Tutorial 2

CPSC 340: Machine Learning and Data Mining

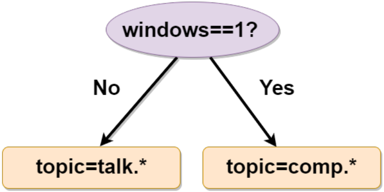
Fall 2017

Overview

1. Decision Tree
   - Decision Stump
   - Decision Tree
2. Training, Testing, and Validation Set

Decision Stump

- Decision stump: simple decision tree with 1 splitting rule based on 1 feature.
- Binary example:



- Assigns a label to each leaf based on the most frequent label.
- How to find the best splitting rule?
  - score the rules
  - intuitive score: classification accuracy.

The Dataset

- X: A matrix
  - each row corresponds to a city
    - first column corresponds to longitude
    - second column corresponds to latitude
- y: class label
  - 1 for blue states, 2 for red states.
- Given new city Xtest
  - predict label ytest

exampledecisionStump.jl

example_decisionStump.jl

```
# Load X and y variable
using JLD
X = load("citiesSmall.jld","X")
y = load("citiesSmall.jld","y")

# Compute number of objects and number of features
(n,d) = size(X)

###############################
### Majority Predictor Model ###
###############################

# Fit majority predictor and compute error
include("majorityPredictor.jl")
model = majorityPredictor(X,y)

# Evaluate training error
yhat = model.predict(X)
trainError = sum(yhat .!= y)/n
@printf("Error with majority predictor: %.2f\n",trainError);

###############################
### Decision Stump Moodel ######
###############################

# Fit decision stump classifier that uses equalities
include("decisionStump.jl")
model = decisionStumpEquality(X,y)

# Evaluate training error
yhat = model.predict(X)
trainError = sum(yhat .!= y)/n
@printf("Error with equality-rule decision stump: %.2f\n",trainError);

# Plot classifier
include("plot2Dclassifier.jl")
plot2Dclassifier(X,y,model)
```

decisionStump.jl

decisionStump.jl

```julia
include("misc.jl") # Includes "mode" function

# First let's define a "type",
# specifying the functions/varaibles we want the stump to have
type StumpModel
    predict # Function that makes predictions
    split # Function that splits data
    baseSplit # Set this to one stump doesn't split
end

function decisionStumpEquality(X,y)
    # Fits a decision stump based on equality after rounding to nearest integer

    # Get the size of the data matrix
    (n,d) = size(X)

    # Round all the data to the nearest integer
    X = round.(X)

    # Initial the best rule with the baseline rule (no split)
    y_mode = mode(y)
    minError = sum(y .!= y_mode);
    splitVariable = [];
    splitValue = [];
    splitYes = y_mode;
    splitNo = [];

    # Search for the best rule
    # (Uses O(n^2d) approach to keep code simple)
    yhat = zeros(n)
    for j in 1:d
        # Try unique values of column as split values
        for val in unique(X[:,j])

            # Test whether each object satisfies equality
            yes = X[:,j] .== val

            # Find correct label on both sides of split
            y_yes = mode(y[yes])
            y_no = mode(y[.!yes])

            # Make predictions
            yhat[yes] = y_yes
            yhat[.!yes] = y_no

            # Compute error
            trainError = sum(yhat .!= y)

            # Update best rule
            if trainError < minError
                minError = trainError
                splitVariable = j
                splitValue = val
                splitYes = y_yes
                splitNo = y_no
            end
        end
    end

    # Now that we have the best rule,
    # let's build our splitting function
    function split(Xhat)
        (t,d) = size(Xhat)
        Xhat = round.(Xhat)
        if isempty(splitVariable)
            return fill(true,t)
        else
            return (Xhat[:,splitVariable] .== splitValue)
        end
    end

    # Now that we have the best rule,
    # let's build our predict function
    function predict(Xhat)
        (t,d) = size(Xhat)
        yes = split(Xhat)
        yhat = fill(splitNo,t)
        yhat[yes] = splitYes
        return yhat
    end

    return StumpModel(predict,split,isempty(splitNo))
end
```

Decision Tree

- Decision stumps have only 1 rule based on only 1 feature.
  - Very limited class of models: usually not very accurate for most tasks.
- Decision trees allow sequences of splits based on multiple features.
  - Very general class of models: can get very high accuracy.
  - However, it's computationally infeasible to find the best decision tree.
- Most common decision tree learning algorithm in practice:
  - Greedy recursive splitting.

exampleDecisionTree.jl and decisionTree.jl

example_decisionTree.jl

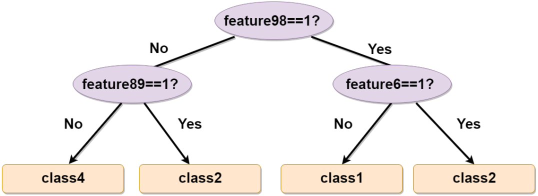
```julia
# Load X and y variable
using JLD
X = load("citiesSmall.jld","X")
y = load("citiesSmall.jld","y")
n = size(X,1)

# Fit a decision tree and compute error
include("decisionTree.jl")
depth = 2
model = decisionTree(X,y,depth)

# Evaluate training error
yhat = model.predict(X)
trainError = sum(yhat .!= y)/n
@printf("Error with depth-%d decision tree: %.3f\n",depth,trainError)

# Plot classifier
include("plot2Dclassifier.jl")
plot2Dclassifier(X,y,model)
```

decisionTree.jl

```julia
include("decisionStump.jl")

function decisionTree(X,y,depth)
    # Fits a decision tree using greedy recursive splitting
    # (recursion to make the code simpler)

    (n,d) = size(X)

    # Learn a decision stump
    splitModel = decisionStump(X,y)

    if depth <= 1 || splitModel.baseSplit
        # Base cases where we stop splitting:
        # - this stump gets us to the max depth
        # - this stump doesn't split the data
        return splitModel
    else
        # Use the decision stump to split the data
        yes = splitModel.split(X)

        # Recusively fit a decision tree to each split
        yesModel = decisionTree(X[yes,:],y[yes],depth-1)
        noModel = decisionTree(X[.!yes,:],y[.!yes],depth-1)

        # Make a predict function
        function predict(Xhat)
            (t,d) = size(Xhat)
            yhat = zeros(t)

            yes = splitModel.split(Xhat)

            yhat[yes] = yesModel.predict(Xhat[yes,:])
            yhat[.!yes] = noModel.predict(Xhat[.!yes,:])
            return yhat
        end

        return GenericModel(predict)
    end
end
```

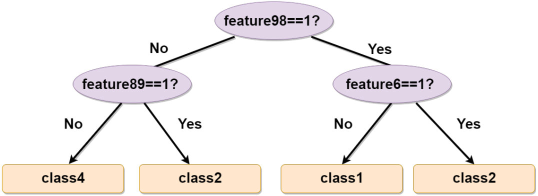
Rewriting a decision tree using if/else statements

- Decision tree:



Rewriting a decision tree using if/else statements

- Decision tree:



- If-else statement:

```
if X(i,98) ==1
    if X(i,6)==1
        return 2
    else
        return 1
    end
else
    if X(i,89)==1
        return 2
    else
        return 4
    end
end
```

Training, Testing, and Validation Set

- Given training data, we would like to learn a model to minimize error on the testing data
- How do we decide decision tree depth?
- We care about test error.
- But we can't look at test data.
- So what do we do?????

Training, Testing, and Validation Set

- Given training data, we would like to learn a model to minimize error on the testing data
- How do we decide decision tree depth?
- We care about test error.
- But we can't look at test data.
- So what do we do?????
- One answer: Use part of your train data to approximate test error.
- Split training objects into training set and validation set:
  - Train model on the training data.
  - Test model on the validation data.

Cross-Validation

- Isn't it wasteful to only use part of your data?
- k-fold cross-validation:
  - Train on k-1 folds of the data, validate on the other fold.
  - Repeat this k times with different splits, and average the score.

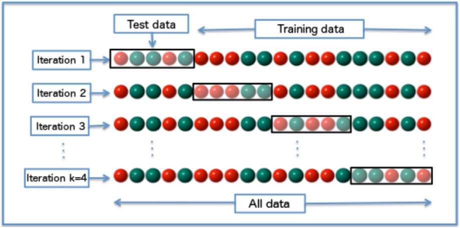


Figure 1: Adapted from Wikipedia.

- Note: if examples are ordered, split should be random.

Problem: 2-Fold Cross Validation

- Modify the code below to compute the 2-fold cross-validation scores on the training data alone.
- Find the depth that would be chosen by cross-validation.

```
% Load X and y variable
load newsgroups.mat
[N,D] = size(X);
T = length(ytest);
depth = 5;
model = decisionTree(X,y,depth);
yhat = model.predictFunc(model,X);
errorTrain = sum(yhat ~= y)/N;
yhat =model.predictFunc(model,Xtest);
errorTest = sum(yhat ~= ytest)/T;
```

Solution: 2-Fold Cross Validation

```
% Load X and y variable
load newsgroups.mat
[N,D] = size(X);
Xtest = X (floor(N/2) + 1 :N  , : );
ytest= y (floor(N/2) +1 : N) ;
X =  X ( 1:floor(N/2) , : )  ;
y = y (1: floor(N/2));
mindepth = -1 ; minError = Inf;
for depth =1 :15
    errorTrain = 0; errorTest = 0;
    for i =1:2
        [N,D] = size(X);
        T = length(ytest);
        model = decisionTree(X,y,depth);
        yhat = model.predictFunc(model,X);
        errorTrain = errorTrain +sum(yhat ~= y)/N;
        yhat = model.predictFunc(model,Xtest);
        errorTest = errorTest + sum(yhat ~= ytest)/T;
        [X, Xtest]=mySwap(Xtest, X);
        [y,ytest] = mySwap(ytest,y) ;
    end
    disp(errorTest/2 ) ;
    if errorTest/2 < minError
        minError= errorTest/2;
        mindepth = depth;
    end
end
disp(minError); disp(mindepth);
```