Data Science(Classification)

Presented By:
Sri Krishnamurthy, CFA, CAP
www.QuantUniversity.com
sri@quantuniversity.com

Classification Trees

- ✓ Introduction
- ✓ Classification error rate
- ✓ Example study using R
- ✓ Example study using python

Classification Trees

- A classification tree is used to predict a qualitative response
- For classifying examples, all of the features are assumed to have finite discrete domains
- Each element of the domain of the classification is called a **class**
- Algorithms for classification trees usually work top-down, by selecting a variable at each step that best splits the set of items

Example study using R

- Data frame Carseats in ISLR package is a simulated data set containing sales of child car seats at 400 different stores
- It contains 400 observations on the following 11 variables
- Variables includes unit sales at each location, price charged by competitor, community income level, local advertising budget, population size in region, price company charges for car seats, the quality of the shelving location for the car seats, average age of the local population, education level, whether the store is an urban location and whether the store is in the US

Example study using R

- Transform Sales into a binary variable High (Yes if Sales is greater than 8, otherwise, No)

```
library (ISLR)
attach (Carseats)
#Transform Sales variable to a binary variable
High=ifelse (Sales >8, "Yes ", " No ")
Carseats =data.frame(Carseats ,High)
head(Carseats)
  Sales CompPrice Income Advertising Population Price ShelveLoc Age Education Urban  US High
1  9.50       138     73          11        276   120       Bad  42        17   Yes Yes Yes
2 11.22       111     48          16        260    83      Good  65        10   Yes Yes Yes
3 10.06       113     35          10        269    80    Medium  59        12   Yes Yes Yes
4  7.40       117    100           4        466    97    Medium  55        14   Yes Yes  No
5  4.15       141     64           3        340   128       Bad  38        13   Yes  No  No
6 10.81       124    113          13        501    72       Bad  78        16    No Yes Yes
```

See ***ClassificationTrees.R***

Example study using R

- To predict High variable, use the tree() function to fit a classification tree

```
> #Use all other variables except Sales to fit a classification tree
> library(tree)
> tree = tree(High ~ . - Sales, Carseats)
> summary(tree)

Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"  "Advertising"
[7] "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

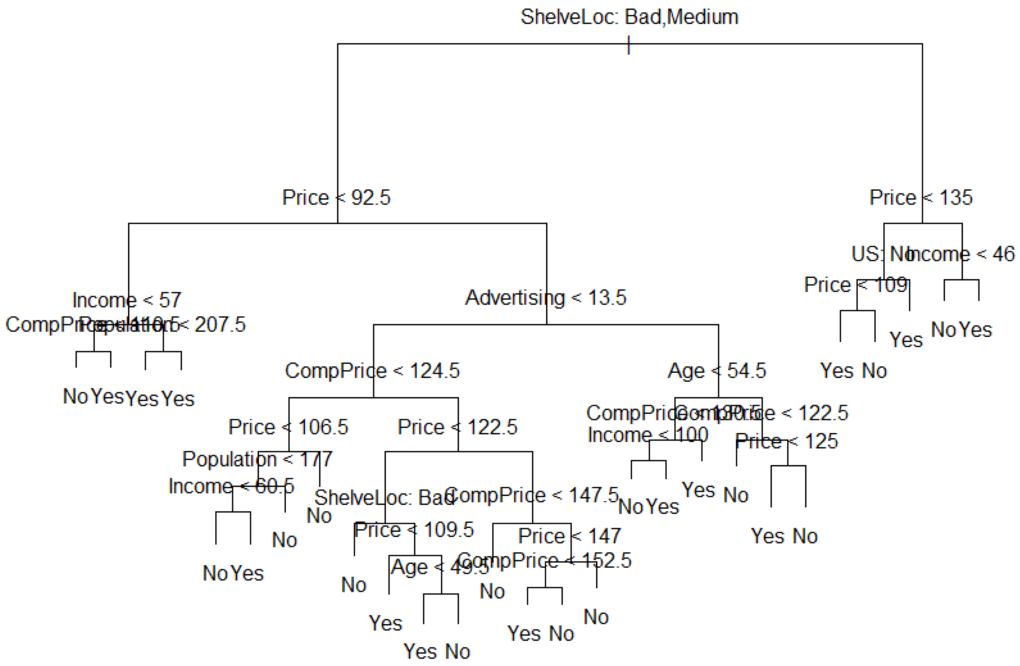
- The error rate is 9%

See ***ClassificationTrees.R***

Example study using R

- Use plot() function to display the tree structure
- Use text() function to display the node labels

```
#Display the tree structure and node labels
plot(tree)
text(tree, pretty =0) #Pretty=0 includes the category names
```



See ***ClassificationTrees.R***

Example study using R

- predict() function can be used for evaluating the performance of a classification tree

```
#Split the dataset into a training set and a test set
set.seed (2)
train = sample (1: nrow(Carseats), 200)
Carseats.test = Carseats [-train,]
High.test = High[-train]
#Build the tree based on the training set
tree.train = tree(High ~ . - Sales, Carseats, subset = train)
#Evaluate its performance on the test data
tree.pred = predict(tree.train, Carseats.test, type = "class")
table(tree.pred, High.test)

          High.test
tree.pred  No  Yes
      No   86   27
      Yes  30   57
```

- The error rate is (27+30)/200=28.5%

See ***ClassificationTrees.R***

Example study using R

- To determine the optimal level of tree complexity, use cv.tree() performs cross-validation

```
#Determine the optimal level
set.seed (3)
#FUN = prune.misclass indicate that classification error rate is used to
#guide the cross-validation and pruning process
cv.carseats = cv.tree(tree, FUN = prune.misclass)
names(cv.carseats)
cv.carseats
> names(cv.carseats)
[1] "size"   "dev"    "k"      "method"
> cv.carseats
$size
 [1] 27 26 24 22 19 17 14 12  7  6  5  3  2  1

$dev
 [1] 107 105 109 109 109 103 103 106 116 118 116 117 119 165

$k
 [1]      -Inf  0.000000  0.500000  1.000000  1.333333  1.500000  1.666667  2.500000  3.800000
[10]  4.000000  5.000000  7.500000 18.000000 47.000000

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

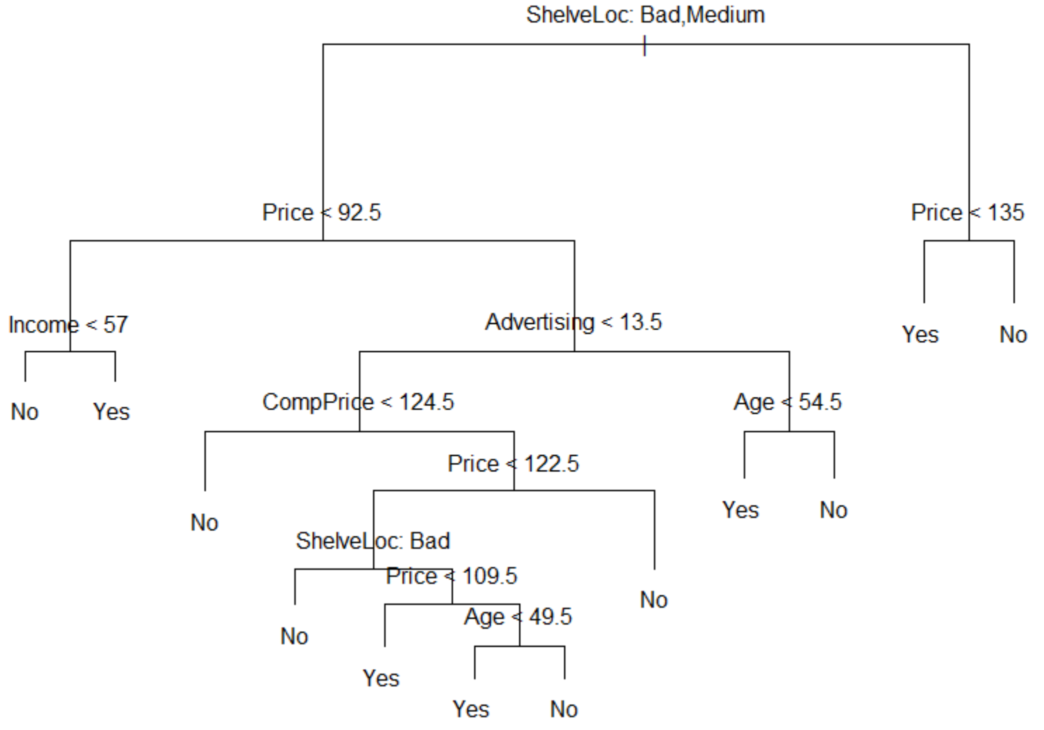
size: number of terminal nodes of each tree considered
k: the value of the cost-complexity parameter used
dev: cross-validation error rate

See ***ClassificationTrees.R***

Example study using R

- According to the cross-validation error rate dev, tree with 9 terminal nodes results in the lowest error rate
- Use prune.misclass() function in order to prune the tree

```
#Prune the tree
prune.carseats = prune.misclass(tree, best =9)
plot(prune.carseats)
text(prune.carseats, pretty =0)
```



See ***ClassificationTrees.R***

QuantUniversity, LLC

Example study using R

- According to the cross-validation error rate dev, tree with 9 terminal nodes results in the lowest error rate
- Use prune.misclass() function in order to prune the tree

```
> #The pruned tree performance
> prune.pred = predict(prune.carseats, Carseats.test, type = "class")
> table(prune.pred, High.test)
          High.test
prune.pred  No  Yes
       No  101   10
       Yes  15   74
```

- The error rate is (15+10)/200=15%

See ***ClassificationTrees.R***

Example study using python

- Export Carseats dataset to an Excel spreadsheet

```
#Export dataset
data(Carseats, package="ISLR")
write.csv(Carseats,"Carseats.csv")
```

- Import data in python

In [1]:
```
#Import Carseats from csv file
import pandas as pd
import numpy as np
carseats = pd.read_csv("Carseats.csv", header=0)
del carseats['Unnamed: 0']
carseats.head(10)
```

Out[1]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes
1	11.22	111	48	16	260	83	Good	65	10	Yes	Yes
2	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
3	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes
4	4.15	141	64	3	340	128	Bad	38	13	Yes	No
5	10.81	124	113	13	501	72	Bad	78	16	No	Yes
6	6.63	115	105	0	45	108	Medium	71	15	Yes	No
7	11.85	136	81	15	425	120	Good	67	10	Yes	Yes
8	6.54	132	110	0	108	124	Medium	76	10	No	No
9	4.69	132	113	0	131	124	Medium	76	17	No	Yes

See ***ClassificationTrees.ipynb***

Example study using python

- Transform Sales into a binary variable High

In [2]:
```
#Transform Sales into a binary factor called High
carseats['High'] = (carseats.Sales > 8).astype(bool)
carseats.head(10)
```

Out[2]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US	High
0	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes	True
1	11.22	111	48	16	260	83	Good	65	10	Yes	Yes	True
2	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes	True
3	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes	False
4	4.15	141	64	3	340	128	Bad	38	13	Yes	No	False
5	10.81	124	113	13	501	72	Bad	78	16	No	Yes	True
6	6.63	115	105	0	45	108	Medium	71	15	Yes	No	False
7	11.85	136	81	15	425	120	Good	67	10	Yes	Yes	True
8	6.54	132	110	0	108	124	Medium	76	10	No	No	False
9	4.69	132	113	0	131	124	Medium	76	17	No	Yes	False

See ***ClassificationTrees.ipynb***

Example study using python

- Prepare the data

In [3]:
```
carseats['Urban'] = (carseats.Urban == "Yes").astype(int)
carseats['US'] = (carseats.US == "Yes").astype(int)
carseats['ShelveLoc_'] = 0
carseats['ShelveLoc_'][carseats['ShelveLoc'] =="Good"] = 1
carseats['ShelveLoc_'][carseats['ShelveLoc'] =="Medium"] = 2
carseats['ShelveLoc_'][carseats['ShelveLoc'] =="Bad"] = 3
carseats.head(10)
```

Out[3]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US	High	ShelveLoc_
0	9.50	138	73	11	276	120	Bad	42	17	1	1	True	3
1	11.22	111	48	16	260	83	Good	65	10	1	1	True	1
2	10.06	113	35	10	269	80	Medium	59	12	1	1	True	2
3	7.40	117	100	4	466	97	Medium	55	14	1	1	False	2
4	4.15	141	64	3	340	128	Bad	38	13	1	0	False	3
5	10.81	124	113	13	501	72	Bad	78	16	0	1	True	3
6	6.63	115	105	0	45	108	Medium	71	15	1	0	False	2
7	11.85	136	81	15	425	120	Good	67	10	1	1	True	1
8	6.54	132	110	0	108	124	Medium	76	10	0	0	False	2
9	4.69	132	113	0	131	124	Medium	76	17	0	1	False	2

See ***ClassificationTrees.ipynb***

Example study using python

- Fit a classification tree using DecisionTreeClassifier() in scikit-learn

```
In [4]: #Select response y and training set X
        y = carseats['High']
        cols_to_keep = ['CompPrice', 'Income', 'Advertising', 'Population', 'Price', 'ShelveLoc_',
                        'Age','Education', 'Urban']
        X = carseats[cols_to_keep]
```

```
In [7]: from sklearn import tree
        cltree = tree.DecisionTreeClassifier()
        cltree = cltree.fit(X, y)
```

- Accuracy of this model

```
In [7]: #Evaluate the model
        from sklearn.cross_validation import cross_val_score
        scores = cross_val_score(cltree, X, y, cv=10)
        print scores.mean()

        0.737307692308
```

- Export the tree

```
In [11]: #Export the tree in Graphviz format using the export_graphviz
         dotfile = open("E:/QuantUniversity/tree.dot", 'w')
         tree.export_graphviz(cltree, out_file = dotfile, feature_names = X.columns)
         dotfile.close()
```

See ***ClassificationTrees.ipynb***

KNN

- What is KNN
- KNN in R
- KNN in Python

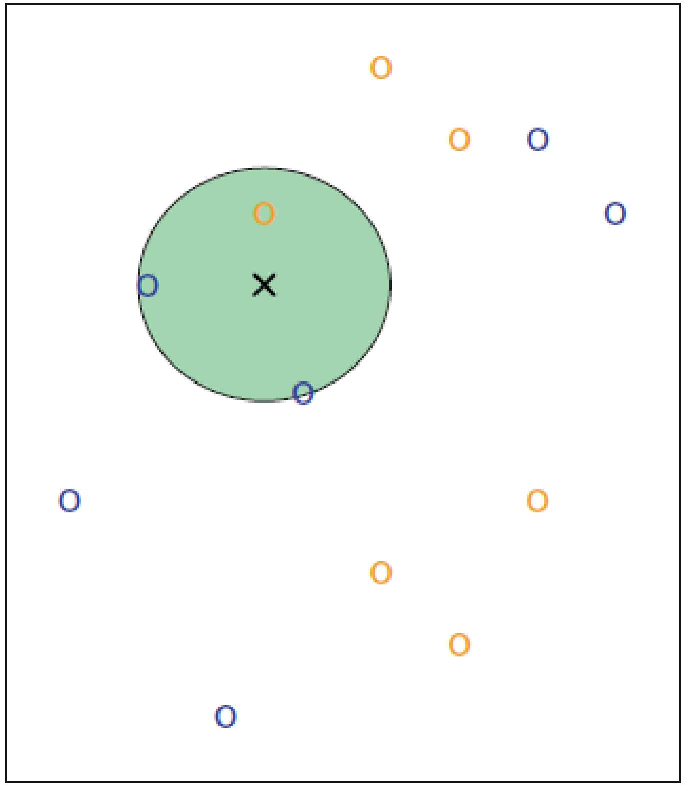
What is KNN

- KNN means K-Nearest Neighbors, it is a very simple algorithm used for classification and regression.
- The KNN classifier first identifies the neighbors K points in the training data that are closest to a test observation x_0, amongst N_0points. It then estimates the conditional probability for Y= *j*, Finally, KNN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability.

$$\Pr(Y = j|X = x_0) = N_0\frac{1}{K}\sum_{i\in N_0} I(y_i = j)$$

QuantUniversity, LLC

What is KNN



Example:

- K=3 (three nearest neighbors of x)
- Class “orange o” and “blue o”.
- Find the class for x

Result :

- Probabilities of 2/3 for the blue class
- Probabilities of 1/3 for the orange class

QuantUniversity, LLC

KNN in R

- “Iris” dataset in R, it has 150 observations and 5 variables. Each observation represent a flower, and there are 5 features which recorded on those flowers. These 150 observations are collected from 3 different species. Based on knowing the first 4 features

(Sepal.length, Sepal.width, Patel.length, Patel,width) we can classifier a new observation into one of these 3 species.

```
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1
```

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

See ***KNN.R***

KNN in R

- Mix up data
- Since it is a very nice organized dataset, we need to mix the entire dataset up. To do this, we have to create a 150 random number dataset and make the original dataset re-organized follow the order of the random number dataset.

```
> group<- runif(nrow(iris))
> group
  [1] 0.9882166183 0.5940190239 0.1337920092 0.1093714633
156312
  [8] 0.6883584394 0.1238144503 0.1765312739 0.0513233980
005529
 [15] 0.2526203436 0.0743994124 0.0132462864 0.8147595294
368890
 [22] 0.0421418818 0.4482217485 0.9431283800 0.3041717457
944780
 [29] 0.9930689118 0.0179640960 0.6333474671 0.3473129433
161311
 [36] 0.3367951058 0.0606662470 0.8249456959 0.8221009444
111697
```

```
> iris
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
66           6.7         3.1          4.4         1.4 versicolor
118          7.7         3.8          6.7         2.2  virginica
17           5.4         3.9          1.3         0.4     setosa
30           4.7         3.2          1.6         0.2     setosa
114          5.7         2.5          5.0         2.0  virginica
75           6.4         2.9          4.3         1.3 versicolor
150          5.9         3.0          5.1         1.8  virginica
22           5.1         3.7          1.5         0.4     setosa
45           5.1         3.8          1.9         0.4     setosa
115          5.8         2.8          5.1         2.4  virginica
11           5.4         3.7          1.5         0.2     setosa
```

See ***KNN.R***

QuantUniversity, LLC

KNN in R

- Re-scale data (Normalization)
- Normalization means adjusting values measured on different scales to a notionally common scale. In this case we adjust the range of first 4 features from 0 to 1 to minimize undue influence from the feature which has a larger range.

```
#Normalization#
normalize<-function(x)(return((x-min(x))/(max(x)-min(x))))
iris_n<-as.data.frame(lapply(iris[,c(1,2,3,4)],normalize))
summary(iris_n)
```

```
> summary(iris_n)
  Sepal.Length     Sepal.Width      Petal.Length     Petal.Width
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
 1st Qu.:0.2222   1st Qu.:0.3333   1st Qu.:0.1017   1st Qu.:0.08333
 Median :0.4167   Median :0.4167   Median :0.5678   Median :0.50000
 Mean   :0.4287   Mean   :0.4406   Mean   :0.4675   Mean   :0.45806
 3rd Qu.:0.5833   3rd Qu.:0.5417   3rd Qu.:0.6949   3rd Qu.:0.70833
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
```

See **_KNN.R_**

KNN in R

- Training dataset & testing dataset
- Training dataset is what we used to learned the pattern, namely a KNN model. The test dataset is going to serve a way to test how well a model predict. In this case we hold 20 observations for testing. We set the species feature as our training target as well as testing target. Separate

```
#Separate training&testing dataset
iris_train<-iris_n[1:129, ]
iris_test<-iris_n[130:150, ]
iris_train_target<-iris_n[1:129, 5]
iris_test_target<-iris_n[130:150, 5]
```

See ***KNN.R***

QuantUniversity, LLC

KNN in R

- Fit in KNN model
- We use knn() function to do the training, we need to fit in the training data frame, the test data frame and the training target variables. before that we have to choose a k value, as k is a place holder for how many nearest neighbor that you want the algorithm to use. The rule of thumb is to take the square root of the total number of observations you have, in this case should be 13.

```
# fit in knn algorithm
require(class)
m1<- knn(train= iris_train, test= iris_test, cl=iris_train_target, k=13)
```

See ***KNN.R***

QuantUniversity, LLC

KNN in R

- Results & comparison
- The predict results are present in a table.

```
> m1
 [1] versicolor virginica  versicolor virginica  versicolor versicolor versicolor
 [8] versicolor versicolor setosa     versicolor setosa     virginica  virginica
[15] versicolor setosa     setosa     versicolor setosa     versicolor virginica
Levels: setosa versicolor virginica
```

- To see how well the model predicted, we use table() to see the difference between predict result and what species they actually are.

```
> table(iris_test_target,m1)
                m1
iris_test_target setosa versicolor virginica
      setosa          5          0         0
      versicolor      0         10         1
      virginica       0          1         4
```

See **_KNN.R_**

QuantUniversity, LLC

KNN in Python

- Create dataset
- First, create a dataset with four observations, each observation has a label. Our goal is to train a classifier to classify a new observation into one of these labels.

```
def createDataSet():
    characters=array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
    labels=['A','A','B','B']
    return characters,labels
```

See ***KNN.ipynb***

KNN in Python

- Train the classifier
- Train a classifier to count the distance between the sample and all the observations in training dataset. Choose k nearest observations then count their labels. The sample's label will be the same as the most label counted in the k nearest observations.

```
def classify(sample,dataSet,labels,k):
    dataSetSize=dataSet.shape[0]
    diffMat=tile(sample,(dataSetSize,1))-dataSet
    sqDiffMat=diffMat**2
    sqDistances=sqDiffMat.sum(axis=1)
    distances=sqDistances**0.5
    sortedDistIndicies=distances.argsort()

    classCount={}
    for i in range(k):
        voteIlabel=labels[sortedDistIndicies[i]]
        classCount[voteIlabel]=classCount.get(voteIlabel,0)+1

        sortedClassCount=sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    return sortedClassCount[0][0]
```

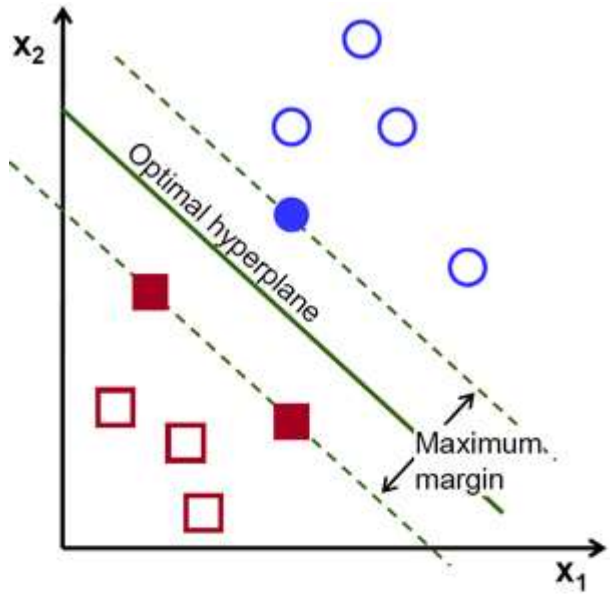
See ***KNN.ipynb***

Support Vector Machines

- ✓ Support vector machine
- ✓ Support vector classifier
- ✓ Example study using R
- ✓ Example study using python

Support vector machine

- A **Support Vector Machine** (**SVM**) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.



Ref: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

QuantUniversity, LLC

Example study using R

- Data frame Khan in ILSR package consists of a number of tissue samples corresponding to four distinct types of small round blue cell tumors
- For each tissue sample, 2308 gene expression measurements are available
- The format is a list containing four components: xtrain, xtest, ytrain, and ytest

Example study using R

- Examine the dimension of the data

```
> library (ISLR)
> names(Khan)
[1] "xtrain" "xtest"  "ytrain" "ytest"
> dim(Khan$xtrain)
[1]   63 2308
> dim(Khan$xtest)
[1]   20 2308
> length(Khan$ytrain)
[1] 63
> length(Khan$ytest)
[1] 20
```

- The training and test sets consist of 63 and 20 observations respectively

See **_SVM.R_**

QuantUniversity, LLC

Example study using R

- Use a support vector approach to predict cancer subtype using gene expression measurements
- Use svm() function in e1071 package

kernel: the kernel used in training and predicting
cost: cost of constraints violation

```
> #Predict cancer subtype
> library (e1071)
> data = data.frame( x = Khan$xtrain, y = as.factor(Khan$ytrain))
> svm = svm(y ~., data=data, kernel = "linear", cost = 10)
> summary(svm)

Call:
svm(formula = y ~ ., data = data, kernel = "linear", cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.0004332756

Number of Support Vectors:  58

 ( 20 20 11 7 )


Number of Classes:  4

Levels:
 1 2 3 4
```

See ***SVM.R***

Example study using R

```
> table(svm$fitted, data$y)

     1  2  3  4
  1  8  0  0  0
  2  0 23  0  0
  3  0  0 12  0
  4  0  0  0 20
```

- There is no training error
- Check support vector classifier's performance on the test observations

```
> #Support vector classifier's performance on the test observations
> data.test = data.frame(x = Khan$xtest, y = as.factor(Khan$ytest))
> pred = predict(svm, newdata = data.test)
> table(pred, data.test$y)

pred 1 2 3 4
   1 3 0 0 0
   2 0 6 2 0
   3 0 0 4 0
   4 0 0 0 5
```

See ***SVM.R***

Example study using python

- Import data

```
In [1]: #Import xtrain from csv file
        import pandas as pd
        import numpy as np
        xtrain = pd.read_csv("Khan_xtrain.csv", header = None)
        del xtrain[0]
        xtrain = xtrain[1:]

In [2]: #Import ytrain from csv file
        ytrain = pd.read_csv("Khan_ytrain.csv", header = None)
        del ytrain[0]
        ytrain = ytrain[1:]

In [4]: #Import xtest from csv file
        xtest = pd.read_csv("Khan_xtest.csv", header = None)
        del xtest[0]
        xtest = xtest[1:]

In [5]: #Import ytest from csv file
        ytest = pd.read_csv("Khan_ytest.csv", header = None)
        del ytest[0]
        ytest = ytest[1:]
```

See ***SVM.ipynb***

Example study using python

- Predict cancer subtype on train dataset
- Fit a SVM using svm() in scikit-learn

```
In [9]: from sklearn import svm
        X = xtrain
        y = ytrain
        clf = svm.SVC(kernel='linear')
        clf.fit(X, y)

Out[9]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
          kernel='linear', max_iter=-1, probability=False, random_state=None,
          shrinking=True, tol=0.001, verbose=False)
```

See ***SVM.ipynb***

Example study using python

- Use the fitted SVM to predict test dataset
- Check the performance on the test observations

In [13]:
```
#Make predictions on xtest dataset
pred = clf.predict(xtest)
pred
```

Out[13]:
```
array(['3', '2', '4', '2', '1', '3', '4', '2', '3', '1', '3', '4', '1',
       '2', '2', '2', '4', '2', '4', '2'], dtype=object)
```

In [18]:
```
#Performance on test dataset
pd.crosstab(pred,ytest[1],rownames=['pred'], colnames=['ytest'])
```

Out[18]:

ytest	1	2	3	4
pred				
1	3	0	0	0
2	0	6	2	0
3	0	0	4	0
4	0	0	0	5

See ***SVM.ipynb***

Neural Network Classification

- ✓ Neural network classification in R
- ✓ Neural network classification in python

Example study using R

- Data frame **infert** is a matched case-control study dating from before the availability of conditional logistic regression
- It contains 248 observations on the following 8 variables
- The variables include education, age, parity, number of prior induced abortions, case status, number of prior spontaneous abortions, matched set number, stratum number

Example study using R

- Fit the network classifying case variable using age, parity, induced, spontaneous
- The network has 3 hidden layers

```
attach(infert)
library(neuralnet)
neuralnet <- neuralnet(case ~ age + parity + induced + spontaneous, data=infert,
                       hidden=3, err.fct="ce", linear.output=FALSE)
neuralnet$result.matrix
plot(neuralnet)
```

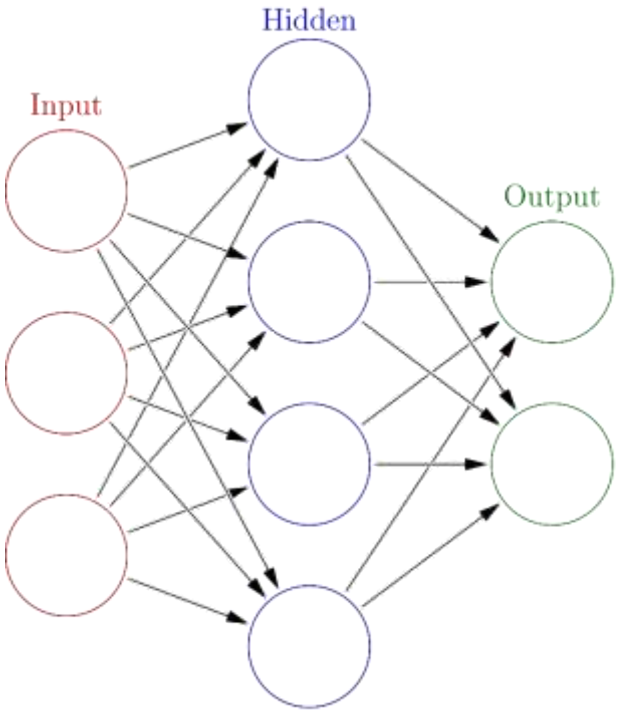
```
> neuralnet$result.matrix
                                        1
error                       113.73879260616
reached.threshold             0.00887480898
steps                     64953.00000000000
Intercept.to.1layhid1         1.70749607309
age.to.1layhid1              23.17106882284
parity.to.1layhid1           11.96279868860
induced.to.1layhid1        -131.17641312579
spontaneous.to.1layhid1    -296.22131305983
Intercept.to.1layhid2        66.13485859737
age.to.1layhid2              -1.09159296623
parity.to.1layhid2            8.84059317974
induced.to.1layhid2         -19.40447872304
spontaneous.to.1layhid2     -25.28284597365
Intercept.to.1layhid3       -15.46184848858
age.to.1layhid3               0.58564168262
parity.to.1layhid3          -18.02353826644
induced.to.1layhid3          15.33660841139
spontaneous.to.1layhid3      20.89864632803
Intercept.to.case            20.41400808431
1layhid.1.to.case            -6.50566456720
1layhid.2.to.case           -15.58740034121
1layhid.3.to.case             2.27536828314
```

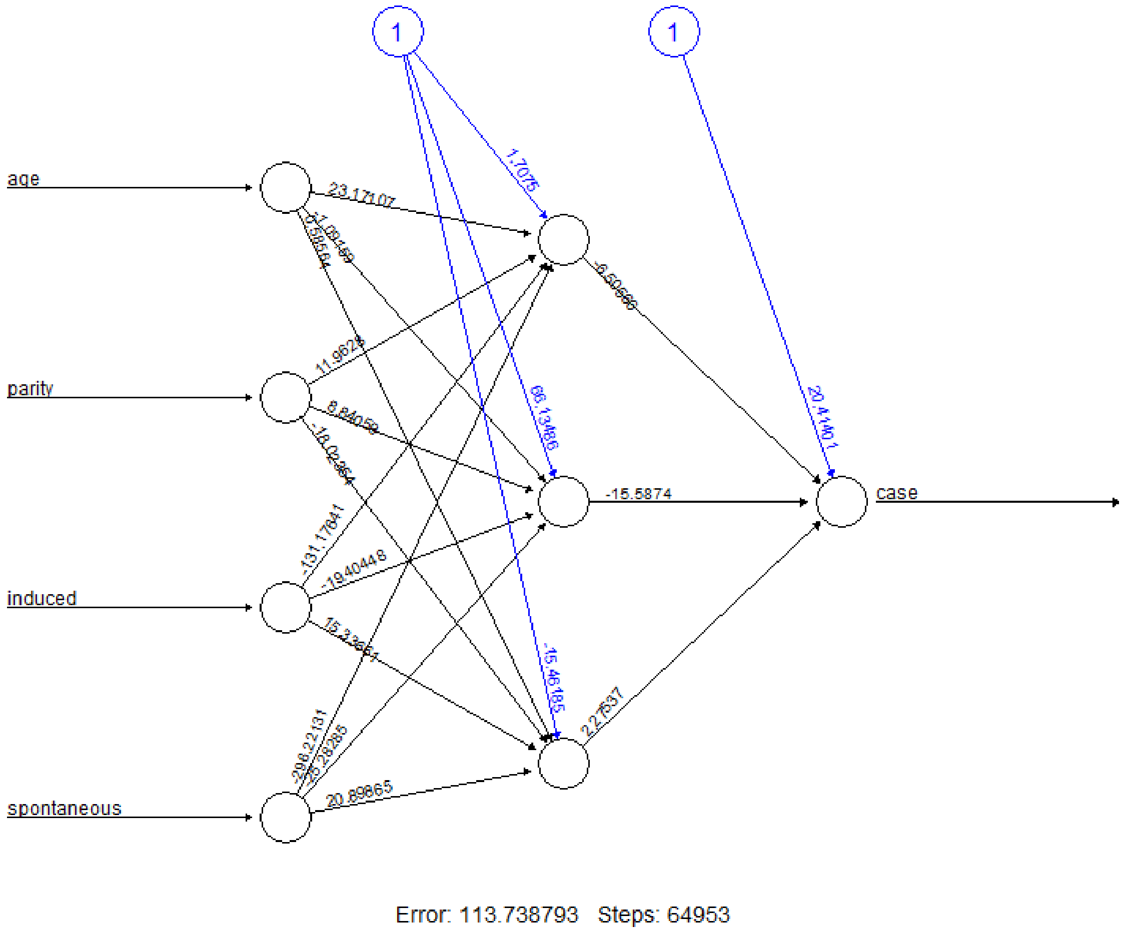
See ***NeuralNet.R***

QuantUniversity, LLC

Example study using R

- Neural network visualization





See ***NeuralNet.R***

Example study using python

- Import data

```
In [1]: #Import xtrain from csv file
        import pandas as pd
        import numpy as np
        infert = pd.read_csv("infert.csv", header = 0)
```

```
In [2]: del infert['Unnamed: 0']
        infert.head(10)
```

Out[2]:

	education	age	parity	induced	case	spontaneous	stratum	pooled.stratum
0	0-5yrs	26	6	1	1	2	1	3
1	0-5yrs	42	1	1	1	0	2	1
2	0-5yrs	39	6	2	1	0	3	4
3	0-5yrs	34	4	2	1	0	4	2
4	6-11yrs	35	3	1	1	1	5	32
5	6-11yrs	36	4	2	1	1	6	36
6	6-11yrs	23	1	0	1	0	7	6
7	6-11yrs	32	2	0	1	0	8	22
8	6-11yrs	21	1	0	1	1	9	5
9	6-11yrs	28	2	0	1	0	10	19

See ***NeuralNet.ipynb***

Example study using python

- Create a network and load the data into the network

```
In [28]: columns = ['age','parity','induced','spontaneous']
         X = infert[columns]
         y = infert['case']
```

```
In [17]: from pybrain.datasets             import ClassificationDataSet
         from pybrain.utilities            import percentError
         from pybrain.tools.shortcuts      import buildNetwork
         from pybrain.supervised.trainers import BackpropTrainer
         from pybrain.structure.modules    import SoftmaxLayer
         from pybrain.tools.xml.networkwriter import NetworkWriter
         from pybrain.tools.xml.networkreader import NetworkReader
```

```
In [30]: ds = ClassificationDataSet(4, 1 , nb_classes=2)
         for k in xrange(len(X)):
             ds.addSample(X.iloc[k],y.iloc[k])
```

```
In [31]: ds._convertToOneOfMany( )
```

See ***NeuralNet.ipynb***

Example study using python

- Build the network and backpropagation trainer

```
In [33]: fnn = buildNetwork( ds.indim, 3 , ds.outdim, outclass=SoftmaxLayer )

In [34]: nn = BackpropTrainer( fnn, dataset=ds, momentum=0.1, verbose=True, weightdecay=0.01)

In [36]: print fnn

FeedForwardNetwork-8
   Modules:
    [<BiasUnit 'bias'>, <LinearLayer 'in'>, <SigmoidLayer 'hidden0'>, <SoftmaxLayer 'out'>]
   Connections:
    [<FullConnection 'FullConnection-4': 'bias' -> 'out'>, <FullConnection 'FullConnection-5': 'bias' -> 'hidden0'>, <FullConne
ction 'FullConnection-6': 'in' -> 'hidden0'>, <FullConnection 'FullConnection-7': 'hidden0' -> 'out'>]
```

See ***NeuralNet.ipynb***

Example study using python

- Evaluate the network

```
In [13]: result = percentError( nn.testOnClassData(), ds['class'] )
         result

Out[13]: 33.46774193548387
```

See ***NeuralNet.ipynb***

Classification

We have covered	Key functionality
Classification tree	✓ Introduction of classification tree and classification error rate ✓ Use tree() function in tree library to construct classification trees in R ✓ Use DecisionTreeClassifier() function in sklearn.tree module to construct classification trees in python
KNN	✓ Introduction of KNN ✓ Use knn() function in class library to construct KNNs in R ✓ How to construct KNNs in python
SVM	✓ Introduction of SVM ✓ Use svm() function in e1071 library to construct SVMs in R ✓ Use svm() function in scikit-learn module to construct SVMs in python
Neural network	✓ Use neuralnet() function in neuralnet library to construct neural networks in R ✓ Use pybrain module to construct neural networks in python

Reference

- Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning. Springer, 2013
- Fritsch S, Günther F: neuralnet: Training of neural networks. R J 2010, 2:30-38
- brms: An R Package for Bayesian Generalized Linear Mixed Models using Stan. Paul-Christian Burkner.
- scikit-learn.org
- pybrain.org

Q&A

QuantUniversity, LLC

www.quantuniversity.com

Thank you!

Contact

Sri Krishnamurthy, CFA, CAP
Founder and CEO
QuantUniversity LLC.

Linkedin srikrishnamurthy

www.QuantUniversity.com