Using Machine learning to predict the Breast Cancer recurrence

Team 25:
Haoning Xia
Tianyi Liu
Menghao Huo

Introduction

Breast cancer is a cancer disease that develops from breast tissue. Signs of breast cancer may include a lump in the breast, a change in breast shape, dimpling of the skin, fluid coming from the nipple, a newly-inverted nipple, or a red or scaly patch of skin. In this experiment, we will research how factors can affect the recurrence of breast cancer such as: age, menopause, tumor-size, inv-nodes, node-caps, degree of malignancy, breast, breast quadrant, irradiate.

Motivation

Problem design:

1. Which affect the recurrence of a patient. Analyze the impact on recurrence event by controlling different variables.
2. How to use the most important attributes of breast cancer to predict the recurrence event.
3. Which machine learning model gives the most accurate prediction result.

Dataset Overview

	class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
256	recurrence-events	40-49	premeno	30-34	0-2	no	1	left	left_low	yes
257	recurrence-events	40-49	premeno	20-24	3-5	yes	2	left	left_low	yes
258	recurrence-events	50-59	ge40	30-34	6-8	yes	2	left	right_low	yes
259	recurrence-events	50-59	ge40	30-34	3-5	no	3	right	left_up	no
260	recurrence-events	60-69	ge40	25-29	3-5	no	2	right	right_up	no
261	recurrence-events	40-49	ge40	25-29	12-14	yes	3	left	right_low	yes
262	recurrence-events	60-69	ge40	25-29	0-2	no	3	left	left_up	no
263	recurrence-events	50-59	lt40	20-24	0-2	?	1	left	left_up	no
264	recurrence-events	50-59	lt40	20-24	0-2	?	1	left	left_low	no
265	recurrence-events	30-39	premeno	35-39	9-11	yes	3	left	left_low	no

Missing Value:

```
node-caps: 8 records
breast-quad: 1 records
```

Two Methods:

1.Drop all rolls which have "?" in them

2.Replace the "?" with the most occurrence value of that column.

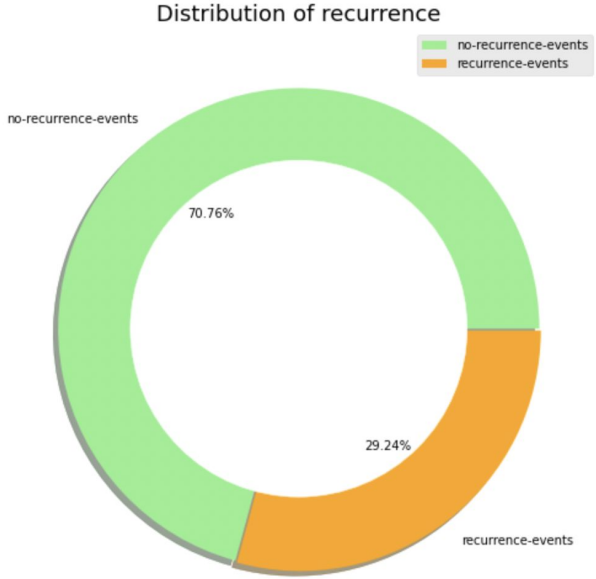
(286,10) -> (277,10)

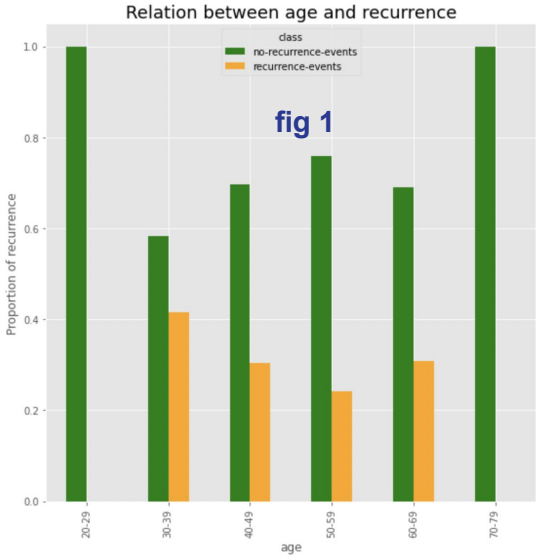
statistics:

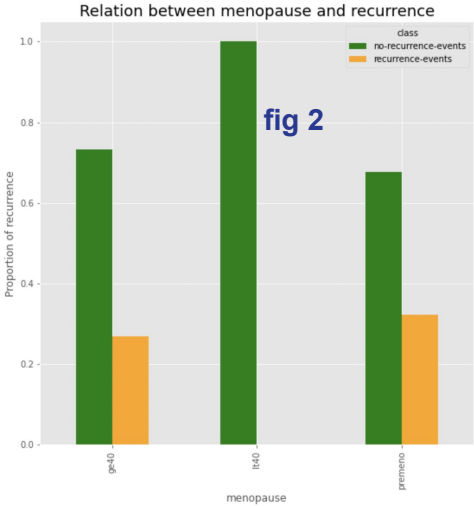
	class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
count	286	286	286	286	286	286	286	286	286	286
unique	2	6	3	11	7	3	3	2	6	2
top	no-recurrence-events	50-59	premeno	30-34	0-2	no	2	left	left_low	no
freq	201	96	150	60	213	222	130	152	110	218

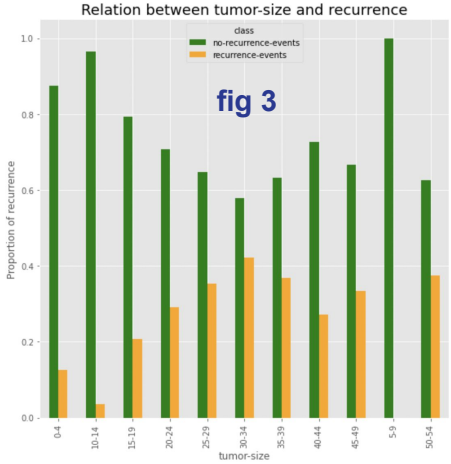
Data Visualization

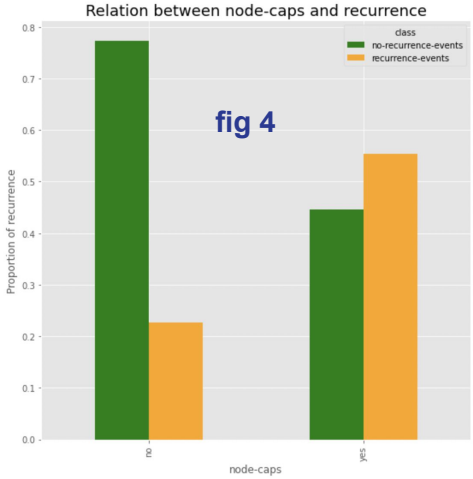
```
no-recurrence-events    196
recurrence-events        81
Name: class, dtype: int64
```

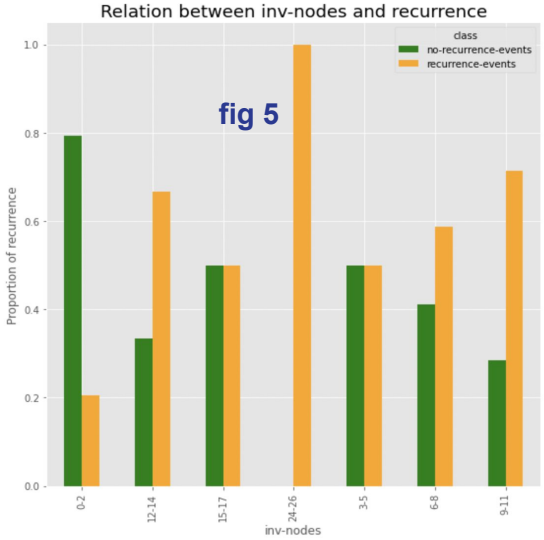


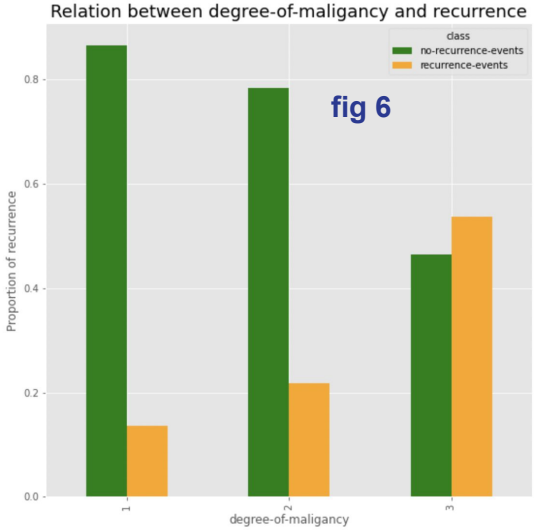


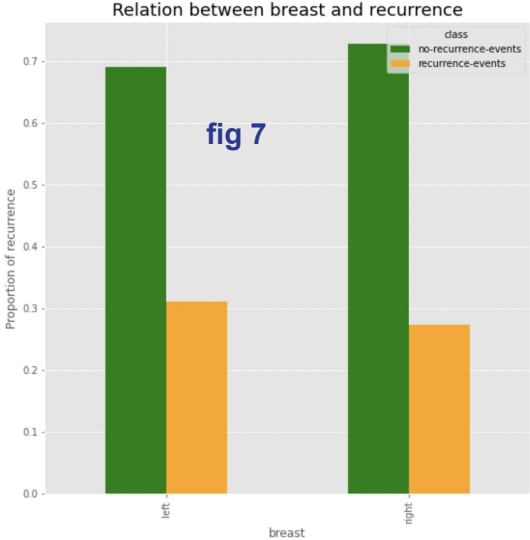


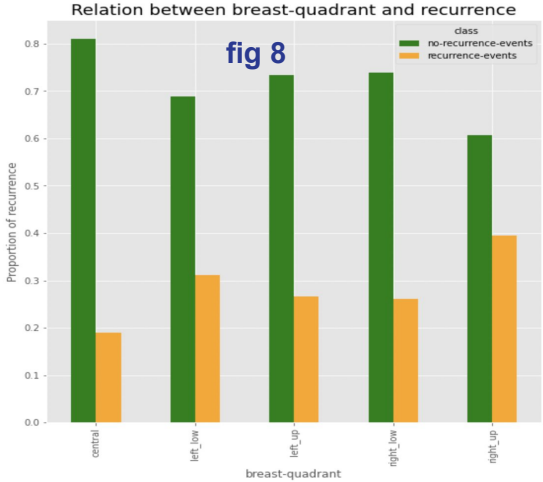


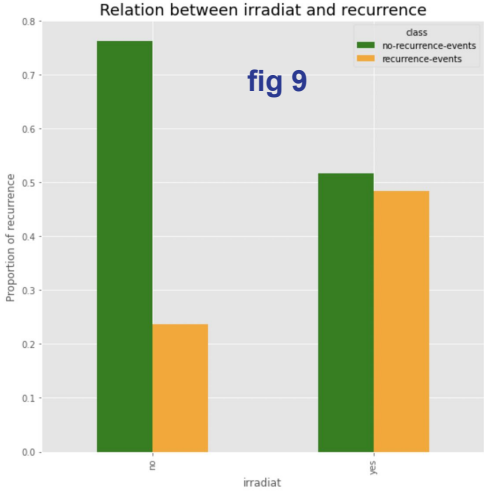



Relation between inv-nodes and recurrence
fig 5
class
no-recurrence-events
recurrence-events
Proportion of recurrence
0.0
0.2
0.4
0.6
0.8
1.0
0-2
12-14
15-17
24-26
3-5
6-8
9-11
inv-nodes


Relation between degree-of-maligancy and recurrence
fig 6
class
no-recurrence-events
recurrence-events
Proportion of recurrence
0.0
0.2
0.4
0.6
0.8
1
2
3
degree-of-maligancy


Relation between breast and recurrence
fig 7
class
no-recurrence-events
recurrence-events
Proportion of recurrence
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
left
right
breast


Relation between breast-quadrant and recurrence
fig 8
class
no-recurrence-events
recurrence-events
Proportion of recurrence
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
central
left_low
left_up
right_low
right_up
breast-quadrant


Relation between irradiat and recurrence
fig 9
class
no-recurrence-events
recurrence-events
Proportion of recurrence
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
no
yes
irradiat

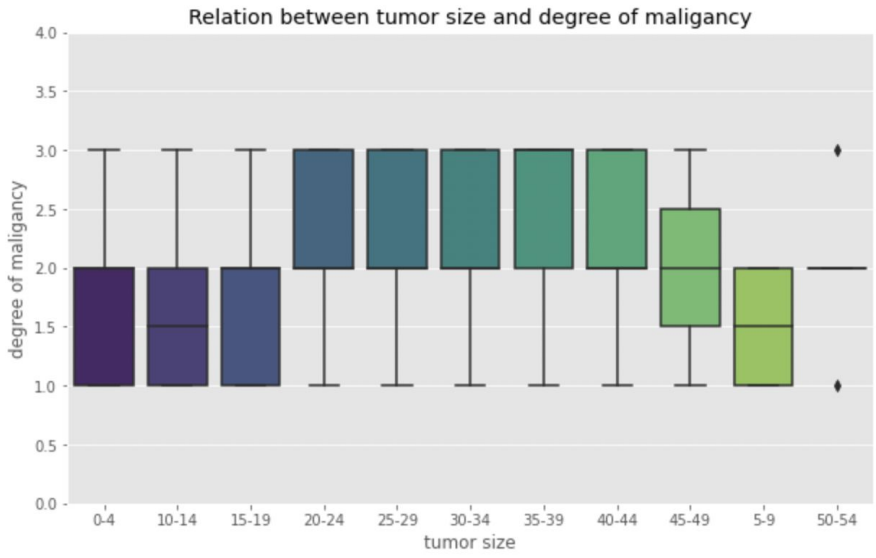


fig 1

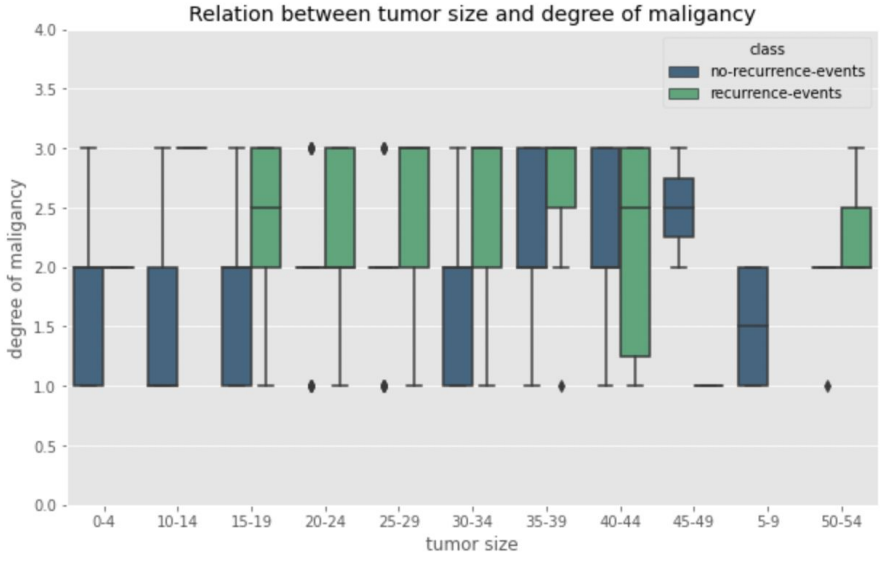


fig 2

Tumor size within 20-44 is more likely to have a high degree of malignancy

Tumor size within 20-44 and the degree of malignancy is high are more likely to recurrence

Data Preprocessing

- loading data as RDD data type .

```
rawData = sc.textFile("breast-cancer.csv")
lines = rawData.map(lambda x: x.split(","))
```

- Delete the line that have unvalid data

```
def delete(line):
    for value in line:
        if (value == "?"): return 1
    return 0
```

```
linesDel = lines.filter(lambda line: dellect(line) != 1)
```

Data Preprocessing

Before:

```
>>> pprint(linesDel.take(5))
[[u'no-recurrence-events',
  u'30-39',
  u'premeno',
  u'30-34',
  u'0-2',
  u'no',
  u'3',
  u'left',
  u'left_low',
  u'no'],
 [u'no-recurrence-events',
  u'40-49',
  u'premeno',
  u'20-24',
  u'0-2',
  u'no',
  u'2',
  u'right',
  u'right_up',
  u'no'],
```

After:

```
>>> pprint(lineTransF.take(5))
[[0.0, 3.0, 2.0, 30.0, 0.0, 0.0, 3.0, 0.0, 22.0, 0.0],
 [0.0, 4.0, 2.0, 20.0, 0.0, 0.0, 2.0, 1.0, 22.0, 0.0],
 [0.0, 4.0, 2.0, 20.0, 0.0, 0.0, 2.0, 0.0, 22.0, 0.0],
 [0.0, 6.0, 1.0, 15.0, 0.0, 0.0, 2.0, 1.0, 22.0, 0.0],
 [0.0, 4.0, 2.0, 0.0, 0.0, 0.0, 2.0, 1.0, 22.0, 0.0]]
>>>
```

Labelpoint data type:

```
>>> pprint(labelpointRDD.take(5))
[Stage 74:>

[LabeledPoint(0.0, [3.0,2.0,30.0,0.0,0.0,3.0,0.0,22.0,0.0]),
 LabeledPoint(0.0, [4.0,2.0,20.0,0.0,0.0,2.0,1.0,22.0,0.0]),
 LabeledPoint(0.0, [4.0,2.0,20.0,0.0,0.0,2.0,0.0,22.0,0.0]),
 LabeledPoint(0.0, [6.0,1.0,15.0,0.0,0.0,2.0,1.0,22.0,0.0]),
 LabeledPoint(0.0, [4.0,2.0,0.0,0.0,0.0,2.0,1.0,22.0,0.0])]
>>>
```

Naive Bayes

```
training, test = labelpointRDD.randomSplit([0.7, 0.3])
model = NaiveBayes.train(training, 1.0)
predictionAndLabel = test.map(lambda p: (model.predict(p.features), p.label))
accuracy = 1.0 * predictionAndLabel.filter(lambda pl: pl[0] == pl[1]).count() / test.count()
print('model accuracy {}'.format(accuracy))
```

```
>>> predictionAndLabel = test.map(lambda p: (model.predict(p.features), p.label)
)
>>> accuracy = 1.0 * predictionAndLabel.filter(lambda pl: pl[0] == pl[1]).count(
) / test.count()
>>> print('model accuracy {}'.format(accuracy))
model accuracy 0.753246753247
>>>
```

Random Forests

```
>>> (trainingData, testData) = labelpointRDD.randomSplit([0.7, 0.3])
>>> model = RandomForest.trainClassifier(trainingData, numClasses=4, categorical
FeaturesInfo={},numTrees=3, featureSubsetStrategy="auto",impurity='gini', maxDep
th=4, maxBins=32)
>>> predictions = model.predict(testData.map(lambda x: x.features))
>>> labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
>>> testErr = labelsAndPredictions.filter(lambda lp: lp[0] != lp[1]).count() / f
loat(testData.count())
>>> print('Test Error = ' + str(testErr))
Test Error = 0.233333333333
>>> print('Learned classification forest model:')
Learned classification forest model:
```

Linear Support Vector Machines (SVMs)

```
>>>
>>> model = SVMWithSGD.train(labelpointRDD, iterations=100)
>>> labelsAndPreds = labelpointRDD.map(lambda p: (p.label, model.predict(p.features)))
>>> trainAcc = labelsAndPreds.filter(lambda lp: lp[0] == lp[1]).count() / float(labelpointRDD.count())
>>> trainAcc
0.7075812274368231
```

```
>>> model = SVMWithSGD.train(labelpointRDD, iterations=100)
>>> labelsAndPreds = labelpointRDD.map(lambda p: (p.label, model.predict(p.features)))
>>> trainAcc = labelsAndPreds.filter(lambda lp: lp[0] == lp[1]).count() / float(labelpointRDD.count())
>>> trainAcc
0.7075812274368231
>>> model = SVMWithSGD.train(labelpointRDD, iterations=1)
>>> labelsAndPreds = labelpointRDD.map(lambda p: (p.label, model.predict(p.features)))
>>> trainAcc = labelsAndPreds.filter(lambda lp: lp[0] == lp[1]).count() / float(labelpointRDD.count())
>>> trainAcc
0.7075812274368231
>>> model = SVMWithSGD.train(labelpointRDD, iterations=10)
>>> labelsAndPreds = labelpointRDD.map(lambda p: (p.label, model.predict(p.features)))
>>> trainAcc = labelsAndPreds.filter(lambda lp: lp[0] == lp[1]).count() / float(labelpointRDD.count())
>>> trainAcc
0.3285198555956679
>>> model = SVMWithSGD.train(labelpointRDD, iterations=1000)
>>> labelsAndPreds = labelpointRDD.map(lambda p: (p.label, model.predict(p.features)))
>>> trainAcc = labelsAndPreds.filter(lambda lp: lp[0] == lp[1]).count() / float(labelpointRDD.count())
>>> trainAcc
0.7364620938628159
```

Decision Tree

```
>>> from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
>>> from pyspark.mllib.util import MLUtils
>>> (trainingData, testData) = labelpointRDD.randomSplit([0.7, 0.3])
>>> trainingData.count()
195
>>> testData.count()
82
>>> model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
...                                      impurity='gini', maxDepth=5, maxBins=32)
>>> model
DecisionTreeModel classifier of depth 5 with 37 nodes
>>> predictions = model.predict(testData.map(lambda x: x.features))
>>> labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
>>> testErr = labelsAndPredictions.filter(
...  lambda lp: lp[0] != lp[1]).count() / float(testData.count())
>>> testErr
0.35365853658536583
>>> testAcc = labelsAndPredictions.filter(
...  lambda lp: lp[0] == lp[1]).count() / float(testData.count())
>>> testAcc
0.6463414634146342
>>> model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
...                                      impurity='entropy', maxDepth=5, maxBins=32)
>>>
>>> predictions = model.predict(testData.map(lambda x: x.features))
>>> labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
>>> testAcc = labelsAndPredictions.filter(
...  lambda lp: lp[0] == lp[1]).count() / float(testData.count())
>>> testAcc
0.6219512195121951
```

Result Compare

SVM : 70%~ accuracy

Decision Tree: 60%~ accuracy

Naive Bayes: 75%~ accuracy

Random Forests: 77%~ accuracy