pipeline.linkPrediction.addNodeProperty('pipe-with-context', 'fastRP', {
  mutateProperty: 'embedding',
  embeddingDimension: 256,
  randomSeed: 42,
  contextNodeLabels: ['CrashLocation', 'Operator'],
  contextRelationshipTypes: ['CRASHED_AT', 'OPERATED_AT']
})
```

```
CALL gds.beta.pipeline.linkPrediction.addFeature('pipe-with-context', 'hadamard', {
```

```
    nodeProperties: ['embedding']
  })
```

```
CALL gds.beta.pipeline.linkPrediction.configureSplit('pipe-with-context', {
  testFraction: 0.25,
  trainFraction: 0.6,
  validationFolds: 3
})
```

```
CALL gds.alpha.pipeline.linkPrediction.addMLP('pipe-with-context',
{hiddenLayerSizes: [4, 2], penalty: 1, patience: 2})
```

```
CALL gds.beta.pipeline.linkPrediction.addLogisticRegression('pipe-with-context')
YIELD parameterSpace
```

```
CALL gds.beta.pipeline.linkPrediction.train('fullGraph', {
  pipeline: 'pipe-with-context',
  modelName: 'lp-pipeline-model-filtered',
  metrics: ['AUCPR', 'OUT_OF_BAG_ERROR'],
  sourceNodeLabel: 'City',
  targetNodeLabel: 'City',
  targetRelationshipType: 'FLIGHT_TO',
  randomSeed: 12
}) YIELD modelInfo, modelSelectionStats
RETURN
  modelInfo.bestParameters AS winningModel,
  modelInfo.metrics.AUCPR.train.avg AS avgTrainScore,
  modelInfo.metrics.AUCPR.outerTrain AS outerTrainScore,
  modelInfo.metrics.AUCPR.test AS testScore,
  [cand IN modelSelectionStats.modelCandidates | cand.metrics.AUCPR.validation.avg] AS
validationScores
```

## //Node classification

//we can only work w numerical properties when it comes to node classification

**Node: AircraftType**

**Properties: Make, Operator**

//since our properties are not numerical, we first make them numerical.

```
//makeNum corresponds to Make, class corresponds to Operator
//class has already been loaded.
//new csv for the makeNum of every Make
```

```
1)
//LOAD makeNum from make.csv
```

```
LOAD CSV WITH HEADERS FROM 'file:///make.csv' AS row
MERGE (at:AircraftType {Make: row.ACType})
SET at.MakeNum = toInteger(row.MakeNum)
```

2) we need to make some nodes unknown

```
MATCH (a:AircraftType)
WITH a
LIMIT 40
SET a:UnknownAircraftType
REMOVE a.class
```

3) need to remove aircrafttype as their label

```
MATCH (a:UnknownAircraftType)
REMOVE a:AircraftType
```

Now we start

**Graph projection:**

```
CALL gds.graph.project('class', {
  AircraftType: { properties: ['MakeNum', 'class'] },
  UnknownAircraftType: { properties: 'MakeNum' }
},
/*
*)
```

**Pipeline:**

```
CALL gds.beta.pipeline.nodeClassification.create('pipe2')
```

```
CALL gds.beta.pipeline.nodeClassification.addNodeProperty('pipe2', 'scaleProperties', {
  nodeProperties: 'MakeNum',
```

```
    scaler: 'Mean',  
    mutateProperty:'scaledSizes'  
  })  
YIELD name, nodePropertySteps
```

```
CALL gds.beta.pipeline.nodeClassification.selectFeatures('pipe2', ['scaledSizes', 'MakeNum'])  
YIELD name, featureProperties
```

```
CALL gds.beta.pipeline.nodeClassification.configureSplit('pipe2', {  
  testFraction: 0.2,  
  validationFolds: 5  
})  
YIELD splitConfig
```

```
CALL gds.beta.pipeline.nodeClassification.addLogisticRegression('pipe2')  
YIELD parameterSpace
```

```
CALL gds.beta.pipeline.nodeClassification.addRandomForest('pipe2', {numberOfDecisionTrees:  
10})  
YIELD parameterSpace
```

```
CALL gds.alpha.pipeline.nodeClassification.addMLP('pipe2', {classWeights: [0.4,0.3,0.3],  
focusWeight: 0.5})  
YIELD parameterSpace
```

```
CALL gds.beta.pipeline.nodeClassification.addLogisticRegression('pipe2', {maxEpochs: 500,  
penalty: {range: [1e-4, 1e2]}})  
YIELD parameterSpace  
RETURN parameterSpace.RandomForest AS randomForestSpace,  
parameterSpace.LogisticRegression AS logisticRegressionSpace,  
parameterSpace.MultilayerPerceptron AS MultilayerPerceptronSpace
```

```
CALL gds.alpha.pipeline.nodeClassification.configureAutoTuning('pipe2', {
  maxTrials: 2
}) YIELD autoTuningConfig
```

```
CALL gds.beta.pipeline.nodeClassification.train.estimate('class', {
  pipeline: 'pipe2',
  targetNodeLabels: ['AircraftType'],
  modelName: 'nc-model',
  targetProperty: 'class',
  randomSeed: 2,
  metrics: ['ACCURACY']
})
YIELD requiredMemory
```

```
CALL gds.beta.pipeline.nodeClassification.train('class', {
  pipeline: 'pipe2',
  targetNodeLabels: ['AircraftType'],
  modelName: 'nc-pipeline-model',
  targetProperty: 'class',
  randomSeed: 1227,
  metrics: ['ACCURACY']
}) YIELD modelInfo, modelSelectionStats
RETURN
  modelInfo.bestParameters AS winningModel,
  modelInfo.metrics.ACCURACY.train.avg AS avgTrainScore,
  modelInfo.metrics.ACCURACY.outerTrain AS outerTrainScore,
  modelInfo.metrics.ACCURACY.test AS testScore,
  [cand IN modelSelectionStats.modelCandidates | cand.metrics.ACCURACY.validation.avg] AS
validationScores
```

winningModel	avgTrainScore	outerTrainScore	testScore	validationScores
<pre>{   "maxDepth": 2147483647,   "criterion": "GINI",   "minSplitSize": 2,   "minLeafSize": 1,   "numberOfSamplesRatio": 1.0,   "methodName": "RandomForest",   "numberOfDecisionTrees": 10 }</pre>	0.748528122	0.73391813	0.6511628	[0.710528564, 0.745609552, 0.710528564, 0.710528564, 0.710528564]

```
CALL gds.beta.pipeline.nodeClassification.predict.stream('class', {
  modelName: 'nc-pipeline-model',
  includePredictedProbabilities: true,
```



```

    targetNodeLabels: ['UnknownAircraftType']
  })
  YIELD nodeId, predictedClass, predictedProbabilities
WITH gds.util.asNode(nodeId) AS aircraftNode, predictedClass, predictedProbabilities
RETURN
  aircraftNode.Make AS classifiedAircraft,
  predictedClass,
  floor(predictedProbabilities[predictedClass] * 100) AS confidence
ORDER BY classifiedAircraft

```

	classifiedAircraft	predictedClass	confidence
1	"ATR 42"	0	100.0
2	"ATR 72"	0	100.0
3	"Aero Commander"	2	100.0
4	"Aeroflot"	2	100.0
5	"Aeromarine"	2	100.0
6	"Aerospatiale"	2	80.0
7			

ted streaming 40 records after 17 ms and completed after 189 ms.

```

CALL gds.beta.pipeline.nodeClassification.predict.mutate('class', {
  targetNodeLabels: ['UnknownAircraftType'],
  modelName: 'nc-pipeline-model',
  mutateProperty: 'predictedClass',
  predictedProbabilityProperty: 'predictedProbabilities'
}) YIELD nodePropertiesWritten

```