all features and top one and four features
all features
Decision tree

```
val filename = "/home/debiansq/data/CancerPrediction_Project/data.csv"
val df=spark.read.format("csv").option("header","true").option("inferSchema","true").load(filename)
import org.apache.spark.ml.feature.RFormula
val supervised = new RFormula (). setFormula ("diagnosis ~ .-id")
val fittedRF = supervised.fit(df)
val preparedDF = fittedRF.transform(df)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))

import org.apache.spark.ml.classification.DecisionTreeClassifier
val dt = new DecisionTreeClassifier()
val dtModel = dt.fit(preparedDF)
val predictionDF =dtModel.transform(preparedDF)
val correct = predictionDF.filter($"label"===$"prediction")

val predictionTrain =dtModel.transform(train)
val predictionTest =dtModel.transform(test)

val correctTr = predictionTrain.filter($"label"===$"prediction")

val correctTe = predictionTest.filter($"label"===$"prediction")
val correct_Te= correctTe.count()

val Total_Test = predictionTest.count()
val rate = correct_Te/(Total_Test*1.0)
rate: Double = 0.979381443298969
```

Logisticregression

```
val filename = "/home/debiansq/data/CancerPrediction_Project/data.csv"
val df=spark.read.format("csv").option("header","true").option("inferSchema","true").load(filename)
import org.apache.spark.ml.feature.RFormula
val supervised = new RFormula (). setFormula ("diagnosis ~ .-id")
val fittedRF = supervised.fit(df)
val preparedDF = fittedRF.transform(df)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.LogisticRegression
val lr = new LogisticRegression()
val fittedLR= lr.fit(preparedDF)
val predictionTrain =fittedLR.transform(train)
val predictionTest =fittedLR.transform(test)
val correctTr = predictionTrain.filter($"label"===$"prediction")
val correctTe = predictionTest.filter($"label"===$"prediction")
val correct_Te= correctTe.count()
val Total_Test = predictionTest.count()
val rate = correct_Te/(Total_Test*1.0)
```

RandomForest

```
val filename = "/home/debiansq/data/CancerPrediction_Project/data.csv"
val df=spark.read.format("csv").option("header","true").option("inferSchema","true").load(filename)
import org.apache.spark.ml.feature.RFormula
val supervised = new RFormula (). setFormula ("diagnosis ~ .-id")
val fittedRF = supervised.fit(df)
val preparedDF = fittedRF.transform(df)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.RandomForestClassifier
val rfclassifer= new RandomForestClassifier()
val rfModel = rfclassifer.fit(preparedDF)
val predictionTrain =rfModel.transform(train)
val predictionTest =rfModel.transform(test)
val correctTe = predictionTest.filter($"label"===$"prediction")
val total_test = predictionTest.count()
val rate = correctTe.count()/(total_test*1.0)
```

rate: Double = 0.9884393063583815

4/27/19

Decision Tree

The best feature area_worst

```
val filename = "/home/debiansq/data/CancerPrediction_Project/data.csv"
val df1=spark.read.format("csv").option("header","true").option("inferSchema","true").load(filename)
import org.apache.spark.ml.feature.RFormula
val supervised = new RFormula (). setFormula ("diagnosis ~ area_worst")
val fittedRF = supervised.fit(df1)
val preparedDF = fittedRF.transform(df1)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.DecisionTreeClassifier
val dt = new DecisionTreeClassifier()
val dtModel = dt.fit(preparedDF)
val predictionDF =dtModel.transform(preparedDF)
val correct = predictionDF.filter($"label"===$"prediction")
val predictionTrain =dtModel.transform(train)
val predictionTest =dtModel.transform(test)
val correctTr = predictionTrain.filter($"label"===$"prediction")
val correctTe = predictionTest.filter($"label"===$"prediction")
val correct_Te= correctTe.count()
val Total_Test = predictionTest.count()
val rate = correct_Te/(Total_Test*1.0)
```

rate: Double = 0.8870967741935484

The four best feature area_worst+area_mean+area_se+perimeter_worst"

```
val supervised = new RFormula (). setFormula ("diagnosis ~
area_worst+area_mean+area_se+perimeter_worst")
val fittedRF = supervised.fit(df1)
val preparedDF = fittedRF.transform(df1)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.DecisionTreeClassifier
val dt = new DecisionTreeClassifier()
val dtModel = dt.fit(preparedDF)
val predictionDF =dtModel.transform(preparedDF)
val correct = predictionDF.filter($"label"===$"prediction")
val predictionTrain =dtModel.transform(train)
val predictionTest =dtModel.transform(test)
val correctTr = predictionTrain.filter($"label"===$"prediction")
val correctTe = predictionTest.filter($"label"===$"prediction")
val correct_Te= correctTe.count()
val Total_Test = predictionTest.count()
val rate = correct_Te/(Total_Test*1.0)
rate: Double = 0.9502762430939227
```

LOGISTIC Regression

The best feature

```
val supervised = new RFormula (). setFormula ("diagnosis ~ area_worst")
val fittedRF = supervised.fit(df1)
val preparedDF = fittedRF.transform(df1)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.LogisticRegression
val lr = new LogisticRegression()
val fittedLR= lr.fit(preparedDF)
val predictionTrain =fittedLR.transform(train)
val predictionTest =fittedLR.transform(test)
val correctTr = predictionTrain.filter($"label"===$"prediction")
val correctTe = predictionTest.filter($"label"===$"prediction")
val correct_Te= correctTe.count()
val Total_Test = predictionTest.count()
val rate = correct_Te/(Total_Test*1.0)
rate: Double = 0.9329608938547486
```

The best four features

```
val supervised = new RFormula (). setFormula ("diagnosis ~
area_worst+area_mean+area_se+perimeter_worst")
val fittedRF = supervised.fit(df1)
val preparedDF = fittedRF.transform(df1)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.LogisticRegression
val lr = new LogisticRegression()
```

```
val fittedLR= lr.fit(preparedDF)
val predictionTrain =fittedLR.transform(train)
val predictionTest =fittedLR.transform(test)
val correctTr = predictionTrain.filter($"label"===$"prediction")
val correctTe = predictionTest.filter($"label"===$"prediction")
val correct_Te= correctTe.count()
val Total_Test = predictionTest.count()
val rate = correct_Te/(Total_Test*1.0)
rate: Double = 0.9197860962566845
```

RandomForest

The best feature

```
val supervised = new RFormula (). setFormula ("diagnosis ~ area_worst")
val fittedRF = supervised.fit(df1)
val preparedDF = fittedRF.transform(df1)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.RandomForestClassifier
val rfclassifer= new RandomForestClassifier()
val rfModel = rfclassifer.fit(preparedDF)
val predictionTrain =rfModel.transform(train)
val predictionTest =rfModel.transform(test)
val correctTe = predictionTest.filter($"label"===$"prediction")
val total_test = predictionTest.count()
val rate = correctTe.count()/(total_test*1.0)
rate: Double = 0.8918918918918919
```

The best four features

```
val supervised = new RFormula (). setFormula ("diagnosis ~
area_worst+area_mean+area_se+perimeter_worst")
val fittedRF = supervised.fit(df1)
val preparedDF = fittedRF.transform(df1)
val Array (train, test) = preparedDF.randomSplit(Array(0.67,0.33))
import org.apache.spark.ml.classification.RandomForestClassifier
val rfclassifer= new RandomForestClassifier()
val rfModel = rfclassifer.fit(preparedDF)
val predictionTrain =rfModel.transform(train)
val predictionTest =rfModel.transform(test)
val correctTe = predictionTest.filter($"label"===$"prediction")
val total_test = predictionTest.count()
val rate = correctTe.count()/(total_test*1.0)
rate: Double = 0.978494623655914
```