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:
  
  ![Decision Tree Diagram](image)

  - 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**
```julia
# 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 Model ###

# 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)
```
- **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.
```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
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)

        # Recursively 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)
            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:

```
feature98==1?
  No -> feature89==1?
    No -> class4
    Yes -> class2
  Yes -> feature6==1?
    No -> class1
    Yes -> class2
```
Rewriting a decision tree using if/else statements

- Decision tree:

- If-else statement:

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

```matlab
% 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

```matlab
% 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);
```