UNIVERSITY COLLEGE DUBLIN

School of Mathematics and Statistics

DATA and COMPUTATION SCIENCE

Module Name : Mathematica For Research
Subject : Applied & Computational Maths
Module Code : ACM40730
Trimester : Autumn
Module Coordinator : Dr Niels Warburton
Student Name : Utpal Mishra
Student Number : 20 207 425

FINAL PROJECT

Project Objective

The general objective of the project is to design a system and automated the process of breast cancer detection using the techniques of machine learning and provide some assistance in diagnosis and treatment to the radiologists and physicians. In order to achieve this, the following are some specific objectives :

1. To analyse previous related works on breast cancer detection and classification so as to select relevant methods and techniques.
2. To opt for befitting methods for segmentation, data preprocessing, feature extraction and classification.
3. To deliver a prototype for the proposed approach.
4. To assess the performance of the project in terms of f-score, precision, recall, error and classification accuracy.
5. To deploy a time - efficient and reliable prototype that could assist pathologists.

Abstract

Among all the form of cancer in women, **breast cancer contributes to over 20 %** and has resulted with more in millions new cases each year (**over 5 lakh recorded in 2012**). With lack in diagnosis and treatment in India, **a women is detected with breast cancer in every 4 minutes** along-with the **highest death cases** world wide (2012) i.e. **in every 8 minutes**. Automating the process will be really beneficial for the people suffering from cancer, being can be cost - effective with elevated accuracy. In this project, KNN and SVM are found to be the most useful algorithms to work with then the data undertaken.

Introduction

Breast cancer is the **fifth most common cause of death** and the **second most common type of cancer** after lung cancer. The National Breast Cancer Foundation has valued around **2 lakh fresh cases** of breast cancer and annually, witnessing **40, 000 deaths** in women while in men, these numbers are 1700 and 450, correspondingly.

Cancer comes under a class of diseases which are characterised by **abnormal growth of cells** which hinder surrounding healthy cell in the body. Breast cancer starts to penetrates in the breast cells of both men and women and spread to different body parts. It results in **lump formation** near the cells due accumulation of tissues thus, early detection is crucial. Recent statistics show that breast cancer is a serious disease with a high incidence rate and one of the leading causes of the early death of women. Being a paramount stage, statistics show a **five - year survival rate of 96 % in early diagnosis** of breast cancer as being one of the curable form of cancer. At the early stage, detection can be performed using the following two strategies:

A. **Early Diagnosis**: Using effective treatments to reduce the risk of cancer by escalating the identification proportion at the early stage.

B. **Screening**: Rectifying the presence of cancerous cells before the symptoms can showcase using various tools such as:
i. Breast Self - Exam (BSE)
ii. Clinical Breast Exam (CBE)
iii. Mammography

*This research project aims to probe into the possibility of detecting and classifying breast cancer from the **Breast Cancer Wisconsin (Diagnostic) Data**.

Algorithms

Following are the **machine learning classification algorithms** used for detecting breast cancer:

1. Naive Bayes (NB)

Naive Bayes is a probabilistic classifier which applies Bayes' theorem with strong independent assumptions. In this model, all properties are separately taken into consideration to find any relationship between them. It assumes that predictive attributes are conditionally independent given the class. Additionally, the values of the numeric attributes are distributed within each class. Naive Bayes is fast and performs well even with a small dataset. However, it is hard to find independent properties in real life. Researchers have deployed NB classifiers for breast cancer detection and achieved the maximum

accuracy with only five dominant.

2. Decision Tree (DT)

Decision Tree is a data mining technique used for early detection of breast cancer. It is a model that presents classification or regression as a tree. In this model, the dataset is split into small sub - data, then into smaller ones. As a result, the tree is developed and at the last level reveals the result. In a tree structure, the leaves characterise the class labels while the branches characterise conjunctions of features which lead to the class labels. Hence, DT is not sensitive to noise. Now, if we are dealing with a larger dataset, greater are the splits and such huge trees result in elevating the complexity and leads to overfitting. A simple approach to limit the splitting is by fixing the minimum training dataset at the input and the maximum depth of the decision making tree. Another approach is by using the method of Pruning.

An effective technique to optimize the performance of Decision Tree is by simply getting rid of the attributes with lesser information or higher entropy. There are two approaches to execute this technique to either start from root or leaves. The bottom - up method in which child attributes having lesser significance are removed without altering the accuracy is known as Reduce Error Pruning.

3. Random Forest (RF)

This algorithm is an extension of Decision Tree, as being the building block.The name "Random Forest", is self - explanatory where "Forest" signifies averaging the prediction of the tree and "Random" explains two concepts : i.For tree building : random sampling of training data points for building the tree.ii.For splitting nodes : after sample data is fetched, a random subset of features is chosen for analysis to split the nodes, as the tree learns from the random sample and their features.For building a tree, multiple samples are used multiple times which is known as Bootstrapping.

It can also be defined as a random sampling of observational data with replacement.While training each tree, different samples used for building trees might have a higher variance but the cumulative variance of the forest must be low and not at the cost of an increase in bias.The prediction is made by averaging the prediction of each Decision Tree and is termed as Bootstrap Aggregating or Bagging.It can also be said as a different randomized (bootstrapped) subset of data for making up the individual trees that are trained and then averaged for prediction.The contrast between Decision Tree and Random Forest is that, instead of shortlisting the result from a single tree, multiple trees are taken into consideration and evaluated for prediction.As in epitome, predicting a product in the market by the cumulative reviews for E - commerce websites, Google and Youtube.

4. K - Nearest Neighbors (KNN)

KNN is a supervised learning method. It can be used for diagnosing and classifying cancer. In this method, the model is trained in a specific field and new data is given to it. Furthermore, similar data is

used by the machine for detecting "K". Therefore, the machine finds KNN for the unknown data. It is recommended to choose a large dataset for training also K value must be an odd number determining closeness.

For calculating the closeness, the distance between data observations is estimated and is known as Minkowski Distance. The distance calculated in a normed vector space i.e. space where the distance is observed in vectors is called a Minkowski Distance. "Normed '' signifies that vectors have length and no vector can have negative length.

5. Support Vector Machines (SVM)

SVM is a supervised machine learning algorithm that is a pattern classification model. It is used as a training algorithm for learning classification and regression rules from collected data. The motive of this method is to segregate data until a hyperplane with high minimum distance is found. Using SVM, we can classify two or more data types. SVM is an efficient method for diagnosing breast cancer. Its accuracy can increase when combined with feature selection. This accuracy can be obtained on the WBCD dataset, considering five features.SVM is the most efficient way of statistical learning. It can easily identify the decision boundaries between different classes of breast cancer. The input features are selected after calculating the F - score of each input feature. The significance of each feature is evaluated by it' s F - score. The SVM parameters are optimised by grid search.

It is called a Discriminative Classifier and learns about classification from given statistics depending on the observed data.It can also be called as a Discriminative Model or a Conditional Model for statistical classification in (Supervised) Machine Learning defined by separating hyperplanes.In a multi - dimensional space for separating classes, labeled training data are used to form optimal hyperplanes (2 D) while 3 D hyperplanes are used for kernel transformations.

6. Neural Networks (NN)

ANN is a machine learning model similar to the human brain' s nerve system having a large number of interconnected nodes. Each node has two states : 0 represents the inactive state and 1 represents the active state. Furthermore, each node has either a positive or a negative weight that tunes the strength of that node and can activate or deactivate it. The machine is trained on samples of data using ANN. The trained machine is then used to detect the pattern of hidden data. It can search for patterns between patients' healthcare and personal records to identify high - risk lesions.

The advantages of using ANN are - They can learn and model non - linear and complex relationships. They can generalise. No restrictions are imposed on the input variables (like how they should be distributed).Its region growing - based segmentation method is improved by using the extracted intensity features from ROIs and applying the ANN to generate an adaptive threshold.

Loading the Datasets

About data

1. The Breast Cancer Classification data has been extracted from **Wisconsin Dataset.**
2. The data contain “diagnosis” as the dependent variable and rest 31 numerical features as independent ones.
3. Within the attribute “diagnosis”, category **1 represents “Malignant” (Abnormal)** while **0 reflects “Benign” (Normal).**
4. The **dimensions of the data are 567x32** which has been divided into **training (0.80)** and **testing data (0.20)** with shapes {455, 32} and {112, 32}, respectively.

Loading the training data

In[ ]:=
```
Clear[train, test]
```

In[253]:=
```
train = Import["E:\\UCD\\Lectures\\Semester 1\\Mathematica\\PROJECT\\train.csv"];
header = train[[1]];
traindata = train[[2 ;;]];
train = Thread[header → #] & /@ traindata // Map[Association] // Dataset;
Dimensions[train]
train[1 ;; 5]
```

Out[257]= {455, 32}

Out[258]=

id	radius_mean	texture_mean	perimeter_mea	area_mean	smoothness_m	compactness_r	co
842 302	17.99	10.38	122.8	1001	0.1184	0.2776	0.
842 517	20.57	17.77	132.9	1326	0.08474	0.07864	0.
84 300 903	19.69	21.25	130	1203	0.1096	0.1599	0.
84 348 301	11.42	20.38	77.58	386.1	0.1425	0.2839	0.
84 358 402	20.29	14.34	135.1	1297	0.1003	0.1328	0.

columns 1–10 of 32

Loading the testing data

In[259]:=
```
test = Import["E:\\UCD\\Lectures\\Semester 1\\Mathematica\\PROJECT\\test.csv"];
header = test[[1]];
testdata = test[[2 ;;]];
test = Thread[header → #] & /@ testdata // Map[Association] // Dataset;
Dimensions[test]
test[1 ;; 5]
```

Out[263]= {114, 32}

Out[264]=

id	radius_mean	texture_mean	perimeter_mea	area_mean	smoothness_m	compactness_r	cor
9 112 085	13.38	30.72	86.34	557.2	0.09245	0.07426	0.0
9 112 366	11.63	29.29	74.87	415.1	0.09357	0.08574	0.0
9 112 367	13.21	25.25	84.1	537.9	0.08791	0.05205	0.0
9 112 594	13	25.13	82.61	520.2	0.08369	0.05073	0.0
9 112 712	9.755	28.2	61.68	290.9	0.07984	0.04626	0.0

columns 1–10 of 32

Preparing the Model

Anomalies Detection

Anomalies are also referred to as outliers, noise, deviations and exceptions.
Anomaly Detection is a data mining technique to remove the data points that does not fit the model well.
In data analysis, rectification of an abnormal behaving data point which is significantly different from its relative values is also termed as Anomaly Detection.
One of the prominent density based technique used is K-Nearest Nearest, and the same is reflected through the obtained fitted model evaluations compared to other models.

In[13]:=
```
adtrain = AnomalyDetection[train]
trainoutliers = FindAnomalies[adtrain, train]
```

Out[13]=
```
AnomalyDetectorFunction[ 
    Input type: Mixed (number: 32)
    False positive rate: 0.001 ]
```

Out[14]=

id	radius_mean	texture_mean	perimeter_mea	area_mean	smoothness_m	compactness_
842302	17.99	10.38	122.8	1001	0.1184	0.2776
84348301	11.42	20.38	77.58	386.1	0.1425	0.2839
84501001	12.46	24.04	83.97	475.9	0.1186	0.2396
846226	19.17	24.8	132.4	1123	0.0974	0.2458
852631	17.14	16.4	116	912.7	0.1186	0.2276
855133	14.99	25.2	95.54	698.8	0.09387	0.05131
855625	19.07	24.81	128.3	1104	0.09081	0.219
859471	9.029	17.33	58.79	250.5	0.1066	0.1413
859711	8.888	14.64	58.79	244	0.09783	0.1531
8610629	13.53	10.94	87.91	559.2	0.1291	0.1047
8610862	20.18	23.97	143.7	1245	0.1286	0.3454
8611555	25.22	24.91	171.5	1878	0.1063	0.2665
8611792	19.1	26.29	129.1	1132	0.1215	0.1791
86355	22.27	19.67	152.8	1509	0.1326	0.2768
86409	14.26	19.65	97.83	629.9	0.07837	0.2233
864726	8.95	15.76	58.74	245.2	0.09462	0.1243
865128	17.95	20.01	114.2	982	0.08402	0.06722
865423	24.25	20.2	166.2	1761	0.1447	0.2867
868826	14.95	17.57	96.85	678.1	0.1167	0.1305
869691	11.8	16.58	78.99	432	0.1091	0.17

rows 1–20 of 53 columns 1–10 of 32

In[15]:=
```
adtest = AnomalyDetection[test]
testoutliers = FindAnomalies[adtest, test]
```

Out[15]= AnomalyDetectorFunction[Input type: Mixed (number: 32) False positive rate: 0.001]

Out[16]=

id	radius_mean	texture_mean	perimeter_mea	area_mean	smoothness_m	compactness_r	co
911296 20	27.42	26.27	186.9	2501	0.1084	0.1988	0.
915186	9.268	12.87	61.49	248.7	0.1634	0.2239	0.
915276	9.676	13.14	64.12	272.5	0.1255	0.2204	0.
925622	15.22	30.62	103.4	716.9	0.1048	0.2087	0.

columns 1–10 of 32

Preparing Y_train and Y_test columns for the model

In[17]:=
```
trainPrep = train[All, "diagnosis"];
testPrep = test[All, "diagnosis"];
```

Extracting the features for training and testing data

In[19]:=
```
trainFeatures =
  train[1 ;; Length[train], {"radius_mean", "texture_mean", "perimeter_mean",
    "area_mean", "smoothness_mean", "compactness_mean", "concavity_mean",
    "concave points_mean", "symmetry_mean", "fractal_dimension_mean",
    "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_se",
    "compactness_se", "concavity_se", "concave points_se", "symmetry_se",
    "fractal_dimension_se", "radius_worst", "texture_worst", "perimeter_worst",
    "area_worst", "smoothness_worst", "compactness_worst", "concavity_worst",
    "concave points_worst", "symmetry_worst", "fractal_dimension_worst", "diagnosis"}];

testFeatures =
  test[1 ;; Length[test], {"radius_mean", "texture_mean", "perimeter_mean", "area_mean",
    "smoothness_mean", "compactness_mean", "concavity_mean", "concave points_mean",
    "symmetry_mean", "fractal_dimension_mean", "radius_se", "texture_se",
    "perimeter_se", "area_se", "smoothness_se", "compactness_se", "concavity_se",
    "concave points_se", "symmetry_se", "fractal_dimension_se", "radius_worst",
    "texture_worst", "perimeter_worst", "area_worst", "smoothness_worst",
    "compactness_worst", "concavity_worst", "concave points_worst",
    "symmetry_worst", "fractal_dimension_worst", "diagnosis"}];
```

Flattening the training and testing data

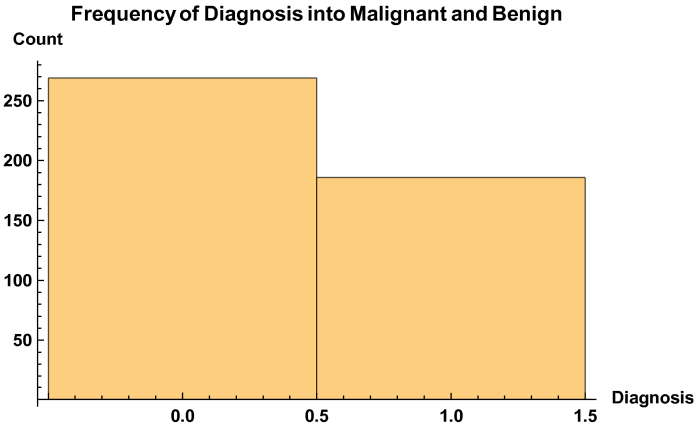
In[21]:=
```
trainFinal =
  Flatten[Normal@trainFeatures[1 ;; Length[trainPrep], {Most@# → Last@#} &], 1];
testFinal = Flatten[Normal@testFeatures[1 ;; Length[testPrep], {Most@# → Last@#} &], 1];
```

Data Visualization

Frequency Plot of Diagnosis into Malignant and Benign

In[25]:=
```
Histogram[train[All, 32], AxesLabel → {HoldForm[Diagnosis], HoldForm[Count]},
 PlotLabel → HoldForm[Frequency of Diagnosis into Malignant and Benign],
 LabelStyle → {GrayLevel[0], Bold}]
```

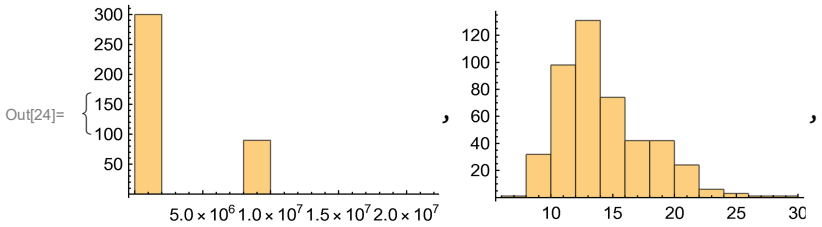
Out[25]=

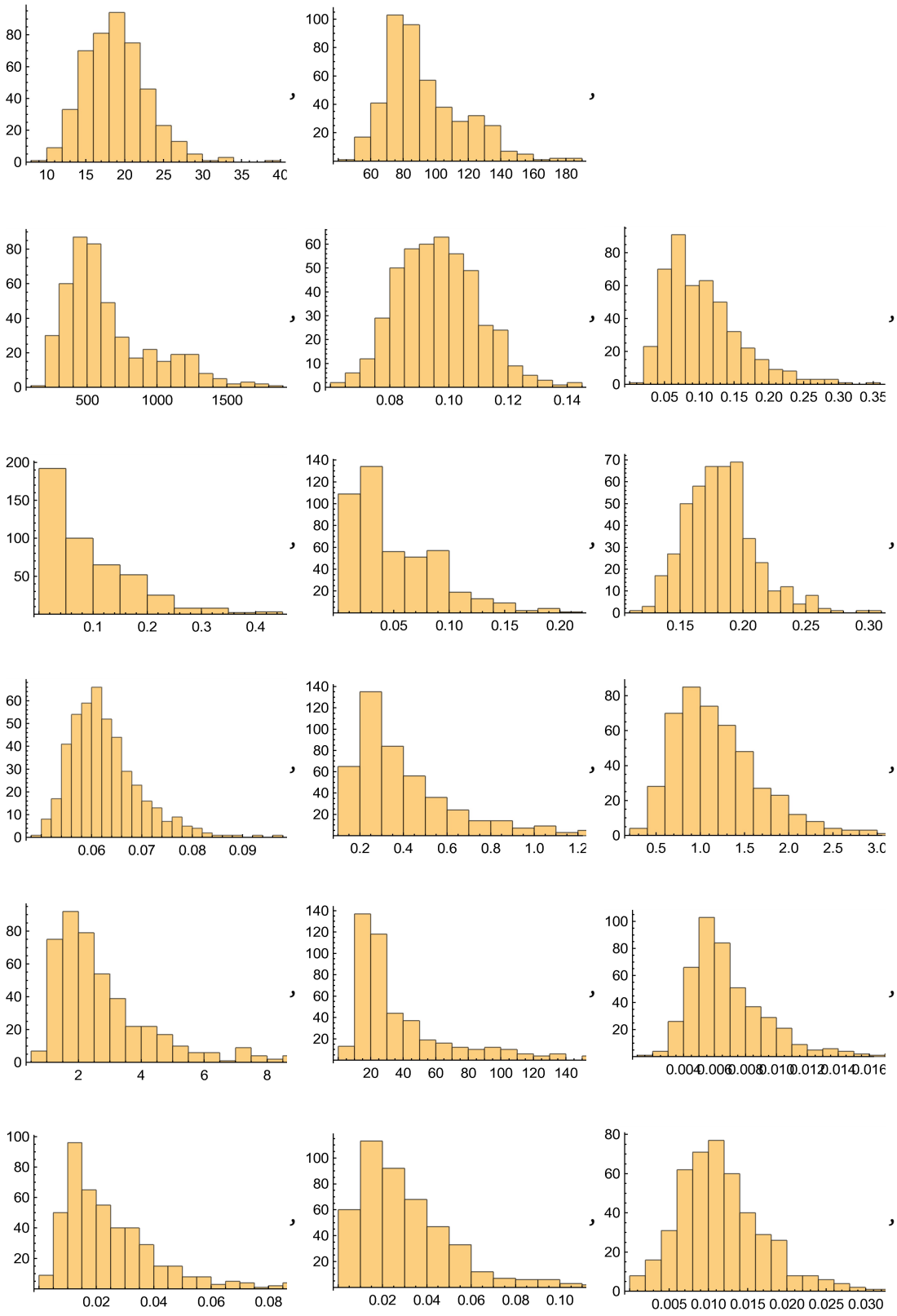


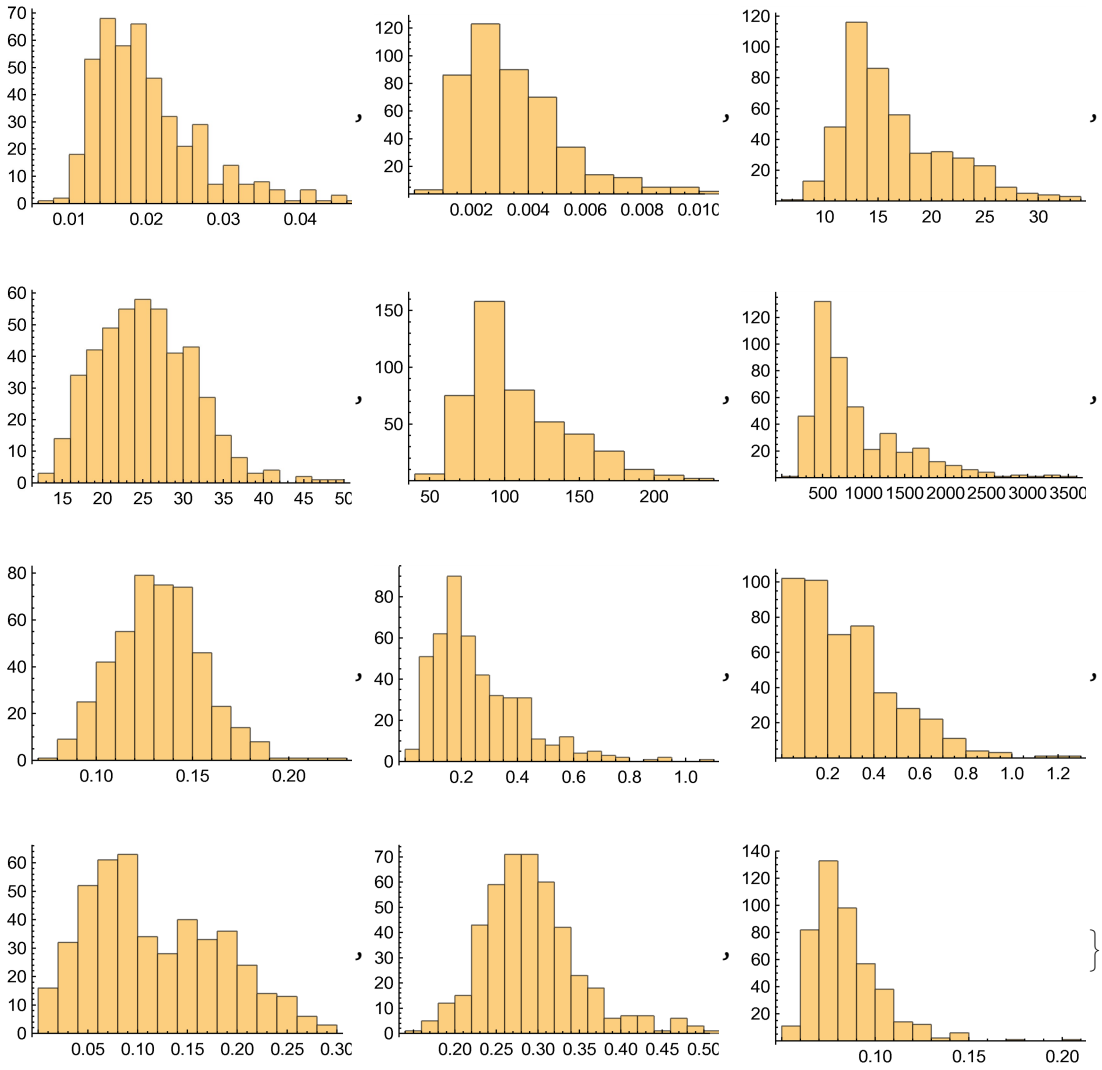
Features Histogram

In[24]:=
```
Table[Histogram[train[All, i]], {i, 31}]
```

Out[24]=







Building Classification Model

Explicit Model Functions

Classification Function

In[26]:=
```
Classification[data_, method_] := Classify[data, Method → method]
```

Algorithm Information

In[27]:=
```
AlgorithmInformation[model_, about_] := ClassifierInformation[model, about]
```

Model Information

In[28]:=
```
ModelInformation[model_] := Information[model]
Info[mod_] := Module[{ModelInformation}, ModelInformation[mod]; mod]
(* Info[mod_]:=Block[{ModelInformation}, ModelInformation[mod]; mod] *)
```

Naive Bayes Classification Algorithm

Fitting the model

In[30]:=
```
Set[algoNB,
 Classify[trainFinal, Method → "NaiveBayes", TrainingProgressReporting → "SimplePanel"]]
```

Out[30]= ClassifierFunction[ Input type: Mixed (number: 30)
Classes: 0, 1]

About the algorithm

In[32]:=
```
ClassifierInformation[algoNB, "MethodDescription"]
```

Out[32]= The naive Bayes classifier assumes that features are generated independently given the class and uses Bayes' theorem to predict the class.

Evaluation Measurements

PROPERTIES: {Accuracy,AccuracyBaseline,AccuracyRejectionPlot,AreaUnderROCCurve,BatchEvaluationTime,BestClassifiedExamples,ClassifierFunction,ClassMeanCrossEntropy,ClassRejectionRate,CohenKappa,ConfusionDistribution,ConfusionFunction,ConfusionMatrix,ConfusionMatrixPlot,CorrectlyClassifiedExamples,DecisionUtilities,Error,EvaluationTime,Examples,F1Score,FalseDiscoveryRate,FalseNegativeExamples,FalseNegativeNumber,FalseNegativeRate,FalsePositiveExamples,FalsePositiveNumber,FalsePositiveRate,GeometricMeanProbability,IndeterminateExamples,LeastCertainExamples,Likelihood,LogLikelihood,MatthewsCorrelationCoefficient,MeanCrossEntropy,MeanDecisionUtility,MisclassifiedExamples,MostCertainExamples,NegativePredictiveValue,Perplexity,Precision,Probabilities,ProbabilityHistogram,Properties,Recall,RejectionRate,Report,ROCCurve,ScottPi,Specificity,TopConfusions,TrueNegativeExamples,TrueNegativeNumber,TruePositiveExamples,TruePositiveNumber,WorstClassifiedExamples}

In[33]:=
```
cmNB = ClassifierMeasurements[algoNB, testFinal]
```

Out[33]= ClassifierMeasurementsObject[ Classifier: NaiveBayes
Number of test examples: 114]

In[ ]:=
```
(* cmNB/@{"Accuracy","Error", "FScore",  "Precision", "Recall", "AreaUnderROCCurve"}//
 TableForm *)
```

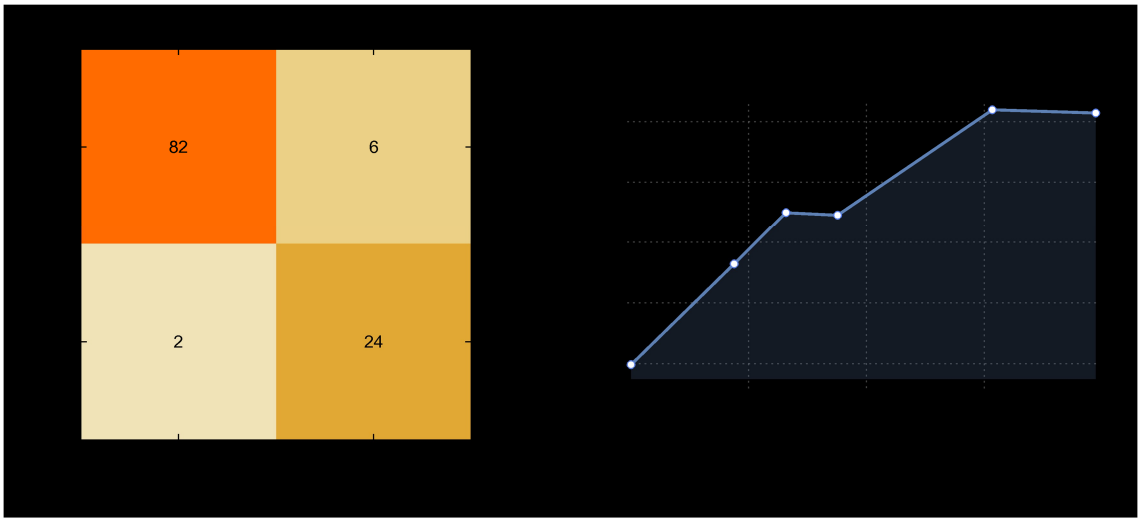
In[34]:=
```
PROPERTIES = {"Accuracy", "EvaluationTime", "Error",
   "FScore", "Precision", "Recall", "Specificity", "AreaUnderROCCurve"};
For[i = 1, i ≤ Length[PROPERTIES], i++,
 Print[PROPERTIES[[i]], ": ", cmNB[PROPERTIES[[i]]]]]
```

```
Accuracy: 0.929825

EvaluationTime: 0.0070

Error: 0.0701754

FScore: ⟨|0 → 0.953488, 1 → 0.857143|⟩

Precision: ⟨|0 → 0.97619, 1 → 0.8|⟩

Recall: ⟨|0 → 0.931818, 1 → 0.923077|⟩

Specificity: ⟨|0 → 0.923077, 1 → 0.931818|⟩

AreaUnderROCCurve: ⟨|0 → 0.994318, 1 → 0.994318|⟩
```

Confusion Matrix Plot and Accuracy vs Rejection Rate Plot

In[36]:=
```
GraphicsRow[
 {cmNB["ConfusionMatrixPlot"], cmNB["AccuracyRejectionPlot"]}, Background → Black]
```

Out[36]=



Model Evaluations

In[37]:=
```
Map[Info, algoNB]
```

Out[37]=
```
ClassifierFunction[
```

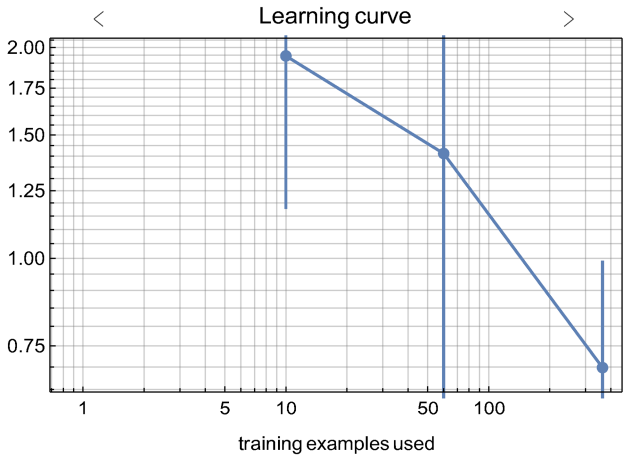
Input type: Mixed (number: 30)
Classes: 0, 1
]

In[38]:=
```
ModelInformation /@ {algoNB}
```

Out[38]=

Classifier information

Data type	Mixed (number: 30)
Classes	0, 1
Accuracy	(94.0 ± 1.6)%
Method	NaiveBayes
Single evaluation time	5.09 ms/example
Batch evaluation speed	7.69 examples/ms
Loss	0.699 ± 0.29
Model memory	361. kB
Training examples used	455 examples
Training time	488. ms



Decision Tree Classification Algorithm

Fitting the model

In[39]:=
```
Set[algoDT, Classify[trainFinal, Method → "DecisionTree"]]
```

Out[39]=
```
ClassifierFunction[
```

Input type: Mixed (number: 30)
Classes: 0, 1
```
]
```

About the algorithm

In[40]:=
```
ClassifierInformation[algoDT, "MethodDescription"]
```

Out[40]=
The decision tree classifier models class probabilities with a decision tree constructed using the CART algorithm.

Evaluation Measurements

PROPERTIES :
{Accuracy, AccuracyBaseline, AccuracyRejectionPlot, AreaUnderROCCurve, BatchEvaluationTime, BestClassifiedExamples, ClassifierFunction, ClassMeanCrossEntropy, ClassRejectionRate, CohenKappa, ConfusionDistribution, ConfusionFunction, ConfusionMatrix, ConfusionMatrixPlot, CorrectlyClassifiedExamples, DecisionUtilities, Error, EvaluationTime, Examples, F1Score, FalseDiscoveryRate, FalseNegativeExamples, FalseNegativeNumber, FalseNegativeRate, FalsePositiveExamples, FalsePositiveNumber, FalsePositiveRate, GeometricMeanProbability, IndeterminateExamples, LeastCertainExamples, Likelihood, LogLikelihood, MatthewsCorrelationCoefficient, MeanCrossEntropy, MeanDecisionUtility, MisclassifiedExamples, MostCertainExamples, NegativePredictiveValue, Perplexity, Precision, Probabilities, ProbabilityHistogram, Properties, Recall, RejectionRate, Report, ROCCurve, ScottPi, Specificity, TopConfusions, TrueNegativeExamples, TrueNegativeNumber, TruePositiveExamples, TruePositiveNumber, WorstClassifiedExamples}

In[41]:=
```
cmDT = ClassifierMeasurements[algoDT, testFinal]
```

Out[41]=
ClassifierMeasurementsObject[Classifier: DecisionTree
Number of test examples: 114]

In[ ]:=
```
(* cmDT/@{"Accuracy","FScore","Error",  "Precision", "Recall"}//TableForm *)
```

In[42]:=
```
PROPERTIES = {"Accuracy", "EvaluationTime", "Error",
   "FScore", "Precision", "Recall", "Specificity", "AreaUnderROCCurve"};
i = 1; While[i ≤ Length[PROPERTIES], Print[PROPERTIES[[i]],
  ": ", cmDT[PROPERTIES[[i]]]]; i++]
```

```
Accuracy: 0.868421

EvaluationTime: 0.0032

Error: 0.131579

FScore: ⟨|0 → 0.909091, 1 → 0.761905|⟩

Precision: ⟨|0 → 0.974026, 1 → 0.648649|⟩

Recall: ⟨|0 → 0.852273, 1 → 0.923077|⟩

Specificity: ⟨|0 → 0.923077, 1 → 0.852273|⟩

AreaUnderROCCurve: ⟨|0 → 0.965472, 1 → 0.971154|⟩
```

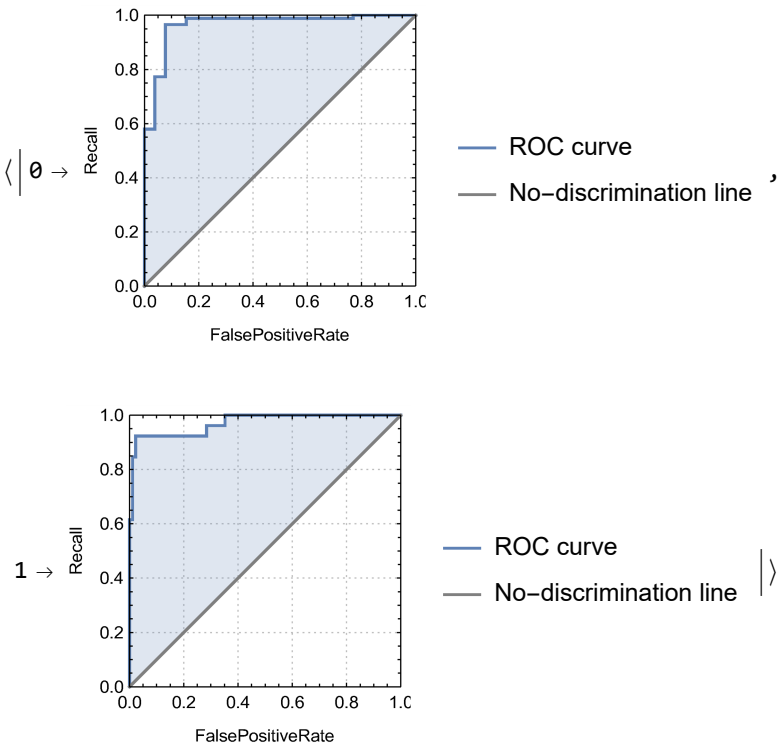
Confusion Matrix Plot and ROC Curve

In[48]:=
```
GraphicsRow[{cmDT["ConfusionMatrixPlot"]}, Background → LightYellow]
```

Out[48]=

In[45]:=
```
cmDT["ROCCurve"]
```

Out[45]=



Model Evaluations

In[46]:=
```
Info[algoDT]
```

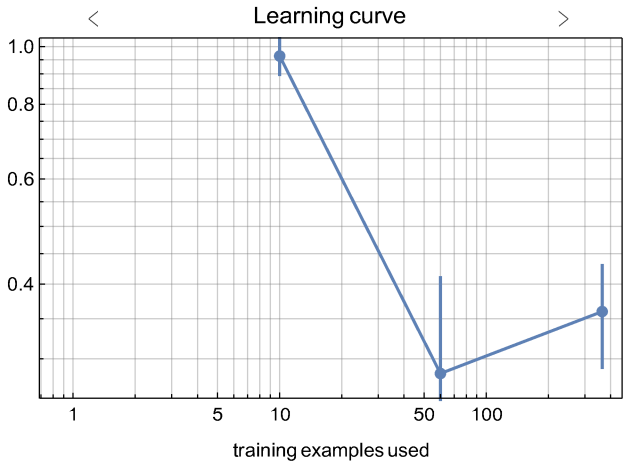
Out[46]=
```
ClassifierFunction[ Input type: Mixed (number: 30)
                    Classes: 0, 1 ]
```

In[47]:=
```
ModelInformation[algoDT]
```

Out[47]=

Classifier information

Data type	Mixed (number: 30)
Classes	0, 1
Accuracy	(91.1 ± 3.2)%
Method	DecisionTree
Single evaluation time	2.54 ms/example
Batch evaluation speed	43.8 examples/ms
Loss	0.284 ± 0.13
Model memory	210. kB
Training examples used	455 examples
Training time	842. ms



Random Forest Classification Algorithm

Fitting the model

In[49]:=
```
Set[algoRF, Classify[trainFinal, Method → "RandomForest"]]
```

Out[49]=
```
ClassifierFunction[  ]
```
Input type: Mixed (number: 30)
Classes: 0, 1

About the algorithm

In[50]:=
```
ClassifierInformation[algoRF, "MethodDescription"]
```

Out[50]=

The random forest classifier uses an ensemble of decision trees to predict the class. Each decision tree has been trained on a random subset of the training set, and only uses a random subset of the features.

Evaluation Measurements

PROPERTIES :
{Accuracy, AccuracyBaseline, AccuracyRejectionPlot, AreaUnderROCCurve, BatchEvaluationTime, BestClassifiedExamples, ClassifierFunction, ClassMeanCrossEntropy, ClassRejectionRate, CohenKappa, ConfusionDistribution, ConfusionFunction, ConfusionMatrix, ConfusionMatrixPlot, CorrectlyClassifiedExamples, DecisionUtilities, Error, EvaluationTime, Examples, F1Score, FalseDiscoveryRate, FalseNegativeExamples, FalseNegativeNumber, FalseNegativeRate, FalsePositiveExamples, FalsePositiveNumber, FalsePositiveRate, GeometricMeanProbability, IndeterminateExamples, LeastCertainExamples, Likelihood, LogLikelihood, MatthewsCorrelationCoefficient, MeanCrossEntropy, MeanDecisionUtility, MisclassifiedExamples, MostCertainExamples, NegativePredictiveValue, Perplexity, Precision, Probabilities, ProbabilityHistogram, Properties, Recall, RejectionRate, Report, ROCCurve, ScottPi, Specificity, TopConfusions, TrueNegativeExamples, TrueNegativeNumber, TruePositiveExamples, TruePositiveNumber, WorstClassifiedExamples}

In[51]:=
```
cmRF = ClassifierMeasurements[algoRF, testFinal]
```

Out[51]=
```
ClassifierMeasurementsObject[  ]
```
Classifier: RandomForest
Number of test examples: 114

In[ ]:=
```
(* cmRF/@{"Accuracy","FScore","Error",  "Precision"}//TableForm *)
```

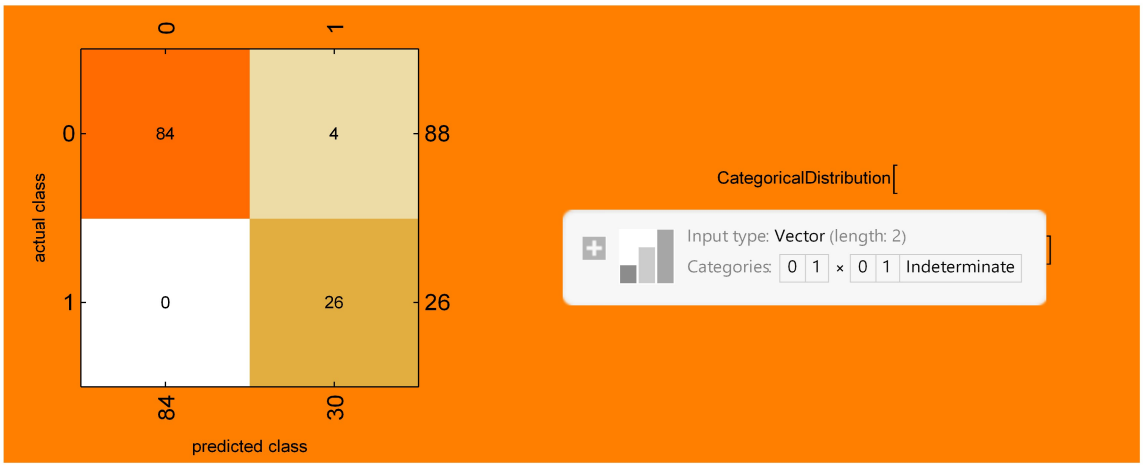
In[52]:=
```
PROPERTIES = {"Accuracy", "EvaluationTime", "Error",
   "FScore",  "Precision", "Recall", "Specificity", "AreaUnderROCCurve"};
For[i = 1, i ≤ Length[PROPERTIES], i++,
 Print[PROPERTIES[[i]], ": ", cmRF[PROPERTIES[[i]]]]]
```

```
Accuracy: 0.964912

EvaluationTime: 0.00619

Error: 0.0350877

FScore: ⟨|0 → 0.976744, 1 → 0.928571|⟩

Precision: ⟨|0 → 1., 1 → 0.866667|⟩

Recall: ⟨|0 → 0.954545, 1 → 1.|⟩

Specificity: ⟨|0 → 1., 1 → 0.954545|⟩

AreaUnderROCCurve: ⟨|0 → 0.998252, 1 → 0.997815|⟩
```

Confusion Matrix Plot and Confusion Distribution

In[54]:=
```
GraphicsRow[
 {cmRF["ConfusionMatrixPlot"], cmRF["ConfusionDistribution"]}, Background → Orange]
```

Out[54]=



Model Evaluations

In[55]:=
```
Info /@ algoRF
```

Out[55]=
```
ClassifierFunction[
```

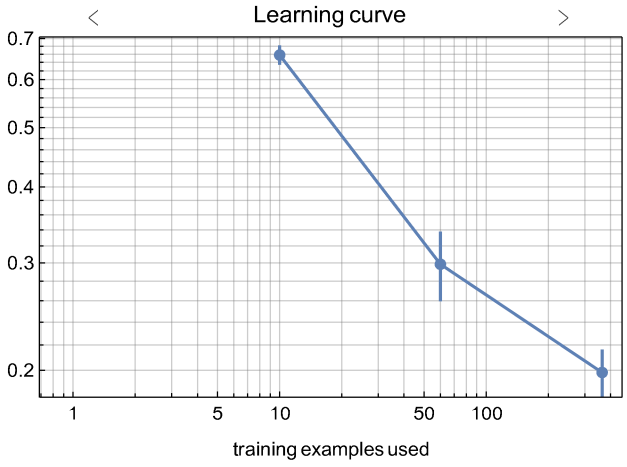

```
]
```

In[56]:=
```
Map[ModelInformation, {algoRF}]
```

Out[56]=

Classifier information

Data type	Mixed (number: 30)
Classes	0, 1
Accuracy	(93.7 ±1.6)%
Method	RandomForest
Single evaluation time	4.79 ms/example
Batch evaluation speed	20.8 examples/ms
Loss	0.198 ± 0.017
Model memory	309. kB
Training examples used	455 examples
Training time	4.98 s



Logistic Regression Classification Algorithm

Fitting the model

In[57]:=
```
Set[algoLR, Classify[trainFinal, Method → "LogisticRegression"]]
```

Out[57]=
```
ClassifierFunction[ ]
```
Input type: Mixed (number: 30)
Classes: 0, 1

About the algorithm

In[58]:=
```
ClassifierInformation[algoLR, "MethodDescription"]
```

Out[58]=

The logistic regression classifier models class probabilities with logistic functions of linear combinations of features. It is also called log-linear model, softmax regression, or maximum-entropy classifier.

Evaluation Measurements

PROPERTIES :
{Accuracy, AccuracyBaseline, AccuracyRejectionPlot, AreaUnderROCCurve, BatchEvaluationTime, BestClassifiedExamples, ClassifierFunction, ClassMeanCrossEntropy, ClassRejectionRate, CohenKappa, ConfusionDistribution, ConfusionFunction, ConfusionMatrix, ConfusionMatrixPlot, CorrectlyClassifiedExamples, DecisionUtilities, Error, EvaluationTime, Examples, F1Score, FalseDiscoveryRate, FalseNegativeExamples, FalseNegativeNumber, FalseNegativeRate, FalsePositiveExamples, FalsePositiveNumber, FalsePositiveRate, GeometricMeanProbability, IndeterminateExamples, LeastCertainExamples, Likelihood, LogLikelihood, MatthewsCorrelationCoefficient, MeanCrossEntropy, MeanDecisionUtility, MisclassifiedExamples, MostCertainExamples, NegativePredictiveValue, Perplexity, Precision, Probabilities, ProbabilityHistogram, Properties, Recall, RejectionRate, Report, ROCCurve, ScottPi, Specificity, TopConfusions, TrueNegativeExamples, TrueNegativeNumber, TruePositiveExamples, TruePositiveNumber, WorstClassifiedExamples}

In[59]:=
```
cmLR = ClassifierMeasurements[algoLR, testFinal]
```

Out[59]=
```
ClassifierMeasurementsObject[ Classifier: LogisticRegression
                              Number of test examples: 114 ]
```

```
(* cmLR/@{"Accuracy","FScore","Error",  "Precision"}//TableForm *)
```

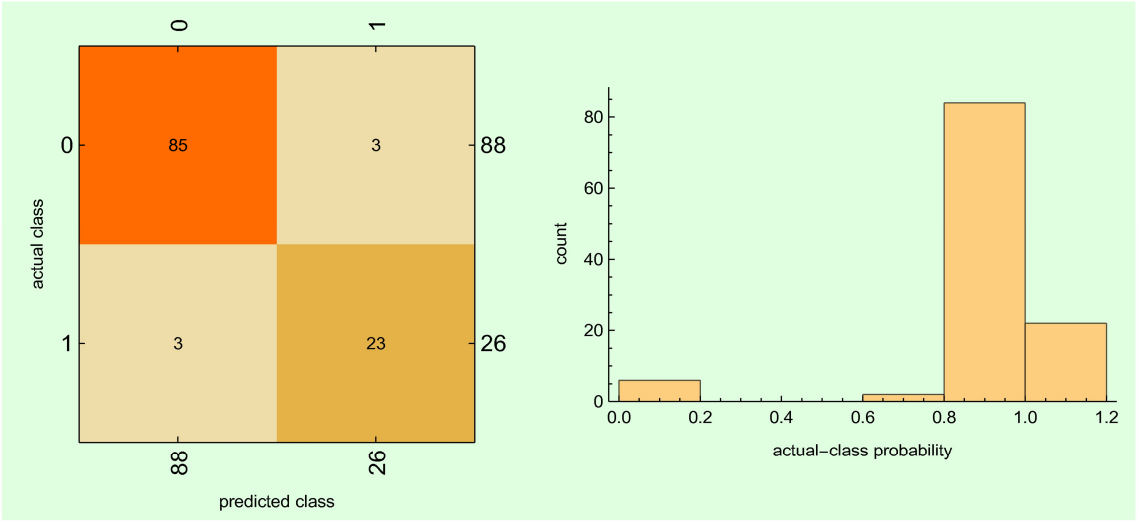
In[60]:=
```
PROPERTIES = {"Accuracy", "EvaluationTime", "Error",
   "FScore", "Precision", "Recall", "Specificity", "AreaUnderROCCurve"};
i = 1; While[i ≤ Length[PROPERTIES], Print[PROPERTIES[[i]],
  ": ", cmLR[PROPERTIES[[i]]]]; i++]
```

```
Accuracy: 0.947368
EvaluationTime: 0.0030
Error: 0.0526316
FScore: ⟨|0 → 0.965909, 1 → 0.884615|⟩
Precision: ⟨|0 → 0.965909, 1 → 0.884615|⟩
Recall: ⟨|0 → 0.965909, 1 → 0.884615|⟩
Specificity: ⟨|0 → 0.884615, 1 → 0.965909|⟩
AreaUnderROCCurve: ⟨|0 → 0.982517, 1 → 0.982517|⟩
```

Confusion Matrix Plot and Probability Histogram

In[62]:=
```
GraphicsRow[{cmLR["ConfusionMatrixPlot"], cmLR["ProbabilityHistogram"]},
 Background → LightGreen]
```

Out[62]=



Model Evaluations

In[63]:=
```
Info[algoLR]
```

Out[63]=
```
ClassifierFunction[
```

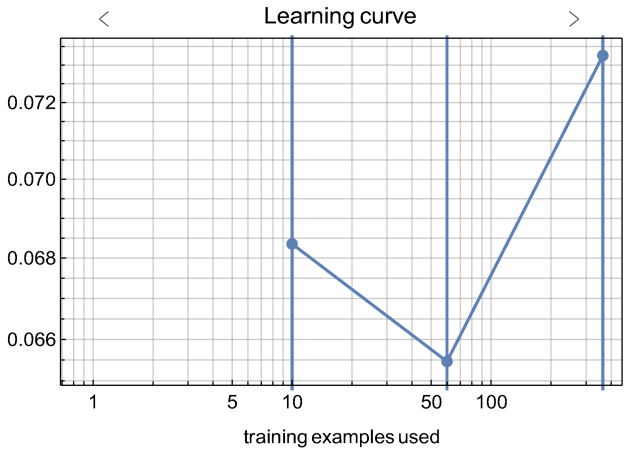
Input type: Mixed (number: 30)
Classes: 0, 1
```
]
```

In[64]:=
```
Map[ModelInformation[#] &, {algoLR}]
```

Out[64]=

Classifier information

Data type	Mixed (number: 30)
Classes	0, 1
Accuracy	(96.5 ±1.4)%
Method	LogisticRegression
Single evaluation time	2.54 ms/example
Batch evaluation speed	53.7 examples/ms
Loss	0.0655 ± 0.038
Model memory	272. kB
Training examples used	455 examples
Training time	1.72 s



Nearest Neighbors Classification Algorithm

Fitting the model

In[65]:=
```
Set[algoKNN, Classify[trainFinal, Method → "NearestNeighbors"]]
```

Out[65]=
```
ClassifierFunction[
```

Input type: Mixed (number: 30)
Classes: 0, 1
]

About the algorithm

In[66]:=
```
ClassifierInformation[algoKNN, "MethodDescription"]
```

Out[66]=

The nearest neighbors classifier infers the class of a new example by analyzing its nearest neighbors in the feature space. In its simplest form, it picks the commonest class amongst the "k"-nearest neighbors.

Evaluation Measurements

PROPERTIES :
{Accuracy, AccuracyBaseline, AccuracyRejectionPlot, AreaUnderROCCurve, BatchEvaluationTime, BestClassifiedExamples, ClassifierFunction, ClassMeanCrossEntropy, ClassRejectionRate, CohenKappa, ConfusionDistribution, ConfusionFunction, ConfusionMatrix, ConfusionMatrixPlot, CorrectlyClassifiedExamples, DecisionUtilities, Error, EvaluationTime, Examples, F1Score, FalseDiscoveryRate, FalseNegativeExamples, FalseNegativeNumber, FalseNegativeRate, FalsePositiveExamples, FalsePositiveNumber, FalsePositiveRate, GeometricMeanProbability, IndeterminateExamples, LeastCertainExamples, Likelihood, LogLikelihood, MatthewsCorrelationCoefficient, MeanCrossEntropy, MeanDecisionUtility, MisclassifiedExamples, MostCertainExamples, NegativePredictiveValue, Perplexity, Precision, Probabilities, ProbabilityHistogram, Properties, Recall, RejectionRate, Report, ROCCurve, ScottPi, Specificity, TopConfusions, TrueNegativeExamples, TrueNegativeNumber, TruePositiveExamples, TruePositiveNumber, WorstClassifiedExamples}

In[67]:=
```
cmKNN = ClassifierMeasurements[algoKNN, testFinal]
```

Out[67]=
```
ClassifierMeasurementsObject[ Classifier: NearestNeighbors
                              Number of test examples: 114 ]
```

```
(* cmKNN/@{"Accuracy","FScore","Error",  "Precision"}//TableForm *)
```

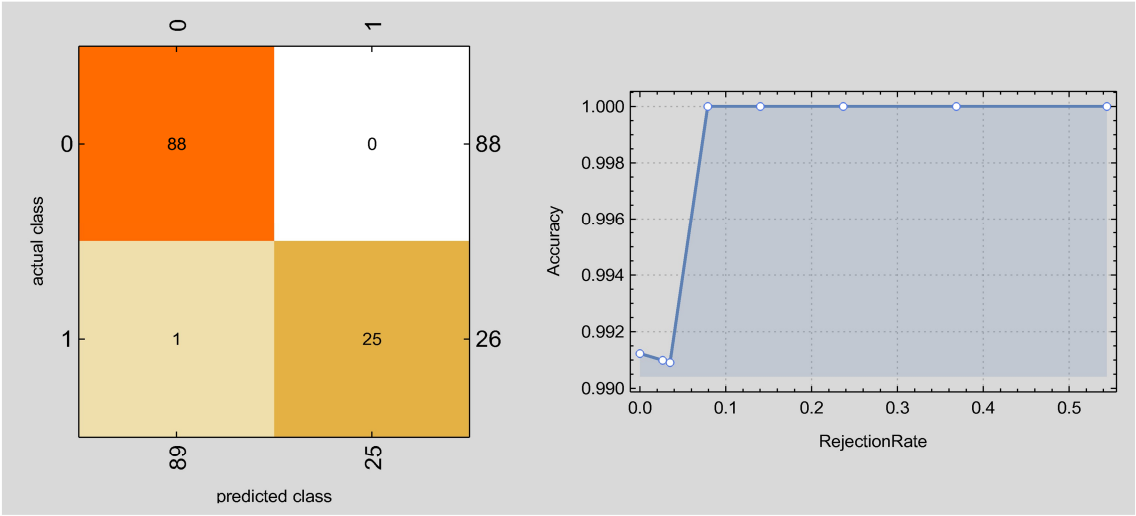
In[68]:=
```
PROPERTIES = {"Accuracy", "EvaluationTime", "Error",
   "FScore", "Precision", "Recall", "Specificity", "AreaUnderROCCurve"};
For[i = 1, i ≤ Length[PROPERTIES], i++,
 Print[PROPERTIES[[i]], ": ", cmKNN[PROPERTIES[[i]]]]]
```

```
Accuracy: 0.991228

EvaluationTime: 0.0034

Error: 0.00877193

FScore: ⟨|0 → 0.99435, 1 → 0.980392|⟩

Precision: ⟨|0 → 0.988764, 1 → 1.|⟩

Recall: ⟨|0 → 1., 1 → 0.961538|⟩

Specificity: ⟨|0 → 0.961538, 1 → 1.|⟩

AreaUnderROCCurve: ⟨|0 → 0.997815, 1 → 0.997815|⟩
```

Confusion Matrix Plot and Accuracy vs Rejection Plot

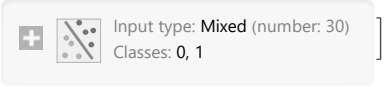
In[72]:=
```
GraphicsRow[{cmKNN["ConfusionMatrixPlot"], cmKNN["AccuracyRejectionPlot"]},
 Background → LightGray]
```

Out[72]=



Model Evaluations

In[73]:=
```
Info[algoKNN]
```

Out[73]=
```
ClassifierFunction[
```


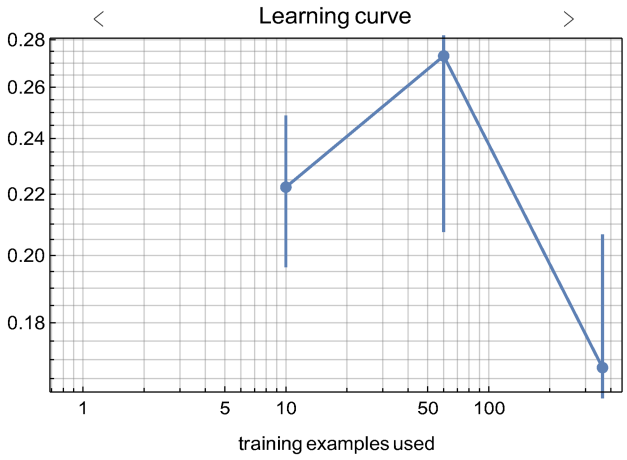
```
]
```

In[74]:=
```
Map[ModelInformation , {algoKNN}]
```

Out[74]=

Classifier information

Data type	Mixed (number: 30)
Classes	0, 1
Accuracy	(96.2 ±1.9)%
Method	NearestNeighbors
Single evaluation time	2.65 ms/example
Batch evaluation speed	37.1 examples/ms
Loss	0.168 ± 0.038
Model memory	324. kB
Training examples used	455 examples
Training time	722. ms



Support Vector Machine Classification Algorithm

Fitting the model

In[75]:=
```
Set[algoSVM, Classify[trainFinal, Method → "SupportVectorMachine"]]
```

Out[75]=
```
ClassifierFunction[
```
Input type: Mixed (number: 30)
Classes: 0, 1
```
]
```

About the algorithm

In[76]:=
```
ClassifierInformation[algoSVM, "MethodDescription"]
```

Out[76]=

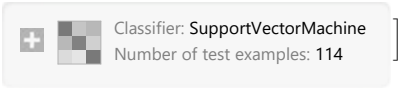
The support vector machine classifier separates the training data into two classes using a maximum–margin hyperplane. The original feature space can be mapped into a higher dimensional space to improve linear separability. The multi–class classification problem is reduced to a set of binary classification problems.

Evaluation Measurements

PROPERTIES :
{Accuracy, AccuracyBaseline, AccuracyRejectionPlot, AreaUnderROCCurve, BatchEvaluationTime, BestClassifiedExamples, ClassifierFunction, ClassMeanCrossEntropy, ClassRejectionRate, CohenKappa, ConfusionDistribution, ConfusionFunction, ConfusionMatrix, ConfusionMatrixPlot, CorrectlyClassifiedExamples, DecisionUtilities, Error, EvaluationTime, Examples, F1Score, FalseDiscoveryRate, FalseNegativeExamples, FalseNegativeNumber, FalseNegativeRate, FalsePositiveExamples, FalsePositiveNumber, FalsePositiveRate, GeometricMeanProbability, IndeterminateExamples, LeastCertainExamples, Likelihood, LogLikelihood, MatthewsCorrelationCoefficient, MeanCrossEntropy, MeanDecisionUtility, MisclassifiedExamples, MostCertainExamples, NegativePredictiveValue, Perplexity, Precision, Probabilities, ProbabilityHistogram, Properties, Recall, RejectionRate, Report, ROCCurve, ScottPi, Specificity, TopConfusions, TrueNegativeExamples, TrueNegativeNumber, TruePositiveExamples, TruePositiveNumber, WorstClassifiedExamples}

In[77]:=
```
cmSVM = ClassifierMeasurements[algoSVM, testFinal]
```

Out[77]=




```
(* cmSVM/@{"Accuracy","FScore","Error",  "Precision"}//TableForm *)
```

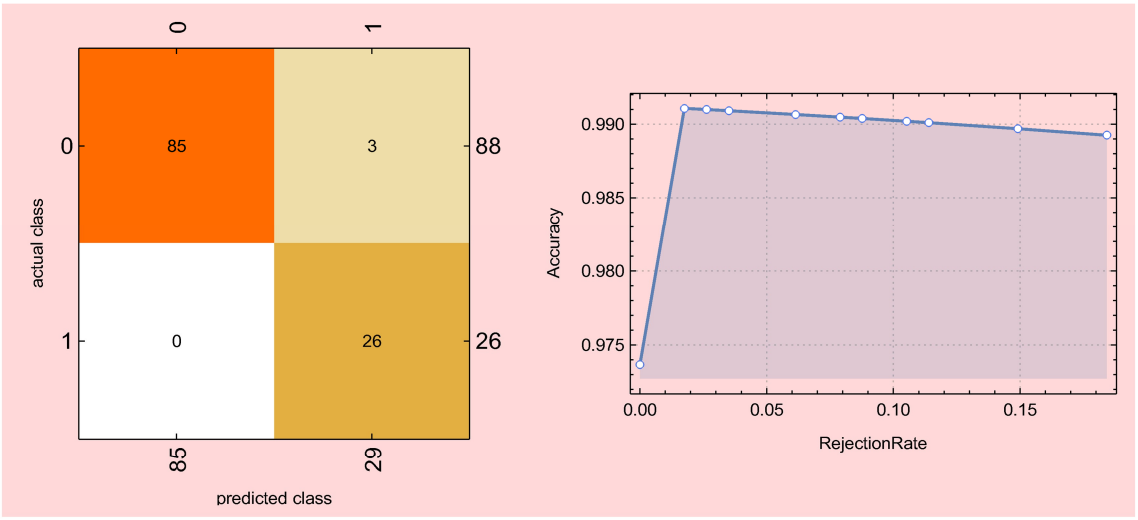
In[78]:=
```
PROPERTIES = {"Accuracy", "EvaluationTime", "Error",
   "FScore", "Precision", "Recall", "Specificity", "AreaUnderROCCurve"};
For[i = 1, i ≤ Length[PROPERTIES], i++,
 Print[PROPERTIES[[i]], ": ", cmSVM[PROPERTIES[[i]]]]]
```

```
Accuracy: 0.973684

EvaluationTime: 0.0050

Error: 0.0263158

FScore: ⟨|0 → 0.982659, 1 → 0.945455|⟩

Precision: ⟨|0 → 1., 1 → 0.896552|⟩

Recall: ⟨|0 → 0.965909, 1 → 1.|⟩

Specificity: ⟨|0 → 1., 1 → 0.965909|⟩

AreaUnderROCCurve: ⟨|0 → 0.998689, 1 → 0.998689|⟩
```

Confusion Matrix Plot and Accuracy vs Rejection Plot

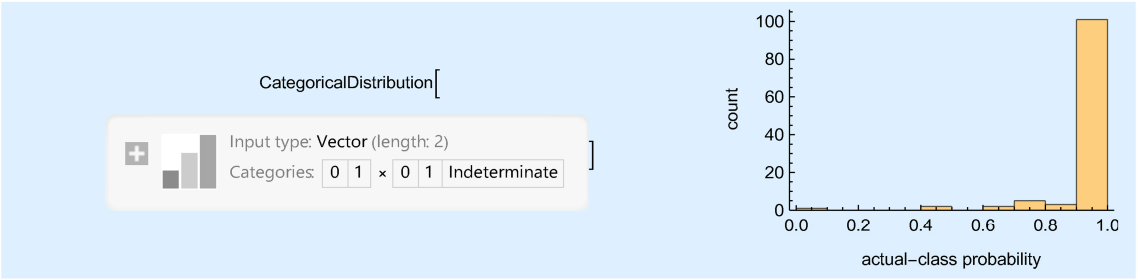
In[80]:=
```
GraphicsRow[{cmSVM["ConfusionMatrixPlot"], cmSVM["AccuracyRejectionPlot"]},
 Background → LightRed]
```

Out[80]=



In[81]:=
```
GraphicsRow[{cmSVM["ConfusionDistribution"], cmSVM["ProbabilityHistogram"]},
 Background → LightBlue]
```

Out[81]=



Model Evaluations

In[82]:=
```
Map[Info, algoSVM]
```

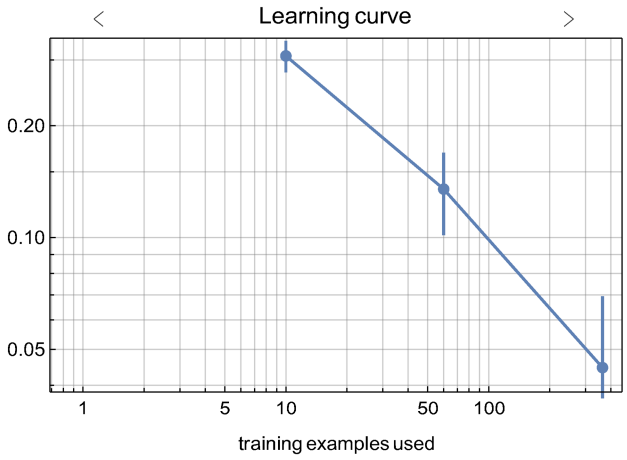
Out[82]=

In[83]:=
```
ModelInformation /@ {algoSVM}
```

Out[83]=

Classifier information

Data type	Mixed (number: 30)
Classes	0, 1
Accuracy	(98.4 ± 2.3)%
Method	SupportVectorMachine
Single evaluation time	3.89 ms/example
Batch evaluation speed	29.6 examples/ms
Loss	0.0446 ± 0.024
Model memory	353. kB
Training examples used	455 examples
Training time	2.72 s



Neural Network Algorithm

Fitting the model

In[84]:=
```
Set[algoNN, Classify[trainFinal, Method → "NeuralNetwork", TargetDevice → "GPU"]]
```

Out[84]=
```
ClassifierFunction[
```

Input type: Mixed (number: 30)
Classes: 0, 1
]

About the algorithm

In[85]:=
```
ClassifierInformation[algoNN, "MethodDescription"]
```

Out[85]=

An neural networks is composed of layers of artificial neuron units. Each unit computes its value as a function of the unit values in the previous layer. Information is processed layer by layer from the feature layer to the output layer which gives the class probabilities. It is also called a feed–forward neural network or a multi–layer perceptron.

Evaluation Measurements

PROPERTIES :

{Accuracy, AccuracyBaseline, AccuracyRejectionPlot, AreaUnderROCCurve, BatchEvaluationTime, BestClassifiedExamples, ClassifierFunction, ClassMeanCrossEntropy, ClassRejectionRate, CohenKappa, ConfusionDistribution, ConfusionFunction, ConfusionMatrix, ConfusionMatrixPlot, CorrectlyClassifiedExamples, DecisionUtilities, Error, EvaluationTime, Examples, F1Score, FalseDiscoveryRate, FalseNegativeExamples, FalseNegativeNumber, FalseNegativeRate, FalsePositiveExamples, FalsePositiveNumber, FalsePositiveRate, GeometricMeanProbability, IndeterminateExamples, LeastCertainExamples, Likelihood, LogLikelihood, MatthewsCorrelationCoefficient, MeanCrossEntropy, MeanDecisionUtility, MisclassifiedExamples, MostCertainExamples, NegativePredictiveValue, Perplexity, Precision, Probabilities, ProbabilityHistogram, Properties, Recall, RejectionRate, Report, ROCCurve, ScottPi, Specificity, TopConfusions, TrueNegativeExamples, TrueNegativeNumber, TruePositiveExamples, TruePositiveNumber, WorstClassifiedExamples}

In[86]:=
```
cmNN = ClassifierMeasurements[algoNN, testFinal]
```

Out[86]=
```
ClassifierMeasurementsObject[ Classifier: NeuralNetwork
                              Number of test examples: 114 ]
```

```
(* cmNN/@{"Accuracy","FScore","Error",  "Precision"}//TableForm *)
```

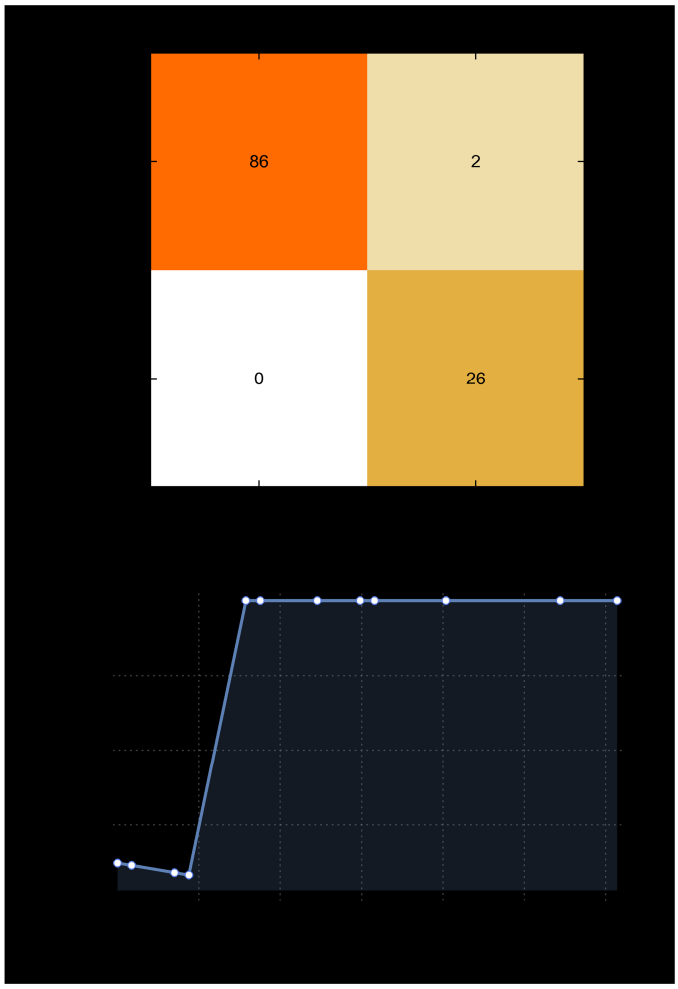
In[87]:=
```
PROPERTIES = {"Accuracy", "EvaluationTime", "Error",
   "FScore", "Precision", "Recall", "Specificity", "AreaUnderROCCurve"};
Do[Print[PROPERTIES[[i]], ": ", cmNN[PROPERTIES[[i]]]], {i, Length[PROPERTIES]}]
```

```
Accuracy: 0.982456

EvaluationTime: 0.0031

Error: 0.0175439

FScore: ⟨|0 → 0.988506, 1 → 0.962963|⟩

Precision: ⟨|0 → 1., 1 → 0.928571|⟩

Recall: ⟨|0 → 0.977273, 1 → 1.|⟩

Specificity: ⟨|0 → 1., 1 → 0.977273|⟩

AreaUnderROCCurve: ⟨|0 → 0.999126, 1 → 0.999126|⟩
```

Confusion Matrix Plot and Accuracy vs Rejection Plot

In[89]:=
```
GraphicsColumn[
 {cmNN["ConfusionMatrixPlot"], cmNN["AccuracyRejectionPlot"]}, Background → Black]
```

Out[89]=

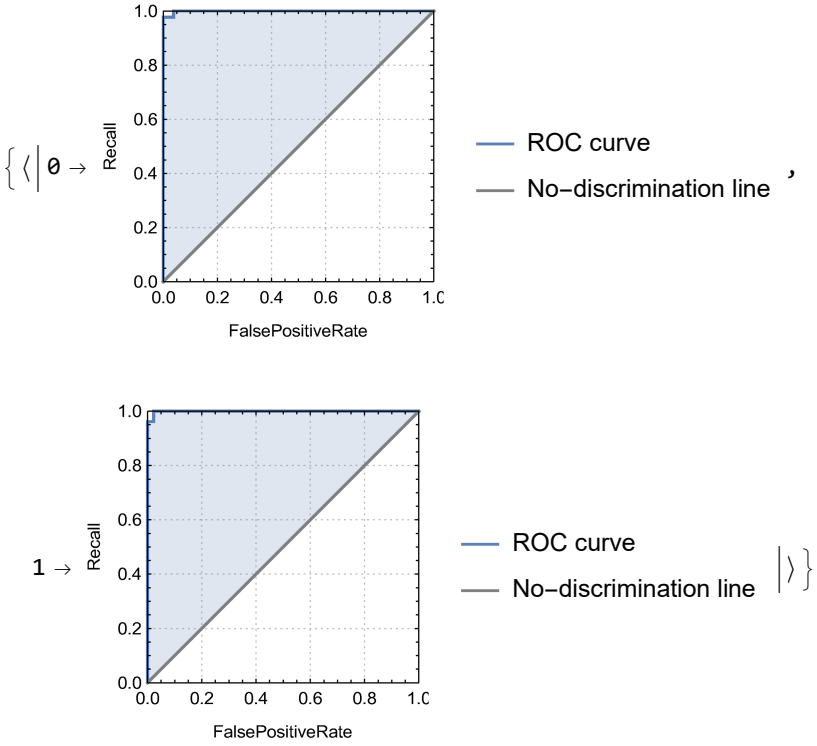


In[92]:=
```
cmNN["ConfusionDistribution"]
```

Out[92]=
```
CategoricalDistribution[
```
Input type: Vector (length: 2)
Categories: 0 1 × 0 1 Indeterminate
```
]
```

In[90]:=
```
cmNN@{"ROCCurve"}
```

Out[90]=



Model Evaluations

In[93]:=
```
Info /@ algoNN
```

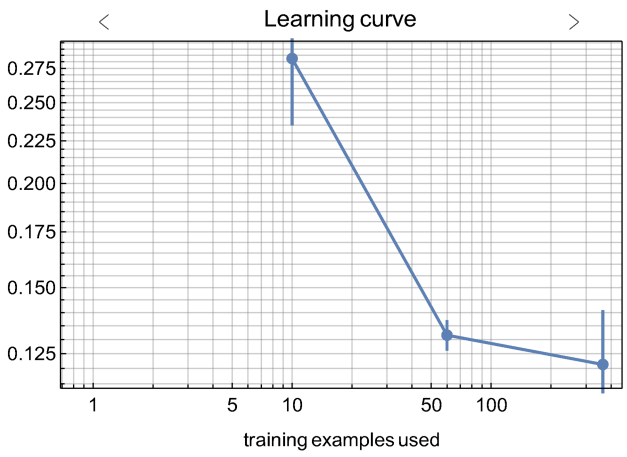
Out[93]=

In[94]:=
```
Map[ModelInformation[#] &, {algoNN}]
```

Out[94]=

Classifier information

Data type	Mixed (number: 30)
Classes	0, 1
Accuracy	(96.3 ± 0.8)%
Method	NeuralNetwork
Single evaluation time	2.27 ms/example
Batch evaluation speed	45.4 examples/ms
Loss	0.121 ± 0.019
Model memory	835. kB
Training examples used	455 examples
Training time	1h 21 min



Evaluation

In[ ]:=

ALGORITHMS	ACCURACY	F – SCORE	ERROR	PRECISION
NaiveBayes	0.8684	⟨|0 → 0.9534, 1 → 0.8571|⟩	0.0701	⟨|0 → 0.9761, 1 → 0.8000|⟩
DecisionTree	0.9298	⟨|0 → 0.9090, 1 → 0.7619|⟩	0.1315	⟨|0 → 0.9740, 1 → 0.6486|⟩
RandomForest	0.9649	⟨|0 → 0.9767, 1 → 0.9285|⟩	0.0350	⟨|0 → 1.000, 1 → 0.8667|⟩
LogisticRegression	0.9473	⟨|0 → 0.9659, 1 → 0.8846|⟩	0.0526	⟨|0 → 0.9659, 1 → 0.8846|⟩
KNearestNeighbour	**0.9912**	**⟨|0 → 0.9943, 1 → 0.9803|⟩**	**0.0087**	**⟨|0 → 0.9887, 1 → 1.0000|⟩**
SupportVectorMachine	0.9736	⟨|0 → 0.9826, 1 → 0.9454|⟩	0.0263	⟨|0 → 1.0000, 1 → 0.8965|⟩
NeuralNetwork	0.9473	⟨|0 → 0.9651, 1 → 0.8928|⟩	0.0526	⟨|0 → 0.9880, 1 → 0.8334|⟩

```
// TableForm;
```

Fitting Models with Highly Correlated Features and Evaluation

Highly correlated features

In[95]:=
```
CorrelationFn[x_, y_] := Correlation[train[All, x] // Normal, train[All, y] // Normal]
```

Fetching Highly Correlated Features from the data

In[97]:=
```
For[i = 1, i ≤ Length[header] - 2, i++,
     For[k = i + 1, k ≤ Length[header] - 1, k++,
  If[CorrelationFn[header[[i]], header[[k]]] > 0.85, Print["CORRELATION :: ",
    header[[i]], "-", header[[k]], "\n", CorrelationFn[header[[i]], header[[k]]]]]]]
```

```
CORRELATION :: radius_mean-perimeter_mean
0.997697

CORRELATION :: radius_mean-area_mean
0.988989

CORRELATION :: radius_mean-radius_worst
0.965381

CORRELATION :: radius_mean-perimeter_worst
0.961365
```

```
CORRELATION :: radius_mean-area_worst
0.939357

CORRELATION :: texture_mean-texture_worst
0.902807

CORRELATION :: perimeter_mean-area_mean
0.987604

CORRELATION :: perimeter_mean-radius_worst
0.965292

CORRELATION :: perimeter_mean-perimeter_worst
0.966843

CORRELATION :: perimeter_mean-area_worst
0.939192

CORRELATION :: area_mean-radius_worst
0.957439

CORRELATION :: area_mean-perimeter_worst
0.953573

CORRELATION :: area_mean-area_worst
0.952004

CORRELATION :: compactness_mean-concavity_mean
0.893376

CORRELATION :: compactness_mean-compactness_worst
0.86917

CORRELATION :: concavity_mean-concave points_mean
0.916922

CORRELATION :: concavity_mean-concavity_worst
0.883579

CORRELATION :: concavity_mean-concave points_worst
0.852318

CORRELATION :: concave points_mean-perimeter_worst
0.851339

CORRELATION :: concave points_mean-concave points_worst
0.906077

CORRELATION :: radius_se-perimeter_se
0.97009

CORRELATION :: radius_se-area_se
0.954597

CORRELATION :: perimeter_se-area_se
0.938311

CORRELATION :: radius_worst-perimeter_worst
0.993583

CORRELATION :: radius_worst-area_worst
0.986142

CORRELATION :: perimeter_worst-area_worst
0.978694
```

```
CORRELATION :: compactness_worst-concavity_worst
0.889038
CORRELATION :: concavity_worst-concave points_worst
0.857839
```

In[98]:=
```
CorrelatedFeatures = {"radius_mean", "texture_mean", "perimeter_mean", "area_mean",
   "compactness_mean", "concavity_mean", "concave points_mean", "radius_se",
   "perimeter_se", "area_se", "radius_worst", "perimeter_worst", "area_worst",
   "compactness_worst", "concavity_worst", "concave points_worst", "diagnosis"};
```

Data Visualization

In[99]:=
```
Needs["StatisticalPlots`"]
```

In[100]:=
```
pairUp[x_, y_] := ({x[[#]], y[[#]]}) & /@ Range[Min[Length[x], Length[y]]];
```

Pair Wise Scatter Plot

In[105]:=
```
SlideView[{PairwiseScatterPlot[
    pairUp[train[All, "concave points_mean"], train[All, "radius_se"]], Background → Blue],
   PairwiseScatterPlot[pairUp[train[All, "concavity_mean"], train[All, "perimeter_se"]],
    Background → Red], PairwiseScatterPlot[
    pairUp[train[All, "compactness_mean"], train[All, "area_se"]], Background → Green],
   PairwiseScatterPlot[pairUp[train[All, "area_mean"], train[All, "radius_worst"]],
    Background → Orange], PairwiseScatterPlot[pairUp[train[All, "perimeter_mean"],
     train[All, "perimeter_worst"]], Background → LightBlue], PairwiseScatterPlot[
    pairUp[train[All, "texture_mean"], train[All, "area_worst"]], Background → Brown],
   PairwiseScatterPlot[pairUp[train[All, "radius_mean"], train[All, "compactness_worst"]],
    Background → Blue], PairwiseScatterPlot[
    pairUp[train[All, "concave points_worst"], train[All, "concavity_worst"]],
    Background → Red], PairwiseScatterPlot[pairUp[train[All, "concavity_worst"],
     train[All, "concave points_worst"]], Background → Green],
   PairwiseScatterPlot[pairUp[train[All, "compactness_worst"], train[All, "radius_mean"]],
    Background → Orange], PairwiseScatterPlot[
    pairUp[train[All, "area_worst"], train[All, "texture_mean"]], Background → LightBlue],
   PairwiseScatterPlot[pairUp[train[All, "perimeter_worst"],
     train[All, "perimeter_mean"]], Background → Brown], PairwiseScatterPlot[
    pairUp[train[All, "radius_worst"], train[All, "area_mean"]], Background → Blue],
   PairwiseScatterPlot[pairUp[train[All, "area_se"], train[All, "compactness_mean"]],
    Background → Red], PairwiseScatterPlot[
    pairUp[train[All, "perimeter_se"], train[All, "concavity_mean"]], Background → Green],
   PairwiseScatterPlot[pairUp[train[All, "radius_se"], train[All, "concave points_mean"]],
    Background → Orange]}, ImageSize → Automatic,
  ImageMargins → 11, Deployed → True, AppearanceElements → All]
```

Out[105]=

Fitting New Features into the Model

In[106]:=
```
trainFeatures = train[1 ;; Length[train], CorrelatedFeatures];
testFeatures = test[1 ;; Length[test], CorrelatedFeatures];
```

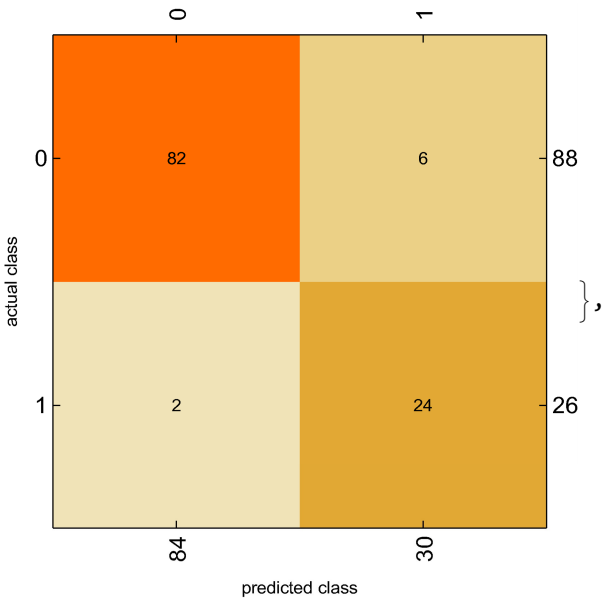
Flattening the Features

In[108]:=
```
trainFinal =
  Flatten[Normal@trainFeatures[1 ;; Length[trainPrep], {Most@# → Last@#} &], 1];
testFinal = Flatten[Normal@testFeatures[1 ;; Length[testPrep], {Most@# → Last@#} &], 1];
```

Building the Models and Evaluations

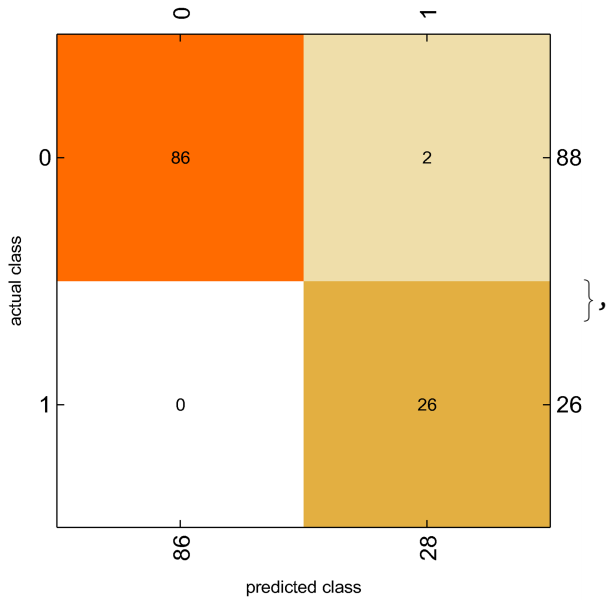
In[110]:=
```
AssociationMap[ClassifierMeasurements[Classify[trainFinal, Method → #],
    testFinal, {"Accuracy", "EvaluationTime", "ConfusionMatrixPlot"}] &,
 {"DecisionTree", "RandomForest", "NaiveBayes", "SupportVectorMachine",
  "NearestNeighbors", "LogisticRegression", "NeuralNetwork"}]
```

Out[110]=
```
⟨|DecisionTree → {0.929825, 0.00254,
```



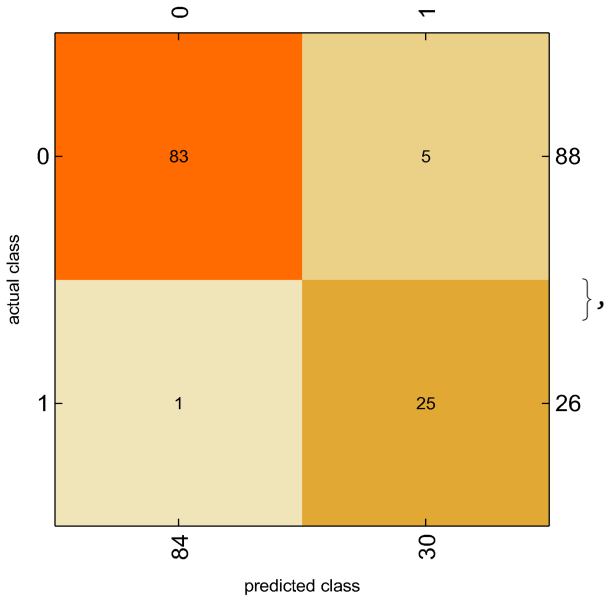

```
}, ...
```

RandomForest → {0.982456, 0.0040,



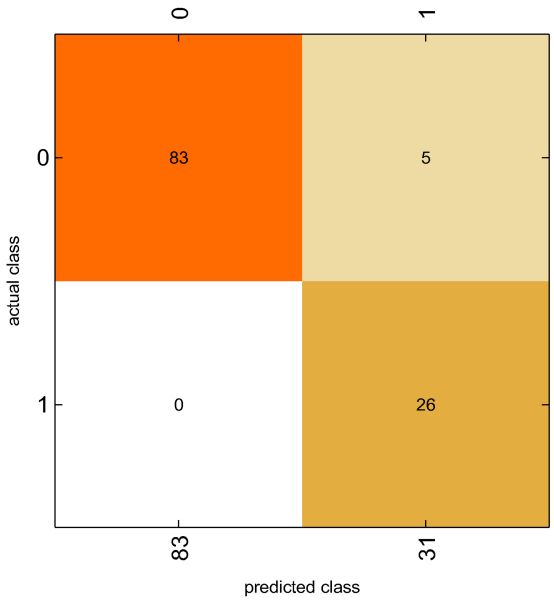
},

NaiveBayes → {0.947368, 0.0043,



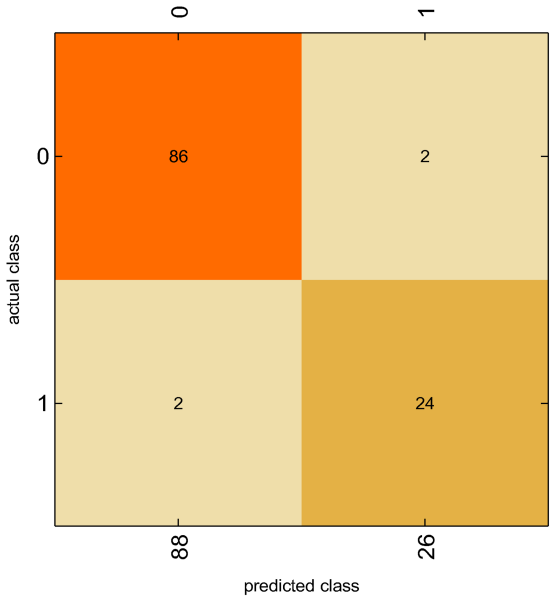
},

SupportVectorMachine → {0.95614, 0.0041,



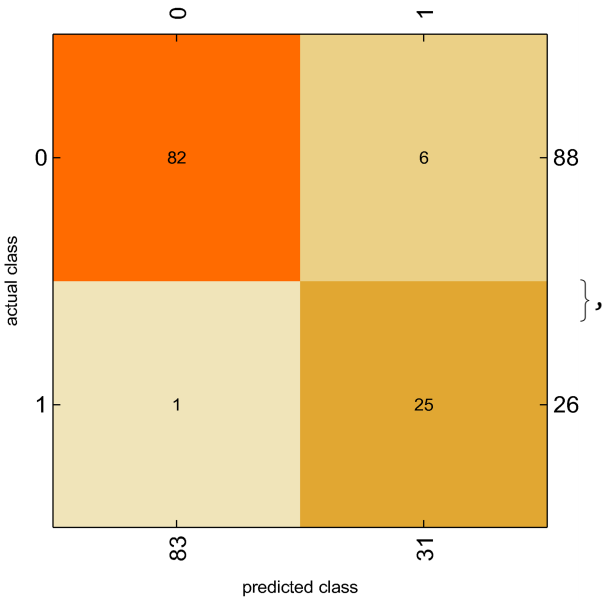
},

NearestNeighbors → {0.964912, 0.0028,



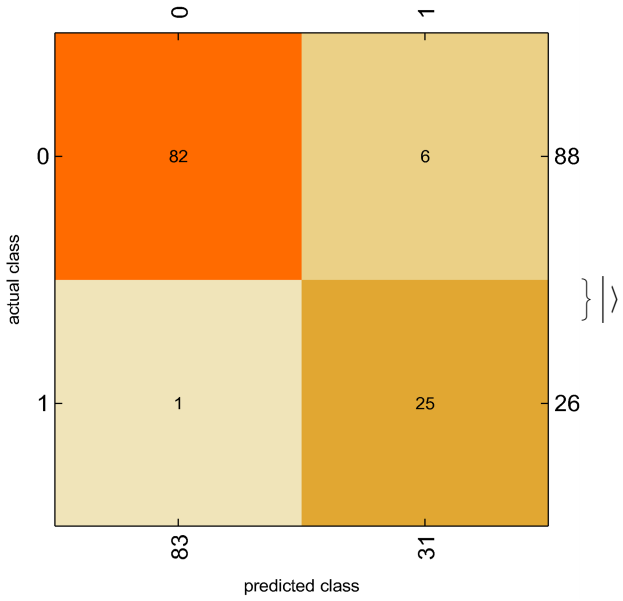
},

LogisticRegression → {0.938596, 0.0025,



},

NeuralNetwork → {0.938596, 0.0028,



}|⟩

Some Trained Neural Networks

First Model

```
model1 = NetInitialize@NetChain[{10, LogisticSigmoid, 2, SoftmaxLayer[]},
   "Input" → {31}, "Output" → NetDecoder[{"Class", {0, 1}}]]
results1 = NetTrain[net, train, All, MaxTrainingRounds → 1000]
trained1 = results["TrainedNet"];
```

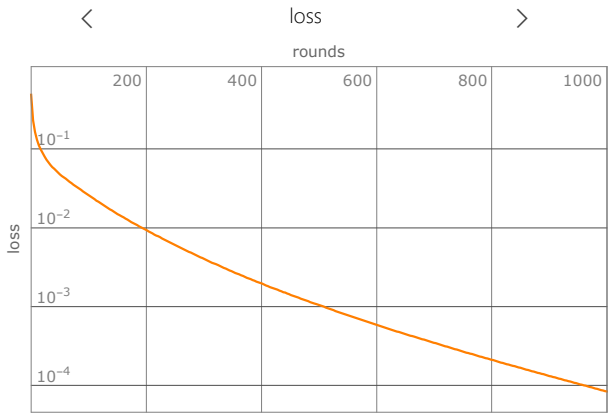
Trained Network

Out[ ]= NetChain [Input port: vector (size: 31) Output port: class]

NetTrain Results

summary	batches: 7000, rounds: 1000, time: 5.4s, examples/s: 83 026
data	training examples: 400, processed examples: 448 000, skipped examples: 0
method	ADAM optimizer, batch size 64, CPU
round	loss: 8.32×10^{-5}, error: 0%

Out[ ]=



Second Model

In[369]:=
```
model2 =
 NetChain[{LinearLayer[30], ElementwiseLayer[Ramp], LinearLayer[2], SoftmaxLayer[]},
  "Input" → {31}, "Output" → NetDecoder[{"Class", {0, 1}}]]
results2 =  NetTrain[net, train, All, MaxTrainingRounds → 1000]
```

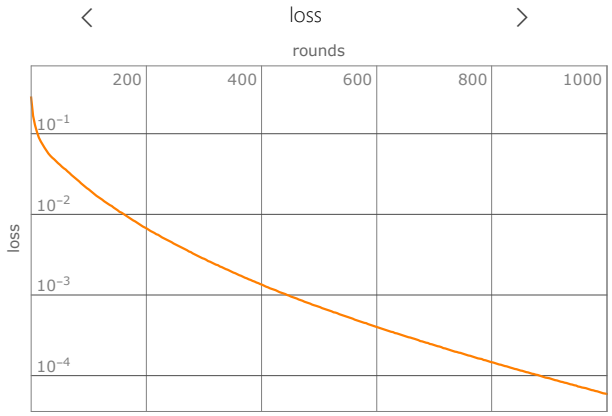
Out[369]= NetChain [uninitialized Input port: vector (size: 31) Output port: class]

Trained Network

Out[370]=

NetTrain Results

summary	batches: 7000, rounds: 1000, time: 5.3s, examples/s: 84 302
data	training examples: 400, processed examples: 448 000, skipped examples: 0
method	ADAM optimizer, batch size 64, CPU
round	loss: 5.92×10^{-5}, error: 0%



Third Model

Standardize Training and Testing Data

In[ ]:=
```
XtrainStandardized = Transpose@Map[Standardize, Transpose[traindata[[All, 1 ;; 31]]]];
Mean[XtrainStandardized]; // N
StandardDeviation[XtrainStandardized]; // N
```

In[ ]:=
```
XtestStandardized =
  Transpose@Table[(testdata[[All, i]] - Mean[traindata[[All, 1 ;; 31]]][[i]]) /
     StandardDeviation[traindata[[All, 1 ;; 31]]][[i]], {i, 31}];
Mean[XtestStandardized]; // N
StandardDeviation[XtestStandardized]; // N
```

Sampling Training and Testing Data

In[ ]:=
```
XtrainN = 400;
```

```
In[ ]:= trainT = Table[XtrainStandardized[[i, All]] → traindata[[All, 32]][[i]], {i, XtrainN}];
        Dimensions[XtrainStandardized]
        trainT[[ ;; 1]] // N
```

```
Out[ ]= {455, 31}
```

```
Out[ ]= {{-0.235825, 1.07241, -2.10875, 1.24768, 0.969138, 1.60098, 3.2042,
     2.58485, 2.47019, 2.11866, 2.29681, 2.52737, -0.556382, 2.87694, 2.67297,
     -0.198597, 1.24639, 0.664473, 0.631758, 1.04287, 0.861088, 1.84685, -1.33608,
     2.27488, 1.98188, 1.28822, 2.50837, 2.04389, 2.2167, 2.57443, 1.89354} → 1.}
```

```
In[ ]:= testT = Table[XtestStandardized[[i, All]] → testdata[[i]], {i, Length[testdata]}];
        Dimensions[testT]
        testT[[ ;; 1]] // N
```

```
Out[ ]= {114}
```

```
Out[ ]= {{-0.161653, -0.244306, 2.87318, -0.263811, -0.306928, -0.292917, -0.570627, -0.782561,
     -0.452191, -1.5938, -0.354458, -0.260925, 1.35935, -0.31197, -0.284836, -0.385183,
     -0.716829, -0.766152, -0.449341, -0.608474, -0.297109, -0.289551, 2.67826, -0.350785,
     -0.341876, -0.666813, -0.713425, -0.982979, -0.592132, -1.15505, -0.394348} →
    {9.11209 × 10^6, 13.38, 30.72, 86.34, 557.2, 0.09245, 0.07426, 0.02819,
     0.03264, 0.1375, 0.06016, 0.3408, 1.924, 2.287, 28.93, 0.005841,
     0.01246, 0.007936, 0.009128, 0.01564, 0.002985, 15.05, 41.61, 96.69,
     705.6, 0.1172, 0.1421, 0.07003, 0.07763, 0.2196, 0.07675, 0.}}
```

Creating Network

Network Decoder

```
In[ ]:= dr = NetDecoder[{"Class", {0, 1}}]
```

```
Out[ ]= NetDecoder[ Type: Class
                    Input: vector (size: 2) ]
```

Modelling

```
In[ ]:= SeedRandom[123]
        network =
         NetInitialize[NetChain[{5, Ramp, 2, SoftmaxLayer[]}, "Output" → dr, "Input" → 31],
          Method → {"Random", "Weights" → 0.01, "Biases" → 0}]
```

```
Out[ ]= NetChain[ Input port: vector (size: 31)
                  Output port: class ]
```

Network Extraction

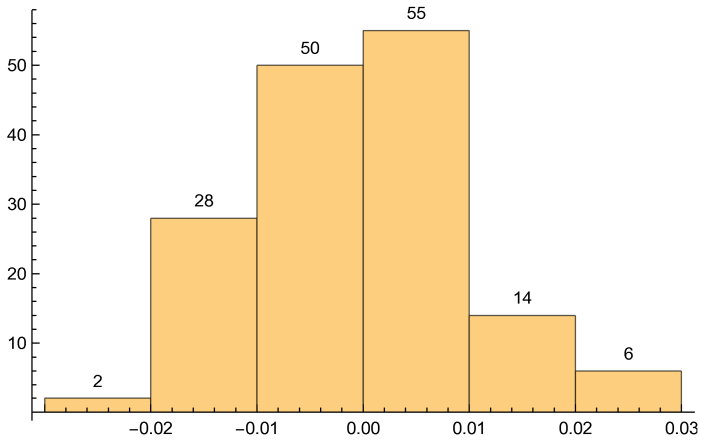
```
In[ ]:= NetExtract[network, {1, "Weights"}]
```

```
Out[ ]= NumericArray[ Type: Real32
                      Dimensions: {5, 31} ]
```

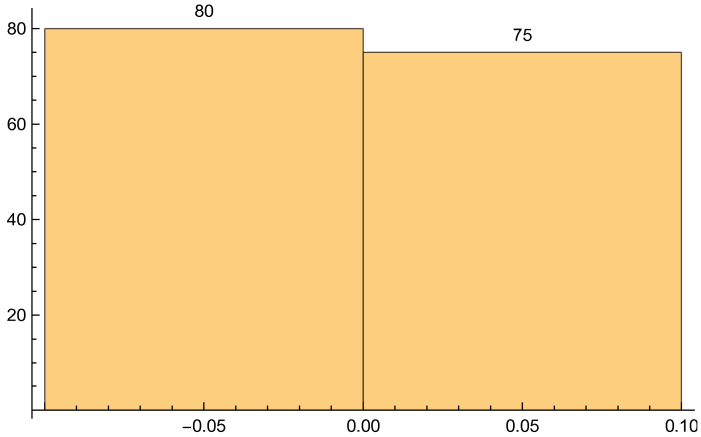
Network Analysis based on Weights and Bias using Histogram

```
In[ ]:= Histogram[Flatten[NetExtract[network, {1, "Weights"}]], {0.01},
         ChartLegends → "Network Analysis with Weights 1 and Bias 0.01",
         LabelingFunction → Above]
        Histogram[Flatten[NetExtract[network, {1, "Weights"}]], {0.1},
         ChartLegends → "Network Analysis with Weights 1 and Bias 0.1", LabelingFunction → Above]
```

Out[ ]=



Network Analysis with Weights 1 and Bia



Network Analysis with Weights 1 and Bia

Trained Network

In[372]:=
```
trained = NetTrain[network, train, All, MaxTrainingRounds → 1000]
```

Out[372]=

NetTrain Results

summary	batches: 7000, rounds: 1000, time: 5.3s, examples/s: 84 245
data	training examples: 400, processed examples: 448 000, skipped examples: 0
method	ADAM optimizer, batch size 64, CPU
round	loss: 1.3×10^{-3}, error: 0%

Conclusion

From the dataset used for breast cancer, without tuning the parameters and hyperparameters we observed that in initial model the best results has been shown by K-Nearest Neighbours with 99.12% and then by SVM algorithm with 97.36% while from the later model, fitting with highly correlated features have given maximum accuracy from Random Forest with 98.24%. Since, the model has obtained a reduce accuracy, it is clear that all the attributes are important to fit the model with higher accuracy (similar to the concept of R-Squared value where the this value always increases on adding more features to the data).

For the point of improvement, hyperparameters can be tuned with similar model, increasing the hidden layer and dynamics of the neural network can also result will give more profound result as we obtained while training the network with 3 optimized (i.e., over hidden layers, activation function, weights , bias and methods) models at the last. Moreover, Image classification can be more accurate and effective in case of Breast Cancer Classification, when integrated with the Data Resampling, Data Augmentation, Transfer Learning and building hybrid models.

References

[1]. Myers ER, Moorman P, Gierisch JM, et al. Benefits and harms of breast cancer screening: a systematic review. JAMA. 2015;314:1615–1634. [PubMed] [Google Scholar]
[2]. Group DES. Systematic review of cancer screening literature for updating American Cancer Society breast cancer screening guidelines. Duke Clinical Research Institute, Durham, NC: Guidelines Development Group. 2014 [Google Scholar]
[3]. Swedish Council on Health Technology Assessment. Computer-Aided Detection (CAD) in mammography screening. Stockholm: 2011. SBU systematic review summaries. [Google Scholar]
[4]. Mehdi Habibzadeh Motlagh, Mahboobeh Jannesari, HamidReza Aboulkheyr, Pegah Khosravi, Olivier Elemento, Mehdi Totonchi, and Iman Hajirasouliha
[5]. Babak Ehteshami Bejnordi, Guido Zuidhof, Maschenka Balkenhol, Meyke Hermsen, Peter Bult, Bram van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen van der Laak
[6]. Alexander Rakhlin, Alexey Shvets, Vladimir Iglovikov and Alexandr A. Kalinin
[7]. Nan Wu, Jason Phang, Jungkyu Park, Yiqiu Shen, Zhe Huang, Masha Zorin, Stanisław Jastrz ˛ebsk, Thibault Févrb, Joe Katsnelson, Eric Kim, Stacey Wolfson, Ujas Parikha, Sushma Gaddama, Leng Leng Young Lina, Kara Hoj* , Joshua D. Weinsteina, Beatriu Reig, Yiming Gao, Hildegard Totha, Kristine Pysarenkoa, Alana Lewina, Jiyon Leea, Krystal Airolaa, Eralda Memaa, Stephanie Chung, Esther Hwang, Naziya Samreena, S. Gene Kima, , Laura Heacocka, Linda Moya, Kyunghyun Chob, and Krzysztof J. Gerasa
[8]. Krzysztof J. Geras, Stacey Wolfson, Yiqiu Shen, Nan Wu, S. Gene Kim, Eric Kim, Laura Heacock, Ujas Parikh, Linda Moy, Kyunghyun Cho
[9]. Quinlan JR, Rivest RL. Inferring decision trees using the minimum description length principle. Information and Computation. 1989 March 1;80(3):227–48.
[10]. Quinlan JR. Simplifying decision trees. International Journal of Man-Machine Studies. 1987 September 1;27(3):221–34.
[11]. Mehta M, Rissanen J, Agrawal R. MDL-based decision tree pruning. KDD. 1995 August 20;21(2):216–221.
[12]. Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu and Philip S. Yu, et al., “Top 10 algorithms in data mining”, Knowledge and Information Systems, Vol. 14, No. 1, 1-37, DOI: 10.1007/s10115-007- 0114-2.
[13]. Hang Yang, Fong, S, “Optimized very fast decision tree with balanced classification accuracy and compact tree size,” Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on, Pp.57-64, 24-26 Oct. 2011.
[14]. Kamiński, B.; Jakubczyk, M.; Szufel, P. (2017). “A framework for sensitivity analysis of decision trees”. Central European Journal of Operations Research. 26 (1): 135–159. - doi:10.1007/s10100-017-0479-6. PMC 5767274. PMID 29375266.
[15]. ^ Quinlan, J. R. (1987). “Simplifying decision trees”. International Journal of Man-Machine Studies. 27 (3): 221–234. CiteSeerX 10.1.1.18.4267. doi:10.1016/S0020-7373(87)80053-6.

[16]. Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.
[17]. ^ Jump up to:a b c d Ho TK (1998). “The Random Subspace Method for Constructing Decision Forests” (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844. - doi:10.1109/34.709601.

In[111]:=
```
WordCloud[{"THANK", "YOU"},
 PreprocessingRules → s_String :→ First[SpellingCorrectionList[s]]]
```

Out[111]=