INTRODUCTION

To Data Science

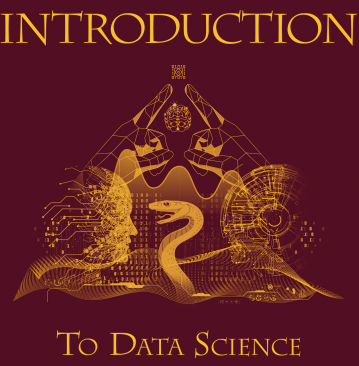
Introduction to Data Science

Local Modeling: KNN and Decision Trees - Class 10

Giora Simchoni

gsimchoni@gmail.com and add #intro2ds in subject

Stat. and OR Department, TAU



K-Nearest Neighbour (KNN)

INTRODUCTION

To Data Science

Global vs local modeling

- So far we have learned two predictive modeling techniques: OLS regression and logistic regression
- Common theme: Global, parametric models (+ probabilistic model for inference) — lots of assumptions!

- A different approach: *Local* modeling: I am similar to my neighbors

Simple example: 1-nearest neighbor

1. Define a distance over the $\mathcal{X}$ space. For $x \in \mathbb{R}^p$ can simply choose the Euclidean distance:

$$d(x, u) = \|x - u\|^2$$

2. For a prediction point (say $x_0 \in Te$), find its nearest neighbor in the Tr

$$i_0 = \arg\min_i d(x_0, x_i)$$

3. Predict x_0 as the response at the nearest neighbor $\hat{y}_0 = y_{i_0}$

K-nearest neighbor (KNN) methods

- Repeat the same steps, but instead of finding the nearest neighbor only, find the k nearest points in Tr to x_0. Assume their indexes are $i_{01}, \ldots, i_{0k}$

- For regression predict the average:

$$\hat{y}_0 = \frac{1}{k} \sum_{j=1}^{k} y_{i_{0j}}$$

- For classification predict the majority:

$$\hat{y}_0 = \begin{cases} 1 & \text{if } \frac{1}{k} \sum_{j=1}^{k} y_{i_{0j}} > 1/2 \\ 0 & \text{otherwise} \end{cases}$$

Reminder: SAHeart Data

```
print(saheart_X.head())
```

```
       sbp  tobacco   ldl  adiposity  typea  obesity  alcohol  age  famhist
index
1      160    12.00  5.73      23.11     49    25.30    97.20   52        0
2      144     0.01  4.41      28.61     55    28.87     2.06   63        1
3      118     0.08  3.48      32.28     52    29.14     3.81   46        0
4      170     7.50  6.41      38.03     51    31.99    24.26   58        0
5      134    13.60  3.50      27.78     60    25.99    57.34   49        0
```

```
print(saheart_y.head())
```

```
index
1    1
2    1
3    0
4    1
5    1
Name: chd, dtype: int64
```

```
SA_Xtr, SA_Xte, SA_Ytr, SA_Yte = train_test_split(saheart_X, saheart_y, test_size=0.2,

print(f'No. of train rows: {SA_Xtr.shape[0]}, no. train of cols: {SA_Xtr.shape[1]}')
print(f'No. of test rows: {SA_Xte.shape[0]}, no. test of cols: {SA_Xte.shape[1]}')
print(f'no. of obs in train y: {SA_Ytr.shape[0]}')
print(f'no. of obs in test y: {SA_Yte.shape[0]}')
```

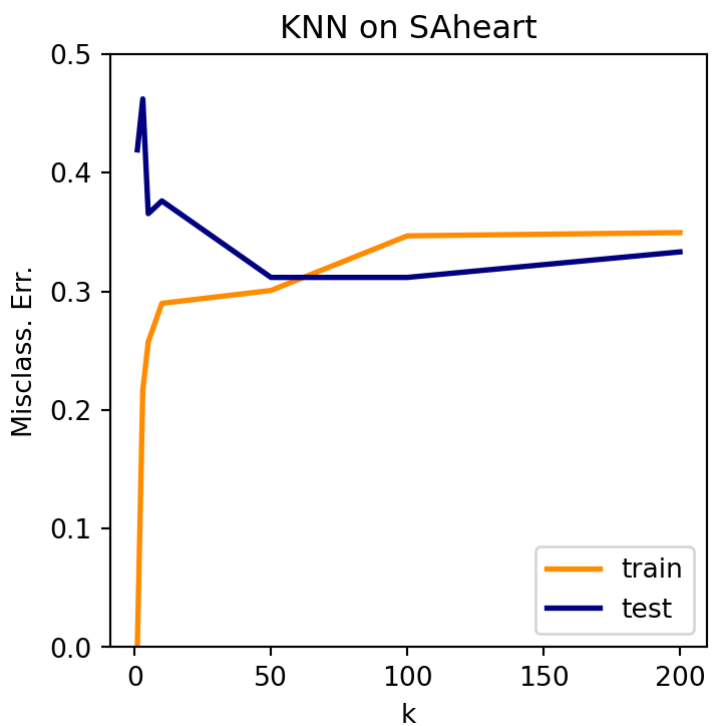
```
No. of train rows: 369, no. train of cols: 9
No. of test rows: 93, no. test of cols: 9
no. of obs in train y: 369
no. of obs in test y: 93
```

KNN for SAHeart (Classification)

```
 1  from sklearn.neighbors import KNeighborsClassifier
 2
 3  ntr = SA_Xtr.shape[0]
 4  nte = SA_Xte.shape[0]
 5  tr_err = []
 6  te_err = []
 7  kvals = [1, 3, 5, 10, 50, 100, 200]
 8
 9  for k in kvals:
10      knn = KNeighborsClassifier(n_neighbors = k)
11      knn.fit(SA_Xtr, SA_Ytr)
12      yhat_tr = knn.predict(SA_Xtr) > 0.5
13      yhat = knn.predict(SA_Xte) > 0.5
14      tr_err.append(np.sum(yhat_tr != SA_Ytr) / ntr)
15      te_err.append(np.sum(yhat != SA_Yte) / nte)
```

Is Euclidean distance suited for this task?

```
plt.figure(figsize=(4, 4))
plt.plot(kvals, tr_err, color='darkorange', lw=2, label='train' )
plt.plot(kvals, te_err, color='navy', lw=2, label='test')
plt.ylim([0.0, 0.5])
plt.xlabel('k')
plt.ylabel('Misclass. Err.')
plt.title('KNN on SAheart')
plt.legend(loc="lower right")
plt.show()
```



Reminder: Netflix Data

```
print(netflix_X.iloc[:5, :3])
```

```
title  Independence Day  The Patriot  The Day After Tomorrow
0                    2            4                       4
1                    4            5                       3
2                    5            5                       5
3                    3            3                       3
4                    4            4                       4
```

```
print(netflix_Y.head())
```

```
0    3
1    4
2    5
3    2
4    5
Name: score, dtype: int64
```

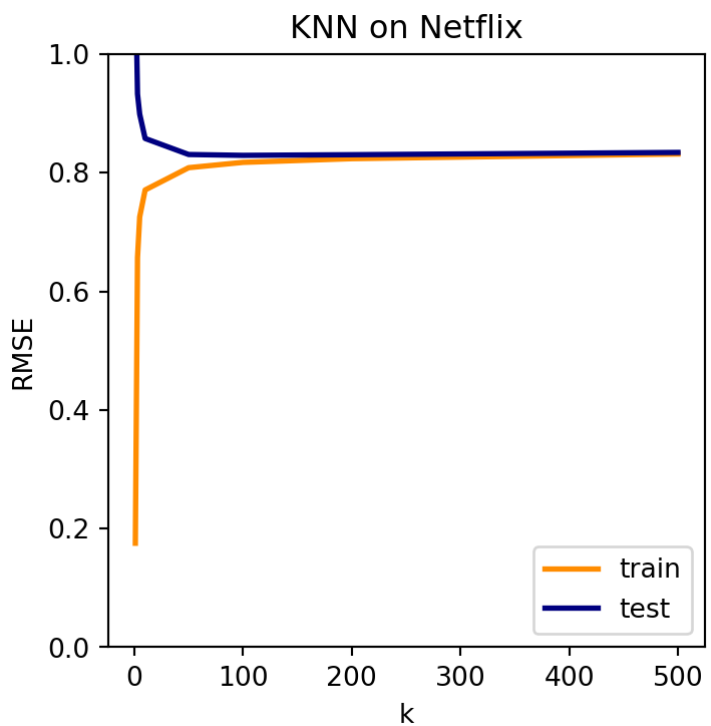
```
NE_Xtr, NE_Xte, NE_Ytr, NE_Yte = train_test_split(netflix_X, netflix_Y, test_size=0.2,

print(f'No. of train rows: {NE_Xtr.shape[0]}, no. train of cols: {NE_Xtr.shape[1]}')
print(f'No. of test rows: {NE_Xte.shape[0]}, no. test of cols: {NE_Xte.shape[1]}')
print(f'no. of obs in train y: {NE_Ytr.shape[0]}')
print(f'no. of obs in test y: {NE_Yte.shape[0]}')
```

```
No. of train rows: 8000, no. train of cols: 14
No. of test rows: 2000, no. test of cols: 14
no. of obs in train y: 8000
no. of obs in test y: 2000
```

KNN for Netflix (Regression)

```
 1 from sklearn.neighbors import KNeighborsRegressor
 2
 3 ntr = NE_Xtr.shape[0]
 4 nte = NE_Xte.shape[0]
 5 tr_err = []
 6 te_err = []
 7 kvals = [1, 3, 5, 10, 50, 100, 200, 500]
 8
 9 for k in kvals:
10     knn = KNeighborsRegressor(n_neighbors = k)
11     knn.fit(NE_Xtr, NE_Ytr)
12     yhat_tr = knn.predict(NE_Xtr)
13     yhat = knn.predict(NE_Xte)
14     tr_err.append(np.sqrt(np.sum((yhat_tr - NE_Ytr)**2) / ntr))
15     te_err.append(np.sqrt(np.sum((yhat - NE_Yte)**2) / nte))
```

```
plt.figure(figsize=(4, 4))
plt.plot(kvals, tr_err, color='darkorange', lw=2, label='train' )
plt.plot(kvals, te_err, color='navy', lw=2, label='test')
plt.ylim([0, 1])
plt.xlabel('k')
plt.ylabel('RMSE')
plt.title('KNN on Netflix')
plt.legend(loc="lower right")
plt.show()
```



The problems with KNN?

1. What is the appropriate distance metric?
2. If the data are “sparse” in the space, nearest neighbors are far and the results can be very bad

- *Curse of dimensionality*: if the dimension p is high, data are by definition sparse
- KNN fails in these settings

Interesting solution to both problems: Adaptive local methods

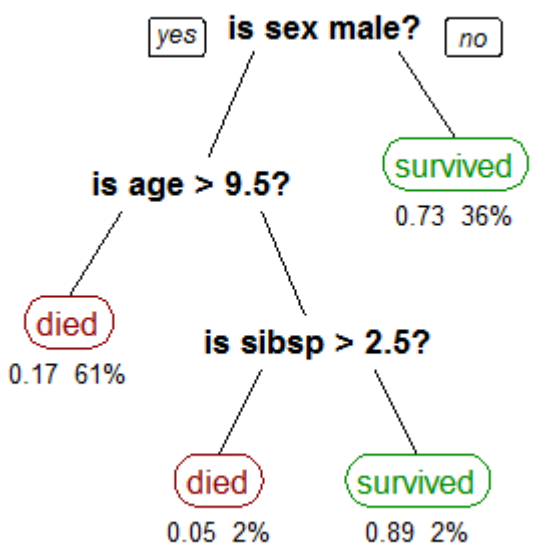
Decision Trees

INTRODUCTION

TO DATA SCIENCE

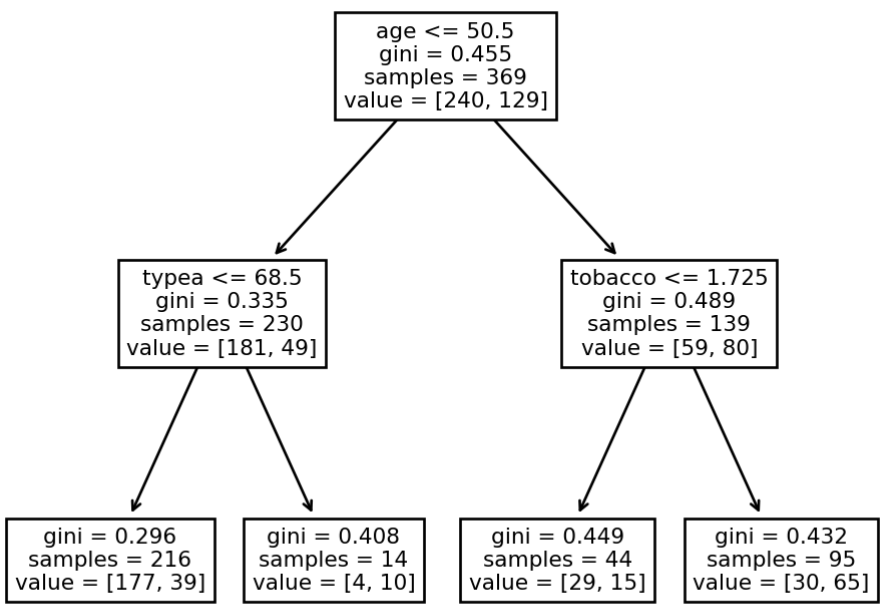
Adaptive local methods: Trees

- The idea: split the space $\mathcal{X}$ into *neighborhoods* by recursive partioning
- Each time: pick a region and split it into two (or more) regions
- Can be described using a tree — binary tree if all splits are in two. Titanic example:



Tree for SAHeart (Classification)

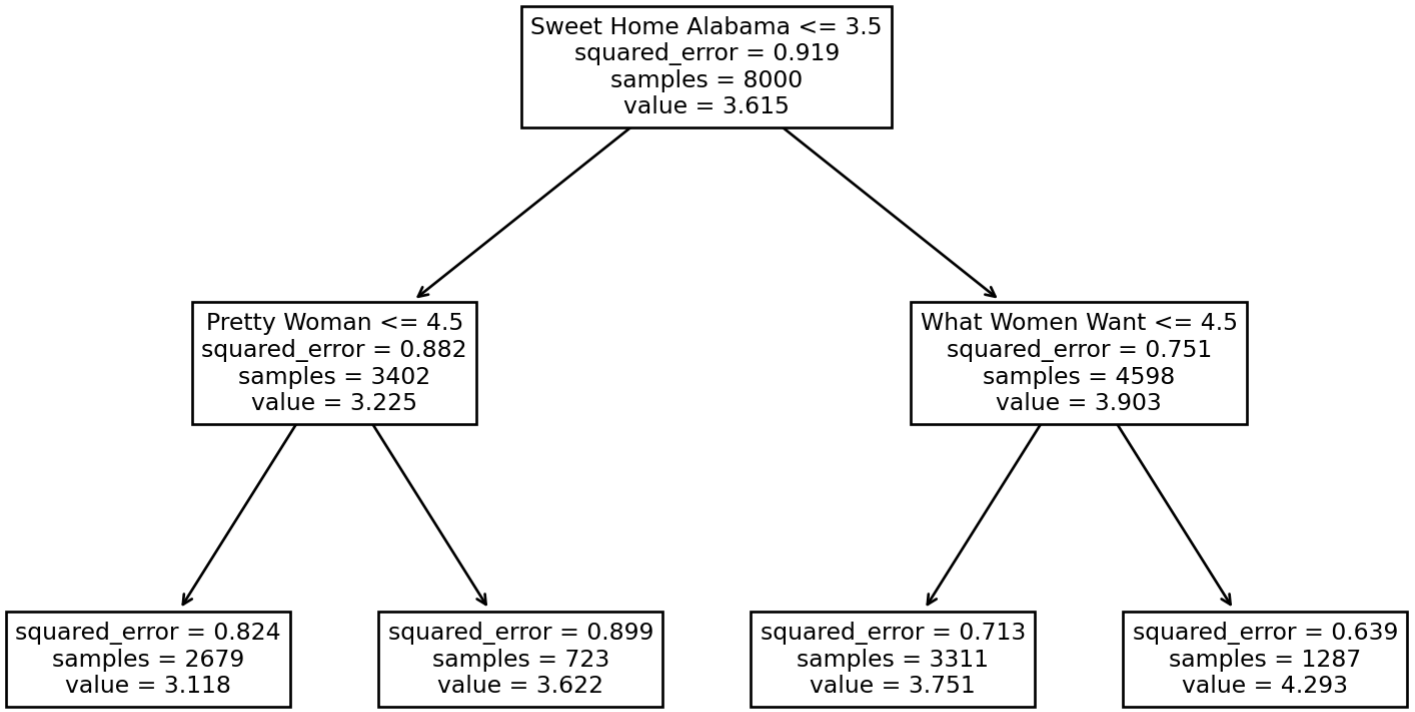
```
1 from sklearn.tree import DecisionTreeClassifier, plot_tree
2
3 tree = DecisionTreeClassifier(max_depth = 2)
4 tree.fit(SA_Xtr, SA_Ytr)
5 plot_tree(tree, feature_names=SA_Xtr.columns)
6 plt.show()
```



```
ntr = SA_Xtr.shape[0]
nte = SA_Xte.shape[0]
tr_err = []
te_err = []
ds = [2, 3, 5, 7, 10, 15]

for depth in ds:
    tree = DecisionTreeClassifier(max_depth = depth)
    tree.fit(SA_Xtr, SA_Ytr)
    yhat_tr = tree.predict(SA_Xtr) > 0.5
    yhat = tree.predict(SA_Xte) > 0.5
    tr_err.append(np.sum(yhat_tr != SA_Ytr) / ntr)
    te_err.append(np.sum(yhat != SA_Yte) / nte)
```

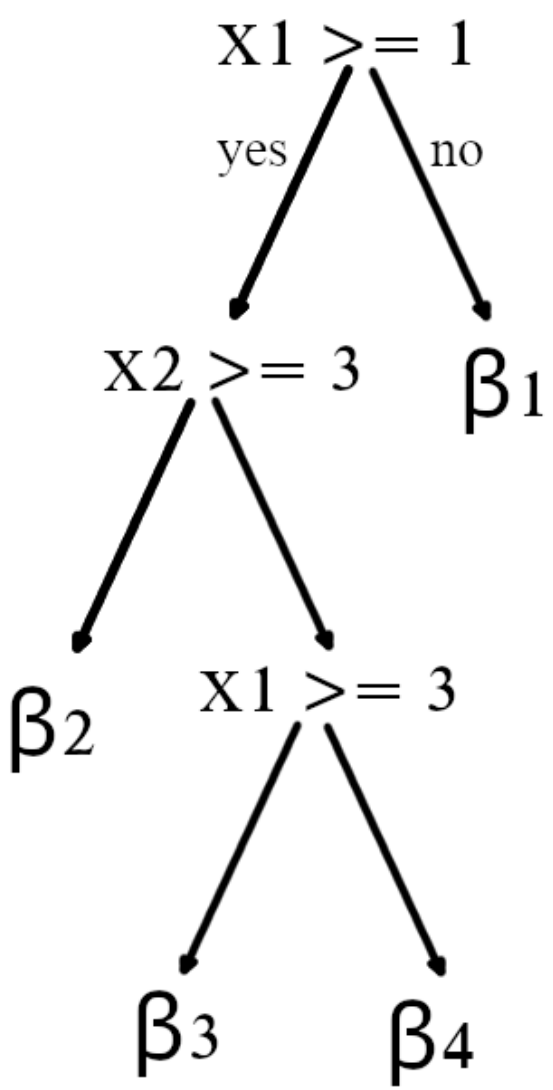
Tree for Netflix (Regression)

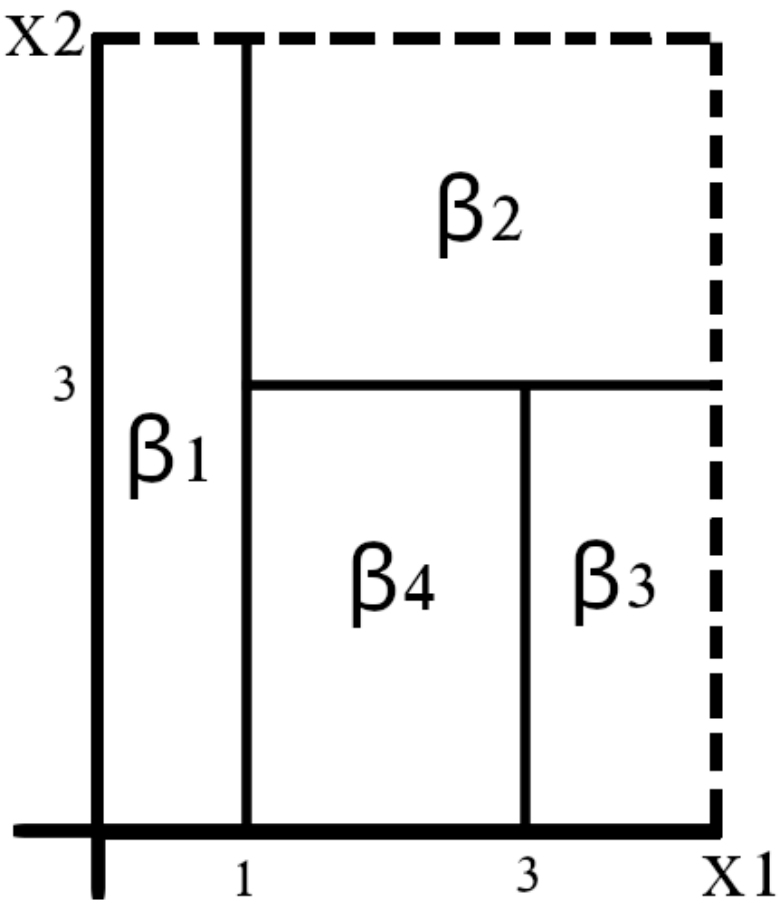
```
from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor(max_depth = 2)
tree.fit(NE_Xtr, NE_Ytr)
plt.figure(figsize=(10, 6))
plot_tree(tree, feature_names=NE_Xtr.columns)
plt.show()
```


Sweet Home Alabama <= 3.5
squared_error = 0.919
samples = 8000
value = 3.615
Pretty Woman <= 4.5
squared_error = 0.882
samples = 3402
value = 3.225
What Women Want <= 4.5
squared_error = 0.751
samples = 4598
value = 3.903
squared_error = 0.824
samples = 2679
value = 3.118
squared_error = 0.899
samples = 723
value = 3.622
squared_error = 0.713
samples = 3311
value = 3.751
squared_error = 0.639
samples = 1287
value = 4.293

```
ntr = NE_Xtr.shape[0]
nte = NE_Xte.shape[0]
tr_err = []
te_err = []
ds = [2, 3, 5, 7, 10, 15]

for depth in ds:
    tree = DecisionTreeRegressor(max_depth = depth)
    tree.fit(NE_Xtr, NE_Ytr)
    yhat_tr = tree.predict(NE_Xtr)
    yhat = tree.predict(NE_Xte)
    tr_err.append(np.sqrt(np.sum((yhat_tr - NE_Ytr)**2) / ntr))
    te_err.append(np.sqrt(np.sum((yhat - NE_Yte)**2) / nte))
```

Rectangular Regions





Building a CART

INTRODUCTION

TO DATA SCIENCE

Defining a decision tree algorithm

There are three main aspects to designing a decision tree algorithm for classification or regression:

1. How do we choose a split at each node of the tree?
2. How do we decide when to stop splitting?
3. How do we fit a value $\hat{y}$ for each terminal node (*leaf*)?

Some well known decision tree algorithms:

- ID3, C4.5, C5.0: for classification only, invented in the CS/machine learning community
- Classification and regression trees (CART): invented in the statistics community
- We are going to mostly describe CART, which is the basis for modern methods we discuss later

CART for regression: splitting process

- Criterion: Minimize RSS on training.
- Given set of r observations in current node, define for a variable j and possible split point s:

$$L(j,s) = \{i \leq r : x_{ij} \leq s\}\ , \quad R(j,s) = \{i \leq r : x_{ij} > s\}$$

$$\bar{y}_L = \frac{\sum_{i \in L(j,s)} y_i}{|L(j,s)|}\ ,\ \bar{y}_R = \frac{\sum_{i \in R(j,s)} y_i}{|R(j,s)|}$$

$$RSS(j,s) = \sum_{i \in L(j,s)} (y_i - \bar{y}_L)^2 + \sum_{i \in R(j,s)} (y_i - \bar{y}_R)^2$$

- And find the pair j, s which minimize this RSS among all possible pairs — this is the split we do
- Split the node into two according to the chosen split and continue

CART for regression: stopping criteria

- Why limit tree size?

- Overfitting, computation,…
- In the examples above: *max_depth* of tree
- Other options: size of nodes not too small, improvement in RSS not too small,…
- Interesting approach of CART: grow a very big tree and *prune* it to smaller tree using test set performance (actually cross-validation, which we have not yet discussed)

CART for regression: fits at leaves

- Similar to OLS, we want to estimate $\hat{y}(x) \approx E(y|x)$
- We interpret the splitting as finding *homogeneous areas* with similar y values in our data, hence hopefully similar $E(y|x)$.
- Consequently, given a leaf (terminal node) with set of observations $Q \subseteq \{1, \ldots, n\}$, we estimate:

$$\hat{y} = \bar{y}_Q = \frac{\sum_{i \in Q} y_i}{|Q|}$$

CART and others for classification

- Various splitting criteria: Gini, information gain, log-likelihood, all give similar trees
- Not a good idea: using misclassification rate as splitting criterion
- Stopping criteria: similar ideas to regression
- Fits at leaves: usually empirical % of classes (or majority if need hard classification)

Closing Remarks

INTRODUCTION

TO DATA SCIENCE

Important properties of trees

1. categoraical features

- Real life data often includes categorical features that have many values but are important for prediction, like:
  1. City of residence
  2. University/department
  3. Customer class

- CART always does binary splits. For a categorical variable with K values $\mathcal{G} = \{g_1, \ldots, g_K\}$ it divides $\mathcal{G}$ into two groups $\mathcal{G}_1, \mathcal{G}_2$ so that:

$$L(j) = \{i : x_{ij} \in \mathcal{G}_1\}\ , \quad \mathcal{R}(j) = \{i : x_{ij} \in \mathcal{G}_2\}.$$

- C4.5/C5.0 do multi-way non-binary splits
- Presents interesting computational and statistical challenges

Important properties of trees:

2. missing data

- Many methods struggle dealing with missing data, trees have nice solutions
- Solution 1 (CART): in addition to the split I want to do, find similar *surrogate splits* on other variables, and if I don't see x_{ij} I can use surrogate split on x_{ik}
- Solution 2 (C4.5): if I want to split on feature j and I don't know x_{ij}, send observation i both left and right

Summary: Decision Trees

Advantages:

1. Intuitive and appealing: divide the space into *flexible* neighborhoods
2. Flexible: categorical variables, missing data, regression or classification, big or small,...
3. Big trees are a very rich class of models: can describe well many true models for $E(y|x)$.

Disadvantages:

1. Intuitive appeal is misleading: very unstable and high variance
2. Not a good prediction model: a single tree is usually not competitive!

Conclusions and next steps:

1. We do not really want to use trees as our prediction models
2. Can we take advantage of their good properties and mitigate the problems?

Intro to Data Science