서울대학교
융합과학기술대학원
Seoul National University
Graduate School of Convergence
Science and Technology

Classification

Classification Algorithms

Decision Tree

Logistic Regression

SVM

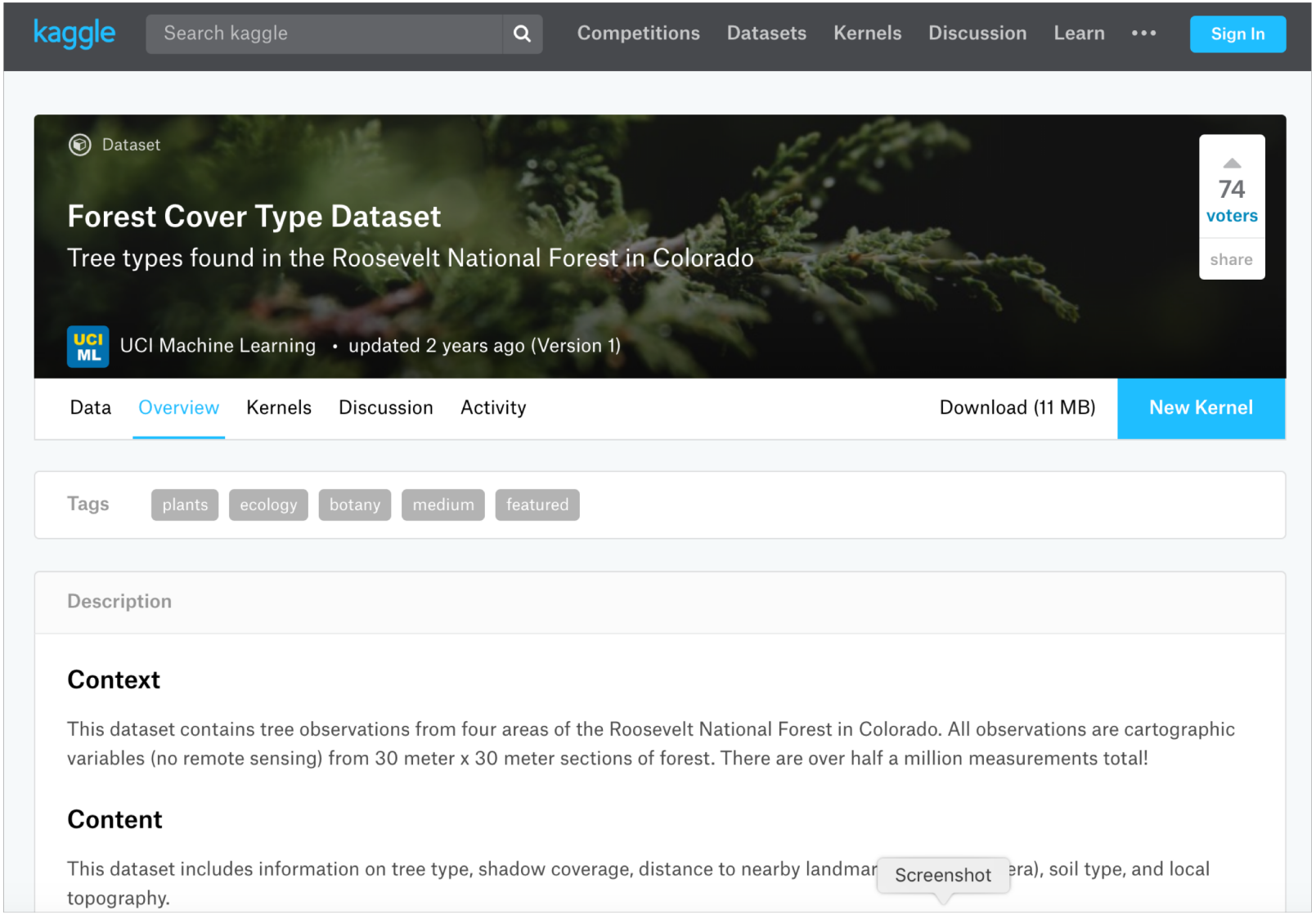
Random Forest

kNN

Ensemble Methods

Forest Cover Data

https://www.kaggle.com/uciml/forest-cover-type-dataset/home



Cover Type Data

- 지형 type data

Elevation	quantitative	meters	Elevation in meters
Aspect	quantitative	azimuth	Aspect in degrees azimuth
Slope	quantitative	degrees	Slope in degrees
Horizontal_Distance_To_Hydrology	quantitative	meters	Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology	quantitative	meters	Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways	quantitative	meters	Horz Dist to nearest roadway
Hillshade_9am	quantitative	0 to 255 index	Hillshade index at 9am, summer solstice
Hillshade_Noon	quantitative	0 to 255 index	Hillshade index at noon, summer soltice
Hillshade_3pm	quantitative	0 to 255 index	Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points	quantitative	meters	Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns)	qualitative	0 (absence) or 1 (presence)	Wilderness area designation
Soil_Type (40 binary columns)	qualitative	0 (absence) or 1 (presence)	Soil Type designation
Cover_Type (7 types)	integer	1 to 7	Forest Cover Type designation

class `pyspark.mllib.tree.`**`DecisionTree`** [source]

Learning algorithm for a decision tree model for classification or regression.

New in version 1.1.0.

classmethod **`trainClassifier`**(*data*, *numClasses*, *categoricalFeaturesInfo*, *impurity='gini'*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*) [source]

Train a decision tree model for classification.

Parameters:
- **data** – Training data: RDD of LabeledPoint. Labels should take values {0, 1, ..., numClasses-1}.
- **numClasses** – Number of classes for classification.
- **categoricalFeaturesInfo** – Map storing arity of categorical features. An entry (n -> k) indicates that feature n is categorical with k categories indexed from 0: {0, 1, ..., k-1}.
- **impurity** – Criterion used for information gain calculation. Supported values: “gini” or “entropy”. (default: “gini”)
- **maxDepth** – Maximum depth of tree (e.g. depth 0 means 1 leaf node, depth 1 means 1 internal node + 2 leaf nodes). (default: 5)
- **maxBins** – Number of bins used for finding splits at each node. (default: 32)
- **minInstancesPerNode** – Minimum number of instances required at child nodes to create the parent split. (default: 1)
- **minInfoGain** – Minimum info gain required to create a split. (default: 0.0)

Returns: DecisionTreeModel.

VectorAssembler

- Input에 0이 많음
- 0이 아닌 숫자의 index와 그 값만 가지고 있는다.

```
assembler = VectorAssembler(inputCols=inputCols,
    outputCol="featureVector")

assembledTrainData.select('featureVector').show(3, truncate=False)
```

Train

```
from pyspark.ml.classification import DecisionTreeClassifier

classifier = DecisionTreeClassifier(labelCol="Cover_Type",
                                    featuresCol="featureVector",
                                    predictionCol="prediction")

model = classifier.fit(assembledTrainData)
```

Prediction

```
predictions = model.transform(assembledTrainData)
```

```
predictions.select(["Cover_Type", "prediction",
"probability"]).show(3, truncate=False)
```

```
predictions.select(["Cover_Type", "prediction", "probability"]).show(3, truncate=False)
```

```
+----------+----------+-------------------------------------------------------------------------
-------------------------+
|Cover_Type|prediction|probability
                          |
+----------+----------+-------------------------------------------------------------------------
-------------------------+
|6.0       |3.0       |[0.0,0.0,0.0392960146827477,0.6253932878867331,0.04758783429470372,0.
0,0.2877228631358154,0.0]|
|6.0       |4.0       |[0.0,0.0,0.048494983277591976,0.2842809364548495,0.411371237458194,0.
0,0.25585284280936454,0.0]|
|6.0       |3.0       |[0.0,0.0,0.0392960146827477,0.6253932878867331,0.04758783429470372,0.
0,0.2877228631358154,0.0]|
+----------+----------+-------------------------------------------------------------------------
-------------------------+
only showing top 3 rows
```

평가

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="Cover_Type",
                                              predictionCol="prediction")
```

```
evaluator
```

```
MulticlassClassificationEvaluator_443a80fe06ed19070a69
```

```
evaluator.setMetricName("accuracy").evaluate(predictions)
```

```
0.7003371765857239
```

```
evaluator.setMetricName("f1").evaluate(predictions)
```

```
0.6833067645308728
```

Pipeline

- 위와 같은 절차를 한번에 수행해 보자

```
from pyspark.ml import Pipeline

inputCols = trainData.columns[:-1]
assembler = VectorAssembler(inputCols=inputCols, outputCol="featureVector")

classifier = DecisionTreeClassifier(labelCol="Cover_Type",
                                    featuresCol="featureVector",
                                    predictionCol="prediction")

pipeline = Pipeline(stages=[assembler, classifier])
```

Parameter Grid

```
from pyspark.ml.tuning import ParamGridBuilder
paramGrid = ParamGridBuilder() \
    .addGrid(classifier.xxxx, ["gini", "entropy"])\     # impurity
    .addGrid(classifier.xxxx, [1, 20])\                 # max depth
    .addGrid(classifier.xxxx, [40, 300])\               # max bins
    .addGrid(classifier.xxxx, [0.0, 0.05])\             # min Infogain
    .build()
```

Evaluator

```
multiclassEval = MulticlassClassificationEvaluator(
    labelCol="Cover_Type",
    predictionCol="prediction",
    metricName="accuracy")

# 기존 방법
# predictions = model.transform(assembledTrainData)
# multiclassEval = MulticlassClassificationEvaluator(
    labelCol="Cover_Type",
    predictionCol="prediction")
# predictions.select(["Cover_Type", "prediction", "probability"]).show(truncate=False)
# multiclassEval.setMetricName("f1")evaluate(predictions)
# multiclassEval.evaluate(predictions)
```

Pipeline + Validator

```
from pyspark.ml.tuning import TrainValidationSplit

validator = TrainValidationSplit(
    estimator=pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=multiclassEval,
    trainRatio=0.9)

validatorModel = validator.fit(trainData)

bestModel = validatorModel.bestModel
bestModel.stages[-1].extractParamMap()

multiclassEval.evaluate(bestModel.transform(testData))

0.702
```

결과

- {Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='cacheNodeIds',
  - doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees.'): False,
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='checkpointInterval',
  - doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext'): 10,
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='featuresCol',
  - doc='features column name'): 'featureVector',
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='impurity',
  - doc='Criterion used for information gain calculation (case-insensitive). Supported options: entropy, gini'): 'gini',
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='labelCol',
  - doc='label column name'): 'Cover_Type',
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='maxBins',
  - doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.'): 32,
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='maxDepth',
  - doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 5,

결과

- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='maxMemoryInMB',
  - doc='Maximum memory in MB allocated to histogram aggregation.'): 256,
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='minInfoGain',
  - doc='Minimum information gain for a split to be considered at a tree node.'): 0.0,
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='minInstancesPerNode',
  - doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.'): 1,
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='predictionCol',
  - doc='prediction column name'): 'prediction',
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='probabilityCol',
  - doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities'): 'probability',
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='rawPredictionCol',
  - doc='raw prediction (a.k.a. confidence) column name'): 'rawPrediction',
- Param(parent='DecisionTreeClassifier_41f4bd890c1c9c33fdc8', name='seed',
  - doc='random seed'): -5973628266081725150}

Random Forest

class pyspark.ml.classification.**RandomForestClassifier**(*featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', numTrees=20, featureSubsetStrategy='auto', seed=None, subsamplingRate=1.0*) [source]

Random Forest learning algorithm for classification. It supports both binary and multiclass labels, as well as both continuous and categorical features.

```
>>> import numpy
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> rf = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="indexed", seed=42)
>>> model = rf.fit(td)
```

Random Forest

```
from pyspark.ml.classification import RandomForestClassifier
classifier = RandomForestClassifier(
    seed=42,
    maxBins=40,
    labelCol="Cover_Type",
    featuresCol="indexedVector",
    predictionCol="prediction")

pipeline = Pipeline(stages=[assembler, indexer, classifier])
```

Random Forest

```
paramGrid = ParamGridBuilder()\
    .addGrid(classifier.minInfoGain, [0.0, 0.05])\
    .addGrid(classifier.numTrees, [1, 10])\
    .build()

multiclassEval = MulticlassClassificationEvaluator(
    labelCol="Cover_Type",
    predictionCol="prediction",
    metricName="accuracy")
```

실행 및 평가

```
validator = TrainValidationSplit(
    seed=42,
    estimator=pipeline,
    evaluator=multiclassEval,
    estimatorParamMaps=paramGrid,
    trainRatio=0.9)

validatorModel = validator.fit(…trainData)
bestModel = validatorModel.bestModel
forestModel = bestModel.stages[-1]
print(forestModel.extractParamMap())

multiclassEval.evaluate(bestModel.transform(…))
0.697
```

Logistic Regression

https://spark.apache.org/docs/latest/ml-classification-regression.html

```
from pyspark.ml.classification import LogisticRegression

# Load training data
training = spark \
    .read \
    .format("libsvm") \
    .load("data/mllib/sample_multiclass_classification_data.txt")

lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for multinomial logistic regression
print("Coefficients: \n" + str(lrModel.coefficientMatrix))
print("Intercept: " + str(lrModel.interceptVector))
```

API

- https://spark.apache.org/docs/latest/api/python/pyspark.ml.html

class pyspark.ml.classification.**LogisticRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol='probability', rawPredictionCol='rawPrediction', standardization=True, weightCol=None, aggregationDepth=2, family='auto', lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None*) [source]

Logistic regression. This class supports multinomial logistic (softmax) and binomial logistic regression.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> bdf = sc.parallelize([
...     Row(label=1.0, weight=1.0, features=Vectors.dense(0.0, 5.0)),
...     Row(label=0.0, weight=2.0, features=Vectors.dense(1.0, 2.0)),
...     Row(label=1.0, weight=3.0, features=Vectors.dense(2.0, 1.0)),
...     Row(label=0.0, weight=4.0, features=Vectors.dense(3.0, 3.0))]).toDF()
>>> blor = LogisticRegression(regParam=0.01, weightCol="weight")
>>> blorModel = blor.fit(bdf)
>>> blorModel.coefficients
DenseVector([-1.080..., -0.646...])
>>> blorModel.intercept
```

elasticNetParam = *Param(parent='undefined', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.')*

SVM

```python
from pyspark.ml.classification import LinearSVC

# Load training data
training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

lsvc = LinearSVC(maxIter=10, regParam=0.1)

# Fit the model
lsvcModel = lsvc.fit(training)

# Print the coefficients and intercept for linear SVC
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))
```

API

class pyspark.ml.classification.**LinearSVC**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2*) [source]

Note: Experimental

Linear SVM Classifier

This binary classifier optimizes the Hinge Loss using the OWLQN optimizer. Only supports L2 regularization currently.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = sc.parallelize([
...     Row(label=1.0, features=Vectors.dense(1.0, 1.0, 1.0)),
...     Row(label=0.0, features=Vectors.dense(1.0, 2.0, 3.0))]).toDF()
>>> svm = LinearSVC(maxIter=5, regParam=0.01)
>>> model = svm.fit(df)
>>> model.coefficients
DenseVector([0.0, -0.2792, -0.1833])
```