In [230]:
```
import Pkg;
#Pkg.add("CSV")
#Pkg.add("Random")
#Pkg.add("PyPlot")
#Pkg.add("MLLabelUtils")
#Pkg.add("MLDataUtils")
#Pkg.add("DecisionTree")
#Pkg.add("ScikitLearn")
#Pkg.add("XGBoost")
#Pkg.add("PyCall")
#Pkg.add("MLBase")
#Pkg.add("LearnBase")
```

Diagnosing Breast Cancer using Julia

Loading necessary packages and reading the [dataset (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data)](https://www.kaggle.com/uciml/breast-cancer-wisconsin-data).

In [231]:
```
using CSV
using DataFrames
df = CSV.read("wdbc.csv");
```

Setting Random Seed to produce same output everytime, it helps in debugging

In [232]:
```
RANDOM_STATE = 42
using Random
Random.seed!(RANDOM_STATE)
```

Out[232]:
```
MersenneTwister(UInt32[0x0000002a], Random.DSFMT.DSFMT_state(Int32[964434469, 1073036706, 1860149520, 107350
3458, 1687169063, 1073083486, -399267803, 1072983952, -909620556, 1072836235  …  -293054293, 1073002412, -13
00127419, 1073642642, 1917177374, -666058738, -337596527, 1830741494, 382, 0]), [0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], UInt128[0x000000000000000000000
00000000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0x000000000000000000000
00000000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0x000000000000000000000
00000000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0x000000000000000000000
00000000000  …  0x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0x000000000000000000
00000000000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0x000000000000000000
00000000000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000000000, 0x000000000000000000
00000000000000, 0x00000000000000000000000000000000], 1002, 0)
```

First 5 samples of the dataset

In [233]:
```
first(df,5)
```

Out[233]: 5 rows × 32 columns (omitted printing of 25 columns)

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
	Int64	String	Float64	Float64	Float64	Float64	Float64
1	842302	M	17.99	10.38	122.8	1001.0	0.1184
2	842517	M	20.57	17.77	132.9	1326.0	0.08474
3	84300903	M	19.69	21.25	130.0	1203.0	0.1096
4	84348301	M	11.42	20.38	77.58	386.1	0.1425
5	84358402	M	20.29	14.34	135.1	1297.0	0.1003

Printing the number of instances and number of columns

In [234]:
```
# size function returns the number of row and columns of the dataset
row, col = size(df)
println("Number of Instances: ", row)
println("Number of Columns: ", col)
```

```
Number of Instances: 569
Number of Columns: 32
```

Headers / Column names of the dataset

In [235]:
```
println(names(df))
```

```
Symbol[:id, :diagnosis, :radius_mean, :texture_mean, :perimeter_mean, :area_mean, :smoothness_mean, :compactness_mean, :concavity_mean, Symbol("concave points_mean"), :symmetry_mean, :fractal_dimension_mean, :radius_se, :texture_se, :perimeter_se, :area_se, :smoothness_se, :compactness_se, :concavity_se, Symbol("concave points_se"), :symmetry_se, :fractal_dimension_se, :radius_worst, :texture_worst, :perimeter_worst, :area_worst, :smoothness_worst, :compactness_worst, :concavity_worst, Symbol("concave points_worst"), :symmetry_worst, :fractal_dimension_worst]
```

Description of dataset

In [236]:
```
println(describe(df))
```

32×8 DataFrame

Row	variable	mean	min	median	max	nunique	nmissing	eltype
	Symbol	Union…	Any	Union…	Any	Union…	Nothing	DataType
1	id	3.03718e7	8670	906024.0	911320502			Int64
2	diagnosis		B		M	2		String
3	radius_mean	14.1273	6.981	13.37	28.11			Float64
4	texture_mean	19.2896	9.71	18.84	39.28			Float64
5	perimeter_mean	91.969	43.79	86.24	188.5			Float64
6	area_mean	654.889	143.5	551.1	2501.0			Float64
7	smoothness_mean	0.0963603	0.05263	0.09587	0.1634			Float64
8	compactness_mean	0.104341	0.01938	0.09263	0.3454			Float64
9	concavity_mean	0.0887993	0.0	0.06154	0.4268			Float64
10	concave points_mean	0.0489191	0.0	0.0335	0.2012			Float64
11	symmetry_mean	0.181162	0.106	0.1792	0.304			Float64
12	fractal_dimension_mean	0.0627976	0.04996	0.06154	0.09744			Float64
13	radius_se	0.405172	0.1115	0.3242	2.873			Float64
14	texture_se	1.21685	0.3602	1.108	4.885			Float64
15	perimeter_se	2.86606	0.757	2.287	21.98			Float64
16	area_se	40.3371	6.802	24.53	542.2			Float64
17	smoothness_se	0.00704098	0.001713	0.00638	0.03113			Float64
18	compactness_se	0.0254781	0.002252	0.02045	0.1354			Float64
19	concavity_se	0.0318937	0.0	0.02589	0.396			Float64
20	concave points_se	0.0117961	0.0	0.01093	0.05279			Float64
21	symmetry_se	0.0205423	0.007882	0.01873	0.07895			Float64
22	fractal_dimension_se	0.0037949	0.0008948	0.003187	0.02984			Float64
23	radius_worst	16.2692	7.93	14.97	36.04			Float64
24	texture_worst	25.6772	12.02	25.41	49.54			Float64
25	perimeter_worst	107.261	50.41	97.66	251.2			Float64
26	area_worst	880.583	185.2	686.5	4254.0			Float64
27	smoothness_worst	0.132369	0.07117	0.1313	0.2226			Float64
28	compactness_worst	0.254265	0.02729	0.2119	1.058			Float64
29	concavity_worst	0.272188	0.0	0.2267	1.252			Float64
30	concave points_worst	0.114606	0.0	0.09993	0.291			Float64
31	symmetry_worst	0.290076	0.1565	0.2822	0.6638			Float64
32	fractal_dimension_worst	0.0839458	0.05504	0.08004	0.2075			Float64

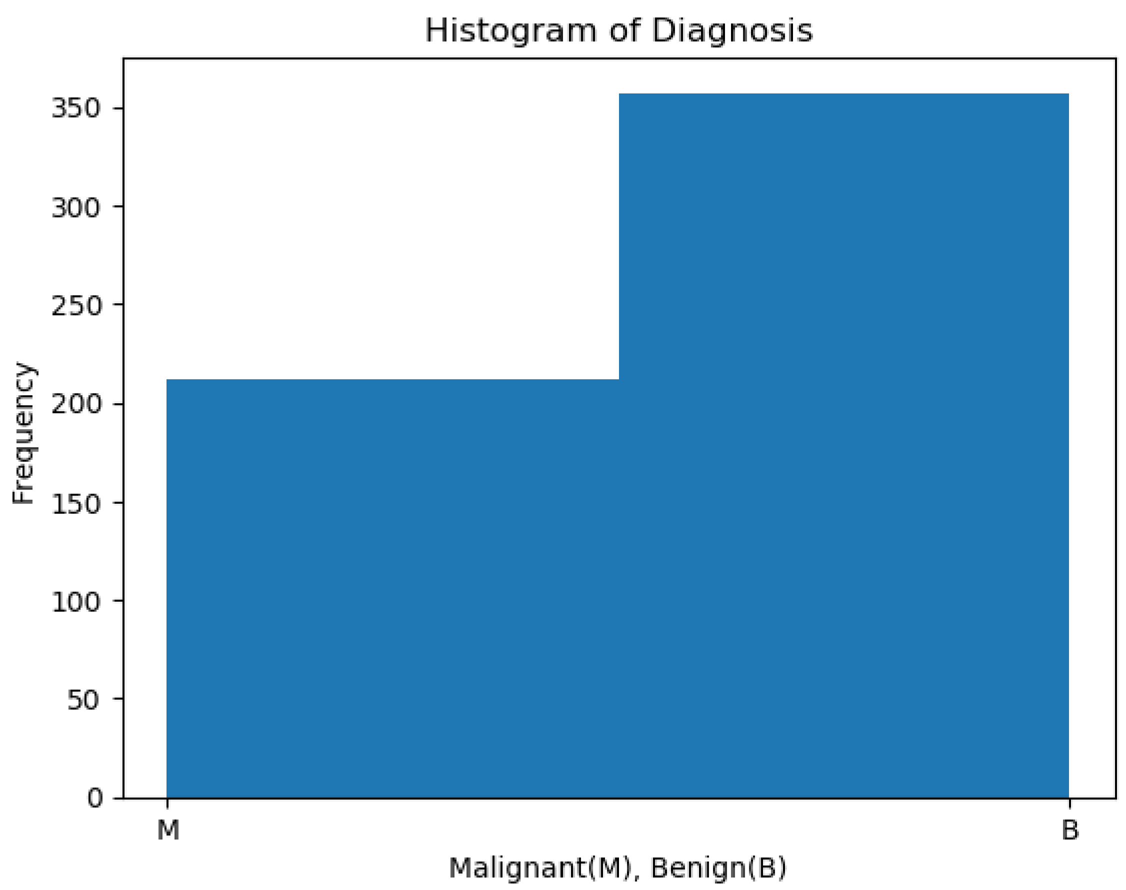
Frequencey of each class

In [237]:
```
using FreqTables
frequency_digonosis = freqtable(df[:diagnosis])
```

Out[237]:
```
2-element Named Array{Int64,1}
Dim1  │
──────┼─────
B     │ 357
M     │ 212
```

Histogram of the frequency of each class

In [238]:
```
using PyPlot
PyPlot.plt.hist(df[:diagnosis], 2)
plt.title("Histogram of Diagnosis")
plt.xlabel("Malignant(M), Benign(B)")
plt.ylabel("Frequency")
```



Out[238]:
```
PyObject Text(24.000000000000007, 0.5, 'Frequency')
```

Data Preprocessing

Converting the categorical feature "diagnosis" to integer by doing label encoding

In [239]:
```
using MLLabelUtils
y = convertlabel(LabelEnc.MarginBased,df[:diagnosis])
y = classify.(y, LabelEnc.ZeroOne(Int,1))
newdf = copy(df)
deletecols!(newdf, :diagnosis)
newdf[:diagnosis] = y;
```

In [240]:
```
#deletecols!(newdf, :id)
```

Dataset after encodning

In [241]:
```
first(newdf,5)
```

Out[241]: 5 rows × 32 columns (omitted printing of 26 columns)

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
	Int64	Float64	Float64	Float64	Float64	Float64
1	842302	17.99	10.38	122.8	1001.0	0.1184
2	842517	20.57	17.77	132.9	1326.0	0.08474
3	84300903	19.69	21.25	130.0	1203.0	0.1096
4	84348301	11.42	20.38	77.58	386.1	0.1425
5	84358402	20.29	14.34	135.1	1297.0	0.1003

Description of Dataset after encodning

In [242]:
```
println(describe(newdf))
```

```
32×8 DataFrame
│ Row │ variable                │ mean       │ min       │ median   │ max       │ nunique │ nmissing │ eltyp
e   │
│     │ Symbol                  │ Float64    │ Real      │ Float64  │ Real      │ Nothing │ Nothing  │ DataT
ype │
├─────┼─────────────────────────┼────────────┼───────────┼──────────┼───────────┼─────────┼──────────┼──────
────┤
│ 1   │ id                      │ 3.03718e7  │ 8670      │ 906024.0 │ 911320502 │         │          │ Int64
│
│ 2   │ radius_mean             │ 14.1273    │ 6.981     │ 13.37    │ 28.11     │         │          │ Float
64  │
│ 3   │ texture_mean            │ 19.2896    │ 9.71      │ 18.84    │ 39.28     │         │          │ Float
64  │
│ 4   │ perimeter_mean          │ 91.969     │ 43.79     │ 86.24    │ 188.5     │         │          │ Float
64  │
│ 5   │ area_mean               │ 654.889    │ 143.5     │ 551.1    │ 2501.0    │         │          │ Float
64  │
│ 6   │ smoothness_mean         │ 0.0963603  │ 0.05263   │ 0.09587  │ 0.1634    │         │          │ Float
64  │
│ 7   │ compactness_mean        │ 0.104341   │ 0.01938   │ 0.09263  │ 0.3454    │         │          │ Float
64  │
│ 8   │ concavity_mean          │ 0.0887993  │ 0.0       │ 0.06154  │ 0.4268    │         │          │ Float
64  │
│ 9   │ concave points_mean     │ 0.0489191  │ 0.0       │ 0.0335   │ 0.2012    │         │          │ Float
64  │
│ 10  │ symmetry_mean           │ 0.181162   │ 0.106     │ 0.1792   │ 0.304     │         │          │ Float
64  │
│ 11  │ fractal_dimension_mean  │ 0.0627976  │ 0.04996   │ 0.06154  │ 0.09744   │         │          │ Float
64  │
│ 12  │ radius_se               │ 0.405172   │ 0.1115    │ 0.3242   │ 2.873     │         │          │ Float
64  │
│ 13  │ texture_se              │ 1.21685    │ 0.3602    │ 1.108    │ 4.885     │         │          │ Float
64  │
│ 14  │ perimeter_se            │ 2.86606    │ 0.757     │ 2.287    │ 21.98     │         │          │ Float
64  │
│ 15  │ area_se                 │ 40.3371    │ 6.802     │ 24.53    │ 542.2     │         │          │ Float
64  │
│ 16  │ smoothness_se           │ 0.00704098 │ 0.001713  │ 0.00638  │ 0.03113   │         │          │ Float
64  │
│ 17  │ compactness_se          │ 0.0254781  │ 0.002252  │ 0.02045  │ 0.1354    │         │          │ Float
64  │
│ 18  │ concavity_se            │ 0.0318937  │ 0.0       │ 0.02589  │ 0.396     │         │          │ Float
64  │
│ 19  │ concave points_se       │ 0.0117961  │ 0.0       │ 0.01093  │ 0.05279   │         │          │ Float
64  │
│ 20  │ symmetry_se             │ 0.0205423  │ 0.007882  │ 0.01873  │ 0.07895   │         │          │ Float
64  │
│ 21  │ fractal_dimension_se    │ 0.0037949  │ 0.0008948 │ 0.003187 │ 0.02984   │         │          │ Float
64  │
│ 22  │ radius_worst            │ 16.2692    │ 7.93      │ 14.97    │ 36.04     │         │          │ Float
64  │
│ 23  │ texture_worst           │ 25.6772    │ 12.02     │ 25.41    │ 49.54     │         │          │ Float
64  │
│ 24  │ perimeter_worst         │ 107.261    │ 50.41     │ 97.66    │ 251.2     │         │          │ Float
64  │
│ 25  │ area_worst              │ 880.583    │ 185.2     │ 686.5    │ 4254.0    │         │          │ Float
64  │
│ 26  │ smoothness_worst        │ 0.132369   │ 0.07117   │ 0.1313   │ 0.2226    │         │          │ Float
64  │
│ 27  │ compactness_worst       │ 0.254265   │ 0.02729   │ 0.2119   │ 1.058     │         │          │ Float
64  │
│ 28  │ concavity_worst         │ 0.272188   │ 0.0       │ 0.2267   │ 1.252     │         │          │ Float
64  │
│ 29  │ concave points_worst    │ 0.114606   │ 0.0       │ 0.09993  │ 0.291     │         │          │ Float
64  │
│ 30  │ symmetry_worst          │ 0.290076   │ 0.1565    │ 0.2822   │ 0.6638    │         │          │ Float
64  │
│ 31  │ fractal_dimension_worst │ 0.0839458  │ 0.05504   │ 0.08004  │ 0.2075    │         │          │ Float
64  │
│ 32  │ diagnosis               │ 0.372583   │ 0         │ 0.0      │ 1         │         │          │ Int64
│
```

Spliting the dataset into train, test

Shuffling the Dataset

In [243]:
```
using MLDataUtils
newdf_s = shuffleobs(newdf);
first(newdf_s,10)
```

Out[243]: 10 rows × 32 columns (omitted printing of 26 columns)

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
	Int64	Float64	Float64	Float64	Float64	Float64
1	914333	14.87	20.21	96.12	680.9	0.09587
2	864292	10.51	20.19	68.64	334.2	0.1122
3	903554	12.1	17.72	78.07	446.2	0.1029
4	894604	10.25	16.18	66.52	324.2	0.1061
5	86517	18.66	17.12	121.4	1077.0	0.1054
6	894329	9.042	18.9	60.07	244.5	0.09968
7	89382601	14.61	15.69	92.68	664.9	0.07618
8	891923	13.77	13.27	88.06	582.7	0.09198
9	915452	16.3	15.7	104.7	819.8	0.09427
10	917897	9.847	15.68	63.0	293.2	0.09492

Spliting into train and test

In [244]:
```
train, test = splitobs(newdf_s, at = 0.70);
```

Seperating the features and class label

In [245]:
```
row,col = size(train)
```

Out[245]:
```
(398, 32)
```

In [246]:
```
X_train = convert(Matrix, train[:,2:col-1]);
y_train = convert(Matrix, train[:,col:col]);
X_test = convert(Matrix, test[:,2:col-1]);
y_test = convert(Matrix, test[:,col:col]);
```

Converting (n,1) dimension to (n,) as per the classifiers requirements

In [247]:
```
y_train = vec(y_train);
y_test = vec(y_test);
```

Applying algorithms to train models

RandomForest

Loading necessary packages

In [248]:
```
using DecisionTree
```

In [249]:
```
?RandomForestClassifier
```

search: **RandomForestClassifier**

Out[249]:
```
RandomForestClassifier(; n_subfeatures::Int=-1,
                       n_trees::Int=10,
                       partial_sampling::Float=0.7,
                       max_depth::Int=-1,
                       rng=Random.GLOBAL_RNG)
```

Random forest classification. See [DecisionTree.jl's documentation (https://github.com/bensadeghi/DecisionTree.jl)](https://github.com/bensadeghi/DecisionTree.jl)

Hyperparameters:

- `n_subfeatures` : number of features to consider at random per split (default: -1, sqrt(# features))
- `n_trees` : number of trees to train (default: 10)
- `partial_sampling` : fraction of samples to train each tree on (default: 0.7)
- `max_depth` : maximum depth of the decision trees (default: no maximum)
- `min_samples_leaf` : the minimum number of samples each leaf needs to have
- `min_samples_split` : the minimum number of samples in needed for a split
- `min_purity_increase` : minimum purity needed for a split
- `rng` : the random number generator to use. Can be an `Int` , which will be used to seed and create a new random number generator.

Implements `fit!` , `predict` , `predict_proba` , `get_classes`

Creating an object of Classifier

In [250]:
```
using DecisionTree
using ScikitLearn: fit!, predict
rfc = RandomForestClassifier(n_trees = 110, n_subfeatures = 15, max_depth = 7);
```

Fitting the classifiers on training data samples

In [251]:
```
fit!(rfc, X_train, y_train);
```

predicting the test data samples

In [252]:
```
y_pred_rfc = predict(rfc, X_test);
println("A portion of prediction:")
y_pred_rfc[1:10]
```

```
A portion of prediction:
```

Out[252]:
```
10-element Array{Int64,1}:
 0
 0
 0
 0
 0
 1
 1
 0
 0
 1
```

In [253]:
```
cm  = confusion_matrix(y_test, y_pred_rfc)
```

```
2×2 Array{Int64,2}:
 96   2
  7  66
```

Out[253]:
```
Classes:  [0, 1]
Matrix:
Accuracy: 0.9473684210526315
Kappa:    0.8914898117464569
```

AdaBoostClassifier

In [254]:
```
?AdaBoostStumpClassifier
```

search: **AdaBoostStumpClassifier**

Out[254]: `AdaBoostStumpClassifier(; n_iterations::Int=0)`

Adaboosted decision tree stumps. See DecisionTree.jl's documentation (https://github.com/bensadeghi/DecisionTree.jl)

Hyperparameters:

- `n_iterations` : number of iterations of AdaBoost
- `rng` : the random number generator to use. Can be an `Int` , which will be used to seed and create a new random number generator.

Implements `fit!` , `predict` , `predict_proba` , `get_classes`

Parameter Tuning

In [255]:
```
using ScikitLearn.GridSearch: GridSearchCV

gridsearch = GridSearchCV(AdaBoostStumpClassifier(), Dict(:n_iterations => 1:1:100))
fit!(gridsearch, X_train, y_train)
println("Best parameters: $(gridsearch.best_params_)")
```

```
Best parameters: Dict{Symbol,Any}(:n_iterations=>65)
```

Creating object of Random Forest

In [256]:
```
abc = AdaBoostStumpClassifier(n_iterations = 89);
fit!(abc, X_train, y_train);
```

In [257]:
```
y_pred_abc = predict(abc, X_test);
println("A portion of prediction:")
y_pred_abc[1:10]
```

```
A portion of prediction:
```

Out[257]:
```
10-element Array{Int64,1}:
 0
 0
 0
 0
 0
 0
 1
 0
 1
 1
```

In [258]:
```
cm  = confusion_matrix(y_test, y_pred_abc)
```

```
2×2 Array{Int64,2}:
 97   1
  6  67
```

Out[258]:
```
Classes:  [0, 1]
Matrix:
Accuracy: 0.9590643274853801
Kappa:    0.915603186913911
```

In [259]:
```
?DecisionTreeClassifier
```

search: **DecisionTreeClassifier**

Out[259]:
```
DecisionTreeClassifier(; pruning_purity_threshold=0.0,
                       max_depth::Int=-1,
                       min_samples_leaf::Int=1,
                       min_samples_split::Int=2,
                       min_purity_increase::Float=0.0,
                       n_subfeatures::Int=0,
                       rng=Random.GLOBAL_RNG)
```

Decision tree classifier. See [DecisionTree.jl's documentation (https://github.com/bensadeghi/DecisionTree.jl)](https://github.com/bensadeghi/DecisionTree.jl)

Hyperparameters:

- `pruning_purity_threshold` : (post-pruning) merge leaves having `>=thresh` combined purity (default: no pruning)
- `max_depth` : maximum depth of the decision tree (default: no maximum)
- `min_samples_leaf` : the minimum number of samples each leaf needs to have (default: 1)
- `min_samples_split` : the minimum number of samples in needed for a split (default: 2)
- `min_purity_increase` : minimum purity needed for a split (default: 0.0)
- `n_subfeatures` : number of features to select at random (default: keep all)
- `rng` : the random number generator to use. Can be an `Int` , which will be used to seed and create a new random number generator.

Implements `fit!` , `predict` , `predict_proba` , `get_classes`

In [260]:
```
dtc = DecisionTreeClassifier()
fit!(dtc, X_train, y_train);
y_pred_dtc = predict(dtc, X_test);
println("A portion of prediction:")
y_pred_dtc[1:10]
```

```
A portion of prediction:
```

Out[260]:
```
10-element Array{Int64,1}:
 0
 0
 0
 0
 0
 0
 1
 0
 0
 1
```

In [261]:
```
cm  = confusion_matrix(y_test, y_pred_dtc)
```

```
2×2 Array{Int64,2}:
 96   2
 10  63
```

Out[261]:
```
Classes:  [0, 1]
Matrix:
Accuracy: 0.9298245614035088
Kappa:    0.8545506095832152
```

XGBoost

In [262]:
```
#import Pkg;
#Pkg.add("XGBoost")
```

In [263]:
```
using XGBoost
```

In [264]:
```
?xgboost
```

search: **xgboost XGBoost**

Out[264]: No documentation found.

`XGBoost.xgboost` is a `Function` .

```
# 1 method for generic function "xgboost":
[1] xgboost(data, nrounds::Integer; label, param, watchlist, metrics, obj, feval, group, kwargs...) in XGBoo
st at C:\Users\idpau\.julia\packages\XGBoost\LXjD0\src\xgboost_lib.jl:145
```

In [265]:
```
num_round = 5
xgbc = xgboost(X_train, num_round, label = y_train, eta = 1, max_depth = 2)
```
```
[1]	train-rmse:0.180252
[2]	train-rmse:0.147482
[3]	train-rmse:0.129229
[4]	train-rmse:0.123099
[5]	train-rmse:0.116709
```
Out[265]:
```
Booster(Ptr{Nothing} @0x00000000307d3490)
```
In [266]:
```
y_pred_xgbc = XGBoost.predict(xgbc, X_test)
print("test-error=", sum((y_pred_xgbc .> 0.5) .!= y_test) / float(size(y_pred_xgbc)[1]), "\n")
```
```
test-error=0.10526315789473684
```
In [267]:
```
y_pred_xgbc
```
Out[267]:
```
171-element Array{Float32,1}:
  0.0123640895
  0.0123640895
  0.0123640895
 -0.35979003
  0.0123640895
  0.4310953
  0.9815798
  0.0123640895
  0.0123640895
  0.9815798
  0.0123640895
  0.26188624
  1.028301
  ⋮
  0.4447895
  1.028301
  0.89318776
  0.0123640895
  0.0123640895
  0.0123640895
  0.0123640895
  0.0123640895
  0.9815798
  0.67093486
  0.9815798
  0.9815798
```
In [268]:
```
cm  = confusion_matrix(y_test, y_pred_xgbc[0])
```
```
BoundsError: attempt to access 171-element Array{Float32,1} at index [0]

Stacktrace:
 [1] getindex(::Array{Float32,1}, ::Int64) at .\array.jl:729
 [2] top-level scope at In[268]:1
```

Performance and Results

Performance evaluation criteria

Defining functions for calculating different performance metrices

In [269]:
```
using PyCall
math = pyimport("math")

function accuracy_(tn,fp,fn,tp)
    return ((tp+tn)/(tp+fp+fn+tn))
end

#True Positive Rate or Recall
function sensitivity(tp,fn)
    return (tp / (tp + fn))
end

tprate = sensitivity
recall = sensitivity

#True Negative Rate
function specificity(tn,fp)
    return (tn / (fp + tn))
end

function mcc(tn,fp,fn,tp)
    return (tp*tn-fp*fn)/math.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
end

function auc_score(tn,fp,fn,tp)
    return (sensitivity(tp,fn) + specificity(tn,fp)) / 2
end

function gmean_score(tn,fp,fn,tp)
    return math.sqrt(sensitivity(tp,fn) * specificity(tn,fp))
end

function precision(tp, fp)
    return (tp / (tp + fp))
end

function f1score(tp, fp, fn)
    return (2*tp/(2*tp + fp + fn))
end

function fprate(tn,fp)
    return (fp / (fp + tn))
end
```

Out[269]:
```
fprate (generic function with 1 method)
```

Definfing a function for producing all scores at once.

In [270]:
```
function getAllScore(tp,tn,fp,fn)
    acc = accuracy_(tn,fp,fn,tp)
    sn = sensitivity(tp,fn)
    sp = specificity(tn,fp)
    auc = auc_score(tn,fp,fn,tp)
    gmean = gmean_score(tn,fp,fn,tp)
    preci = precision(tp, fp)
    f1 = f1score(tp,fp,fn)
    fpr = fprate(tn,fp)
    return acc, sn, sp, auc, gmean, preci, f1, fpr
end
```

Out[270]:
```
getAllScore (generic function with 1 method)
```

Result of RandomForest

Loading MLBase to compute roc performance parameter

In [271]:
```
using MLBase
r_rfc = roc(y_test, y_pred_rfc)
```

Out[271]:
```
ROCNums{Int64}
  p = 73
  n = 98
  tp = 66
  tn = 96
  fp = 2
  fn = 7
```

Calling the getAllScore function

In [272]:
```
allScoresTuples = getAllScore(r_rfc.tp, r_rfc.tn, r_rfc.fp, r_rfc.fn)
allScoresArray = [i for i in allScoresTuples]
allScores = transpose(allScoresArray)
```

Out[272]:
```
1×8 LinearAlgebra.Transpose{Float64,Array{Float64,1}}:
 0.947368  0.90411  0.979592  0.941851  …  0.970588  0.93617  0.0204082
```

In [273]:
```
?CSV.write
```

Out[273]:
```
CSV.write(file, table; kwargs...) => file
table |> CSV.write(file; kwargs...) => file
```

Write a [Tables.jl interface input (https://github.com/JuliaData/Tables.jl)](https://github.com/JuliaData/Tables.jl) to a csv file, given as an `IO` argument or `String` /FilePaths.jl type representing the file name to write to.

Supported keyword arguments include:

- `delim::Union{Char, String}=','` : a character or string to print out as the file's delimiter
- `quotechar::Char='"'` : ascii character to use for quoting text fields that may contain delimiters or newlines
- `openquotechar::Char` : instead of `quotechar`, use `openquotechar` and `closequotechar` to support different starting and ending quote characters
- `escapechar::Char='"'` : ascii character used to escape quote characters in a text field
- `missingstring::String=""` : string to print for `missing` values
- `dateformat=Dates.default_format(T)` : the date format string to use for printing out `Date` & `DateTime` columns
- `append=false` : whether to append writing to an existing file/IO, if `true`, it will not write column names by default
- `writeheader=!append` : whether to write an initial row of delimited column names, not written by default if appending
- `header` : pass a list of column names (Symbols or Strings) to use instead of the column names of the input table
- `newline='\n'` : character or string to use to separate rows (lines in the csv file)
- `quotestrings=false` : whether to force all strings to be quoted or not
- `decimal='.'` : character to use as the decimal point when writing floating point numbers

In [274]:
```
allScoresArray
```

Out[274]:
```
8-element Array{Float64,1}:
 0.9473684210526315
 0.9041095890410958
 0.9795918367346939
 0.9418507128878948
 0.941094242325505
 0.9705882352941176
 0.9361702127659575
 0.02040816326530612
```

Storing the all scores into a dataframe

In [275]:
```
columnNames = ["accuracy", "sensitivity", "specificity", "auc", "gmean", "precision", "f1_score", "fpr"]
resultDF = DataFrame(allScores, Symbol.(columnNames), )
```

Out[275]: 1 rows × 8 columns

	accuracy	sensitivity	specificity	auc	gmean	precision	f1_score	fpr
	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64
1	0.947368	0.90411	0.979592	0.941851	0.941094	0.970588	0.93617	0.0204082

Result of AdaBoost

In [276]:
```
using MLBase
r_abc = roc(y_test, y_pred_abc)
```

Out[276]:
```
ROCNums{Int64}
  p = 73
  n = 98
  tp = 67
  tn = 97
  fp = 1
  fn = 6
```

Calling the getAllScore function

In [277]:
```
allScoresTuples = getAllScore(r_abc.tp, r_abc.tn, r_abc.fp, r_abc.fn)
allScoresArray = [i for i in allScoresTuples]
allScores = transpose(allScoresArray)
```

Out[277]:
```
1×8 LinearAlgebra.Transpose{Float64,Array{Float64,1}}:
 0.959064  0.917808  0.989796  0.953802  …  0.985294  0.950355  0.0102041
```

In [278]:
```
allScoresArray
```

Out[278]:
```
8-element Array{Float64,1}:
 0.9590643274853801
 0.9178082191780822
 0.9897959183673469
 0.9538020687727146
 0.9531226726851424
 0.9852941176470589
 0.950354609929078
 0.01020408163265306
```

Storing the all scores into a dataframe

In [279]:
```
columnNames = ["accuracy", "sensitivity", "specificity", "auc", "gmean", "precision", "f1_score", "fpr"]
resultDF = DataFrame(allScores, Symbol.(columnNames), )
```

Out[279]: 1 rows × 8 columns

	accuracy	sensitivity	specificity	auc	gmean	precision	f1_score	fpr
	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64
1	0.959064	0.917808	0.989796	0.953802	0.953123	0.985294	0.950355	0.0102041

Drawing a ROC curve with auc score

In [280]:
```
fprArr = [0, fprate(r_rfc.tn,r_rfc.fp), 1]
tprArr = [0, tprate(r_rfc.tp,r_rfc.fn), 1]
auc_ = round(auc_score(r_rfc.tn,r_rfc.fp,r_rfc.fn,r_rfc.tp), digits=3)
plt.plot(fprArr,tprArr,label= string( "RandomForest", " ROC (auc =", auc_, ")" ) )

fprArr = [0, fprate(r_abc.tn,r_abc.fp), 1]
tprArr = [0, tprate(r_abc.tp,r_abc.fn), 1]
auc_ = round(auc_score(r_abc.tn,r_abc.fp,r_abc.fn,r_abc.tp), digits=3)
plt.plot(fprArr,tprArr,label= string( "AdaBoost", " ROC (auc =", auc_, ")" ) )

xlabel("1 - Specificity or (False Positive Rate)")
ylabel("Sensitivity(True Positive Rate)")
plt.plot([0, 1], [0, 1],"r--")
plt.xlim([0.0, 1.05])
plt.ylim([0.0, 1.05])
PyPlot.title("Receiver Operating Characteristic")
grid("on")
legend()
plt.savefig("roc_curve.png",bbox_inches="tight", format="png", dpi=1200)
plt.show()   # Display
```