Chapter 26: Classification

Use Cases

- Predicting credit risk
- News classification
- Classifying human activity

Types of Classification

- Binary Classification
  - Image: cat/dog
- Multiclass Classification
  - Weather: sunny, rainy, cloudy
- Multilabel Classification
  - Genre

Classification Methods

- Logistic Regression
- Decision Trees
- Random Forests
- Gradient-boosted Trees
- Naïve Bayes

Model Scalability

Model	Features count	Training examples	Output classes
Logistic Regression	1 to 10 million	No limit	Features * Classes < 10 million
Decision Trees	1,000s	No limit	Features * Classes < 10,000s
Random Forests	10,000s	No limit	Features * Classes < 100,000s
Gradient-boosted Trees	1,000s	No limit	Features * Classes < 10,000s

Load Data

```
// in Scala
val bInput = spark.read.format("parquet").load("/data/binary-classification")
.selectExpr("features", "cast(label as double) as label")
```

Logistic Regression

- a linear method
- combines each of the individual features with specific weights
- outputs a probability of belonging to a particular class

Model Hyperparameters

hyperparameter	**values**	**Description**
family	multinomial, binary	

Model Hyperparameters

hyperparameter	**values**	**Description**
`family`	`multinomial, binary`	
`elasticNetParam`	`0.0 to 1.0`	Specifies the mix between **L1** and **L2** regularizations. 1 means L1 (sparse) and 0 means L2.

Model Hyperparameters

hyperparameter	**values**	**Description**
`family`	`multinomial, binary`	
`elasticNetParam`	`0.0 to 1.0`	Specifies the mix between **L1** and **L2** regularizations. 1 means L1 (sparse) and 0 means L2.
`fitIntercept`	`true, false`	true for un-normalized data

Model Hyperparameters

hyperparameter	**values**	**Description**
`family`	`multinomial, binary`	
`elasticNetParam`	`0.0 to 1.0`	Specifies the mix between **L1** and **L2** regularizations. 1 means L1 (sparse) and 0 means L2.
`fitIntercept`	`true, false`	true for un-normalized data
`regParam`	`>= 0.0`	Weight of regularization term corresponds to **λ**

Model Hyperparameters

hyperparameter	**values**	**Description**
`family`	`multinomial, binary`	
`elasticNetParam`	`0.0 to 1.0`	Specifies the mix between **L1** and **L2** regularizations. 1 means L1 (sparse) and 0 means L2.
`fitIntercept`	`true, false`	true for un-normalized data
`regParam`	`>= 0.0`	Weight of regularization term
`standardization`	`true, false`	

Training Parameters

parameters	**value**	**Description**
`maxIter`	`100 (default)`	Total number of iterations over the data

Training Parameters

parameters	**value**	**Description**
`maxIter`	`100 (default)`	Total number of iterations over the data
`tol`	`1.0 E-6 (default)`	Threshold to indicate sufficient optimization

Training Parameters

parameters	**value**	**Description**
maxIter	100 (default)	Total number of iterations over the data
tol	1.0 E-6 (default)	Threshold to indicate sufficient optimization
weightCol	name of a column	weigh the labels that are correct more than the others

Prediction Parameters

Parameters	**value**	**description**
threshold	0.0 to 1.0	threshold probability for deciding a class

Prediction Parameters

Parameters	**value**	**description**
`threshold`	`0.0 to 1.0`	threshold probability for deciding a class
`thresholds`	`array of threshold values`	threshold probabilities for multiclass classification problem

Example

```
import org.apache.spark.ml.classification.LogisticRegression
val lr = new LogisticRegression()
println(lr.explainParams()) // see all parameters
val lrModel = lr.fit(bInput)
```

Example

```
import org.apache.spark.ml.classification.LogisticRegression
val lr = new LogisticRegression()
println(lr.explainParams()) // see all parameters
val lrModel = lr.fit(bInput)

println(lrModel.coefficients)
println(lrModel.intercept)
```

Example

```
import org.apache.spark.ml.classification.LogisticRegression
val lr = new LogisticRegression()
println(lr.explainParams()) // see all parameters
val lrModel = lr.fit(bInput)

println(lrModel.coefficients)
println(lrModel.intercept)

import org.apache.spark.ml.classification.BinaryLogisticRegressionSummary
val summary = lrModel.summary
val bSummary = summary.asInstanceOf[BinaryLogisticRegressionSummary]
println(bSummary.areaUnderROC)
bSummary.roc.show()
bSummary.pr.show()
```

Example

```
import org.apache.spark.ml.classification.LogisticRegression
val lr = new LogisticRegression()
println(lr.explainParams()) // see all parameters
val lrModel = lr.fit(bInput)

println(lrModel.coefficients)
println(lrModel.intercept)

import org.apache.spark.ml.classification.BinaryLogisticRegressionSummary
val summary = lrModel.summary
val bSummary = summary.asInstanceOf[BinaryLogisticRegressionSummary]
println(bSummary.areaUnderROC)
bSummary.roc.show()
bSummary.pr.show()

summary.objectiveHistory
```

Decision Tree

- An interpretable models for performing classification
- Simple decision models that works like humans
- Structure with all the inputs and follows a set of branches to make a prediction
- Easy to reason, easy to inspect
- Supports multiclass classification

Decision Tree

- An interpretable models for performing classification
- Simple decision models that works like humans
- Structure with all the inputs and follows a set of branches to make a prediction
- Easy to reason, easy to inspect
- Supports multiclass classification
- Overfit data quickly

Model Hyperparameters

hyperparameter	**values**	**Description**
maxDepth	Default is 5	To avoid overfitting

Model Hyperparameters

hyperparameter	**values**	**Description**
`maxDepth`	`Default is 5`	To avoid overfitting
`maxBins`	`Value >= 2 && Value >= number of categories`	More bins gives a higher level of granularity

Model Hyperparameters

hyperparameter	**values**	**Description**
`maxDepth`	`Default is 5`	To avoid overfitting
`maxBins`	`Value >= 2 && Value >= number of categories`	More bins gives a higher level of granularity
`impurity`	`entropy or gini (default)`	determine whether or not the model should **split** at a particular leaf node

Model Hyperparameters

hyperparameter	**values**	**Description**
`maxDepth`	`Default is 5`	To avoid overfitting
`maxBins`	`Value >= 2 && Value >= number of categories`	More bins gives a higher level of granularity
`impurity`	`entropy or gini (default)`	determine whether or not the model should split at a particular leaf node
`minInfoGain`	`default is zero`	determines the minimum information gain that can be used for a split

Model Hyperparameters

hyperparameter	**values**	**Description**
`maxDepth`	`Default is 5`	To avoid overfitting
`maxBins`	`Value >= 2 && Value >= number of categories`	More bins gives a higher level of granularity
`impurity`	`entropy or gini (default)`	determine whether or not the model should split at a particular leaf node
`minInfoGain`	`default is zero`	determines the minimum information gain that can be used for a split
`minInstancePerNode`	`any value greater than 1`	controlling max depth and overfitting

Training Parameters

parameters	**value**	**Description**
`checkpointInterval`	`-1 or any value > 0`	A value of 10 means the model will get checkpointed every 10 iterations. Set this to -1 to turn off checkpointing

Prediction Parameters

Parameters	**value**	**description**
`threshold`	`0.0 to 1.0`	threshold probability for deciding a class
`thresholds`	`array of threshold values`	threshold probabilities for multiclass classification problem

Example

```
// in Scala
import
org.apache.spark.ml.classification.DecisionTreeClassifier
val dt = new DecisionTreeClassifier()
println(dt.explainParams())
val dtModel = dt.fit(bInput)
```

Random Forest and Gradient Boosted Trees

- extensions of decision trees
- wisdom of the crowds

Random Forest and Gradient Boosted Trees

- extensions of decision trees
- wisdom of the crowds
- random forest: averaging responses to make prediction
- gradient boosted tree: making weighted prediction

Model Hyperparameters

hyperparameter	**values**	**Description**
`numTrees`	`int`	number of trees to train
`featureSubsetStrategy`	`auto,all, sqrt, log2, 1<=n<=number of features`	how many features to consider for split

Model Hyperparameters

hyperparameter	**values**	**Description**
numTrees	int	number of trees to train
featureSubsetStrategy	auto,all, sqrt, log2, 1<=n<=number of features	how many features to consider for split
lossType	logistic loss (till now)	loss function to minimize
maxIter	100 (default)	number of iterations on data
stepSize	0 to 1; default 0.1	learning rate

- Unshaded parameters are only available for random forest
- Shaded parameters are only available for gradient boosted tree

Training Parameters

parameters	**value**	**Description**
checkPointInterval	same as decision trees	

Prediction Parameters

Parameters	**value**	**description**
same as decision trees		

Example (Scala)

```
import
org.apache.spark.ml.classification.RandomForestClassifier
val rfClassifier = new RandomForestClassifier()
println(rfClassifier.explainParams())
val trainedModel = rfClassifier.fit(bInput)

import org.apache.spark.ml.classification.GBTClassifier
val gbtClassifier = new GBTClassifier()
println(gbtClassifier.explainParams())
val trainedModel = gbtClassifier.fit(bInput)
```

Bayes' Theorem

Bayes' theorem is stated mathematically as the following equation:[17]

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events and $P(B) \neq 0$.

- $P(A|B)$ is a conditional probability: the probability of event A occurring given that B is true. It is also called the posterior probability of A given B.
- $P(B|A)$ is also a conditional probability: the probability of event B occurring given that A is true. It can also be interpreted as the likelihood of A given a fixed B because $P(B|A) = L(A|B)$.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B respectively without any given conditions; they are known as the prior probability and marginal probability.

https://en.wikipedia.org/wiki/Bayes%27_theorem

Naive Bayes

- A collection of classifiers based on Bayes' theorem
- All features are independent of one another (core assumption)
- Commonly used in text or document classification

Naive Bayes

- A collection of classifiers based on Bayes' theorem
- All features are independent of one another (core assumption)
- Commonly used in text or document classification
- Two different model types:
  - Multivariate **Bernoulli** model: variables represent the existence of a term in a document
  - **Multinomial** model: the total counts of terms are used
- All input features must be non-negative.

Model Hyperparameters

hyperparameter	**values**	**Description**
`modelType`	**`bernoulli`** or **`multinomial`**	To avoid overfitting
`weightCol`		Allows weighing different data points differently

Training Parameters

parameters	**value**	**Description**
smoothing	default value is 1	helps smooth out categorical data and avoid overfitting on the training data

Prediction Parameters

Parameters	**value**	**description**
`threshold`	`0.0 to 1.0`	threshold probability for deciding a class
`thresholds`	`array of threshold values`	threshold probabilities for multiclass classification problem

Example

```
// in Scala
import org.apache.spark.ml.classification.NaiveBayes
val nb = new NaiveBayes()
println(nb.explainParams())
val trainedModel = nb.fit(bInput.where("label != 0"))
```

Evaluators

- Binary Classification Metrics
  - `BinaryClassificationEvaluator:`
    - `areaUnderROC`
    - `areaUnderPR (Area under Precision Recall)`
- Multiclass Classification Metrics
  - `MulticlassClassificationEvaluator:`
    - `f1`
    - `weightedPrecision`
    - `weightedRecall`
    - `accuracy`
- Multilabel Classification Metrics

Example (Scala)

```
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
val out = model.transform(bInput)
.select("prediction", "label")
.rdd.map(x => (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double]))
val metrics = new BinaryClassificationMetrics(out)
```

Example (Scala)

```
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
val out = model.transform(bInput)
.select("prediction", "label")
.rdd.map(x => (x(0).asInstanceOf[Double], x(1).asInstanceOf[Double]))
val metrics = new BinaryClassificationMetrics(out)


metrics.areaUnderPR
metrics.areaUnderROC
println("Receiver Operating Characteristic")
metrics.roc.toDF().show()
```

One vs Rest Classifier

- Some Mllib models do not support multiclass classification
- One vs rest classifier
- Implemented as an estimator

Example (Scala)

```scala
import org.apache.spark.ml.classification.{LogisticRegression, OneVsRest}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
// load data file
val inputData = spark.read.format("libsvm")
.load("data/mllib/sample_multiclass_classification_data.txt")
// generate the train/test split.
val Array(train, test) = inputData.randomSplit(Array(0.8, 0.2))
// instantiate the base classifier
val classifier = new LogisticRegression().setMaxIter(10).setTol(1E-6).setFitIntercept(true)
// instantiate the One Vs Rest Classifier.
val ovr = new OneVsRest().setClassifier(classifier)
// train the multiclass model.
val ovrModel = ovr.fit(train)
// score the model on test data.
val predictions = ovrModel.transform(test)
// obtain evaluator.
val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")
// compute the classification error on test data.
val accuracy = evaluator.evaluate(predictions)
println(s"Test Error = ${1 - accuracy}")
```

Multilayer Perceptron

- based on neural networks
- a configurable number of layers and layer sizes
- Discuss it later.