# Overview

## Data Mining for Business Analytics in Python

**Shmueli, Bruce, Gedeck, & Patel**

 rev 10/17/19

# Core Ideas in Data Mining

- Classification
- Prediction
- Association Rules & Recommenders
- Data & Dimension Reduction
- Data Exploration
- Visualization

# Paradigms for Data Mining (variations)

- SEMMA (from SAS)
  - Sample
  - Explore
  - Modify
  - Model
  - Assess
- CRISP-DM (SPSS/IBM)
  - Business Understanding
  - Data Understanding
  - Data Preparation
  - Modeling
  - Evaluation
  - Deployment

# Supervised Learning

- Goal: Predict a single “target” or “outcome” variable
- Training data, where target value is known
- Score to data where value is not known
- Methods: Classification and Prediction

# Unsupervised Learning

- Goal: Segment data into meaningful segments; detect patterns
- There is no target (outcome) variable to predict or classify
- Methods: Association rules, collaborative filters, data reduction & exploration, visualization

# Supervised: Classification

- Goal: Predict categorical target (outcome) variable
- Examples: Purchase/no purchase, fraud/no fraud, creditworthy/not creditworthy…
- Each row is a case (customer, tax return, applicant)
- Each column is a variable
- Target variable is often binary (yes/no)

# Supervised: Prediction

- Goal: Predict numerical target (outcome) variable
- Examples: sales, revenue, performance
- As in classification:
- Each row is a case (customer, tax return, applicant)
- Each column is a variable
- Taken together, classification and prediction constitute “predictive analytics”

# Unsupervised: Association Rules

- Goal: Produce rules that define “what goes with what” in transactions
- Example: “If X was purchased, Y was also purchased”
- Rows are transactions
- Used in recommender systems – “Our records show you bought X, you may also like Y”
- Also called “affinity analysis”

# Unsupervised: Collaborative Filtering

- Goal: Recommend products to purchase
- Based on products that customer rates, selects, views, or purchases
- Recommend products that “customers like you” purchase (user-based)
- Or, recommend products that share a “product purchaser profile” with your purchases (item-based)

# Unsupervised: Data Reduction

- Distillation of complex/large data into simpler/smaller data
- Reducing the number of variables/columns (e.g., principal components)
- Reducing the number of records/rows (e.g., clustering)

# Unsupervised: Data Visualization

- Graphs and plots of data
- Histograms, boxplots, bar charts, scatterplots
- Especially useful to examine relationships between pairs of variables

# Data Exploration

- Data sets are typically large, complex & messy
- Need to review the data to help refine the task
- Use techniques of Reduction and Visualization

# The Process of Data Mining

# Steps in Data Mining

1. Define/understand purpose
2. Obtain data (may involve random sampling)
3. Explore, clean, pre-process data
4. Reduce the data; if supervised DM, partition it
5. Specify task (classification, clustering, etc.)
6. Choose the techniques (regression, CART, neural networks, etc.)
7. Iterative implementation and “tuning”
8. Assess results – compare models
9. Deploy best model

# Preliminary Exploration in Python

## loading data, viewing it, summary statistics

Python Hands-on

# Obtaining Data: Sampling

- Data mining typically deals with huge databases
- For piloting/prototyping, algorithms and models are typically applied to a sample from a database, to produce statistically-valid results
- Once you develop and select a final model, you use it to “score” (predict values or classes for) the observations in the larger database

# Rare Event Oversampling

- Often the event of interest is rare
- Examples: response to mailing, fraud in taxes, …
- Sampling may yield too few “interesting” cases to effectively train a model
- A popular solution: oversample the rare cases to obtain a more balanced training set
- Later, need to adjust results for the oversampling

# Sampling & Oversampling

**Python Hands-on**

# Types of Variables

- Determine the types of pre-processing needed, and algorithms used
- Main distinction: Categorical vs. numeric
- Numeric
  - Continuous
  - Integer
- Categorical
  - Ordered (low, medium, high)
  - Unordered (male, female)

# Variable handling

- Numeric
  - Most algorithms can handle numeric data
  - May occasionally need to “bin” into categories
- Categorical
  - Naïve Bayes can use as-is
  - In most other algorithms, must create binary dummies (number of dummies = number of categories – 1) [see Table 2.6 for R code]

# Reviewing Variables

Python Hands-on

# Creating binary dummies

Python Hands-on

output on next slide…

# Detecting Outliers

- An outlier is an observation that is “extreme”, being distant from the rest of the data (definition of “distant” is deliberately vague)
- Outliers can have disproportionate influence on models (a problem if it is spurious)
- An important step in data pre-processing is detecting outliers
- Once detected, domain knowledge is required to determine if it is an error, or truly extreme.

# Detecting Outliers

- In some contexts, finding outliers is the purpose of the DM exercise (airport security screening). This is called “anomaly detection”.

# Handling Missing Data

- Most algorithms will not process records with missing values. Default is to drop those records.
- Solution 1: Omission
  - If a small number of records have missing values, can omit them
  - If many records are missing values on a small set of variables, can drop those variables (or use proxies)
  - If many records have missing values, omission is not practical
- Solution 2: Imputation
  - Replace missing values with reasonable substitutes
  - Lets you keep the record and use the rest of its (non-missing) information

# Replacing Missing Data

Python Hands-on

# Normalizing (Standardizing) Data

- Used in some techniques when variables with the largest scales would dominate and skew results
- Puts all variables on same scale
- Normalizing function: Subtract mean and divide by standard deviation
- Alternative function: scale to 0-1 by subtracting minimum and dividing by the range
  - Useful when the data contain dummies and numeric

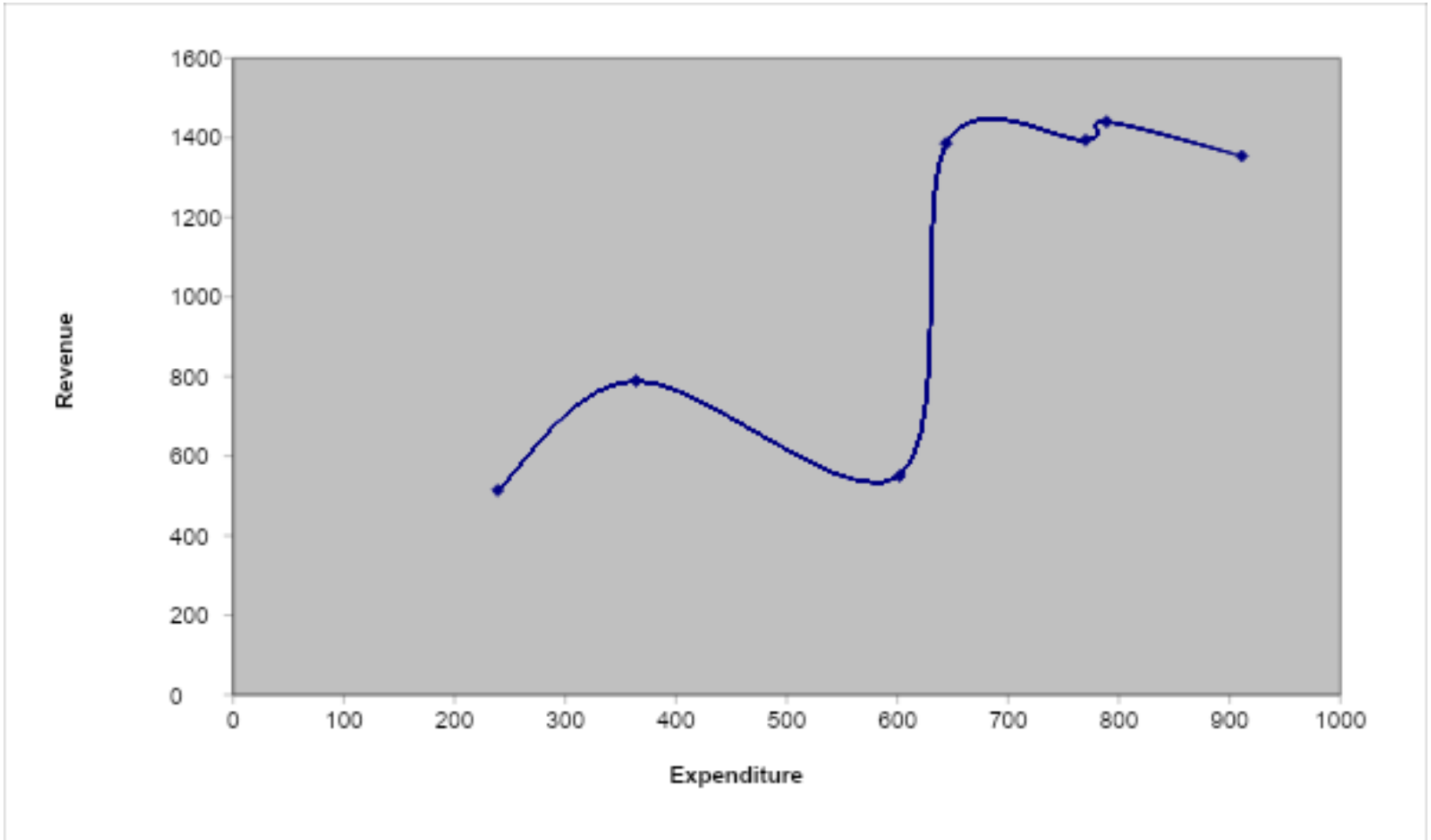
# Code for Normalizing Data

Python Hands-on

# The Problem of Overfitting

- Statistical models can produce highly complex explanations of relationships between variables
- The “fit” may be excellent
- When used with new data, models of great complexity do not do so well.

# 100% fit – not useful for new data



# Overfitting (cont.)

Causes:

- Too many predictors
- A model with too many parameters
- Trying many different models

Consequence: Deployed model will not work as well as expected with completely new data.

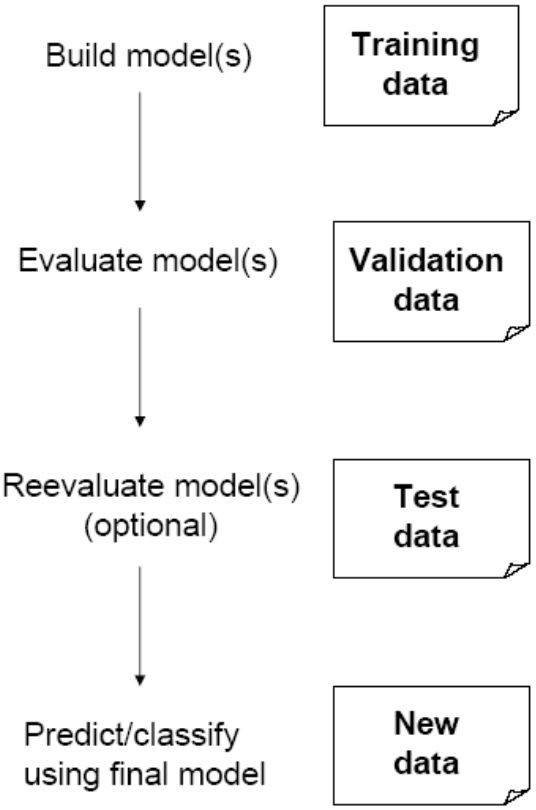
# Partitioning the Data

Problem: How well will our model perform with new data?

Solution: Separate data into two parts

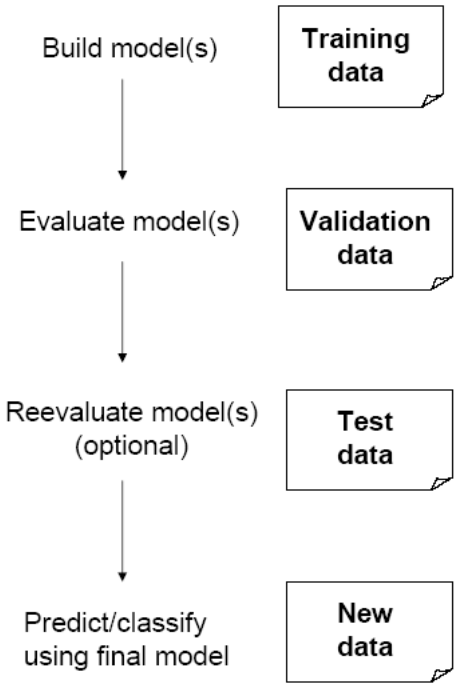
- Training partition to develop the model
- Validation partition to implement the model and evaluate its performance on "new" data

Addresses the issue of overfitting



# Test Partition

- When a model is developed on **training data**, it can overfit the training data (hence need to assess on validation)
- Assessing multiple models on same **validation data** can overfit validation data
- Some methods use the validation data to choose a parameter. This too can lead to overfitting the validation data
- Solution: final selected model is applied to a **test partition** to give unbiased estimate of its performance on new data



# “Test” Partition - Terminology

- In *Data Mining for Business Analytics*, the “test” partition is the third partition for unbiased assessment.
- More generally in data science, and in Python syntax, “test” refers to the partition set aside from the training partition

# Partitioning the Data

Python Hands-on

# Cross Validation

- Repeated partitioning = cross-validation (“cv”)
- k-fold cross validation,  e.g. k=5
  - For each fold, set aside ⅕ of data as validation
  - Use full remainder as training
  - The validation folds are non-overlapping
- In Python
  - `cross\_val\_score()`
  - more general `cross\_validate`
  - argument `cv` determines the number of folds

# Example – Linear Regression West Roxbury Housing Data

**TABLE 2.1 DESCRIPTION OF VARIABLES IN WEST ROXBURY (BOSTON) HOME VALUE DATASET**

| | |
|---|---|
| TOTAL VALUE | Total assessed value for property, in thousands of USD |
| TAX | Tax bill amount based on total assessed value multiplied by the tax rate, in USD |
| LOT SQ FT | Total lot size of parcel in square feet |
| YR BUILT | Year the property was built |
| GROSS AREA | Gross floor area |
| LIVING AREA | Total living area for residential properties ($ft^2$) |
| FLOORS | Number of floors |
| ROOMS | Total number of rooms |
| BEDROOMS | Total number of bedrooms |
| FULL BATH | Total number of full baths |
| HALF BATH | Total number of half baths |
| KITCHEN | Total number of kitchens |
| FIREPLACE | Total number of fireplaces |
| REMODEL | When the house was remodeled (Recent/Old/None) |

# Error metrics

Error = actual – predicted

ME = Mean error

RMSE = Root-mean-squared error = Square root of average squared error

MAE = Mean absolute error

MPE = Pean percentage error

MAPE = Mean absolute percentage error

# Summary

- Data mining consists of supervised methods (Classification & Prediction) and unsupervised methods (Association Rules, Data Reduction, Data Exploration & Visualization)
- Before algorithms can be applied, data must be explored and pre-processed
- To evaluate performance and to avoid overfitting, data partitioning is used
- Models are fit to the training partition and assessed on the validation and test partitions
- Data mining methods are usually applied to a sample from a large database, and then the best model is used to score the entire database

# Chapter 3 – Data Visualization

## Data Mining for Business Analytics in Python

**Shmueli, Bruce, Gedeck & Patel**

# Plots for Data Exploration

Basic Plots
- Line Graphs
- Bar Charts
- Scatterplots

Distribution Plots
- Boxplots
- Histograms

# Possible Goals in Visualization

1. Presentation (reporting & story-telling)
2. Exploration and preliminary analysis

Our focus here is mainly #2.

# The Term “graph”

1. Can mean a figure to represent data (bar graph, etc.)
2. A slightly more technical meaning is a data structure for connections among items or people

To avoid ambiguity, we try to use “plot” for the first meaning.

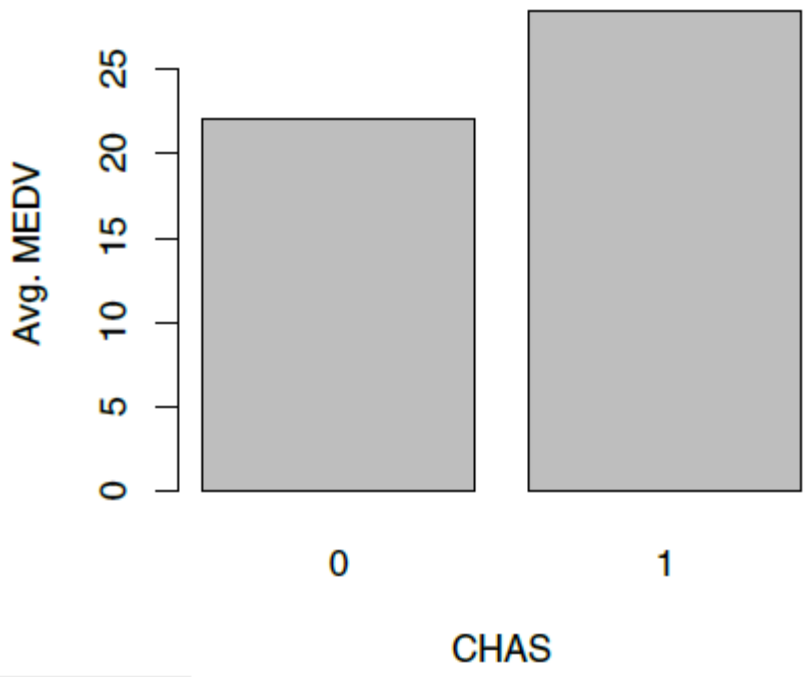
# Python Libraries

`matplotlib` - oldest and most flexible

`seaborn`, `pandas` - wrappers around `matplotlib`; help create plots quickly, but knowledge of `matplotlib` allows better control over final plot

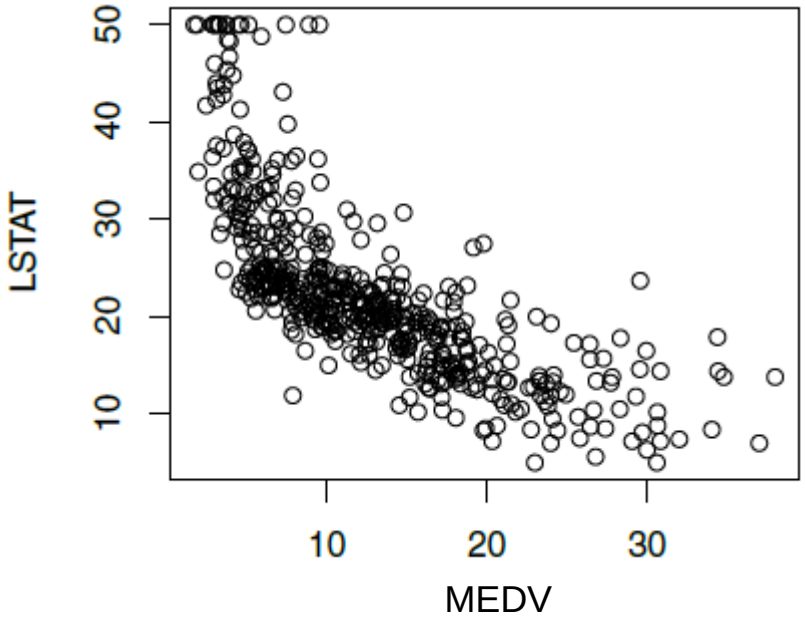
# Bar Chart for Categorical Variable

Average median neighborhood value for neighborhoods that do and do not border the Charles River



# Scatterplot

Displays relationship between two numerical variables



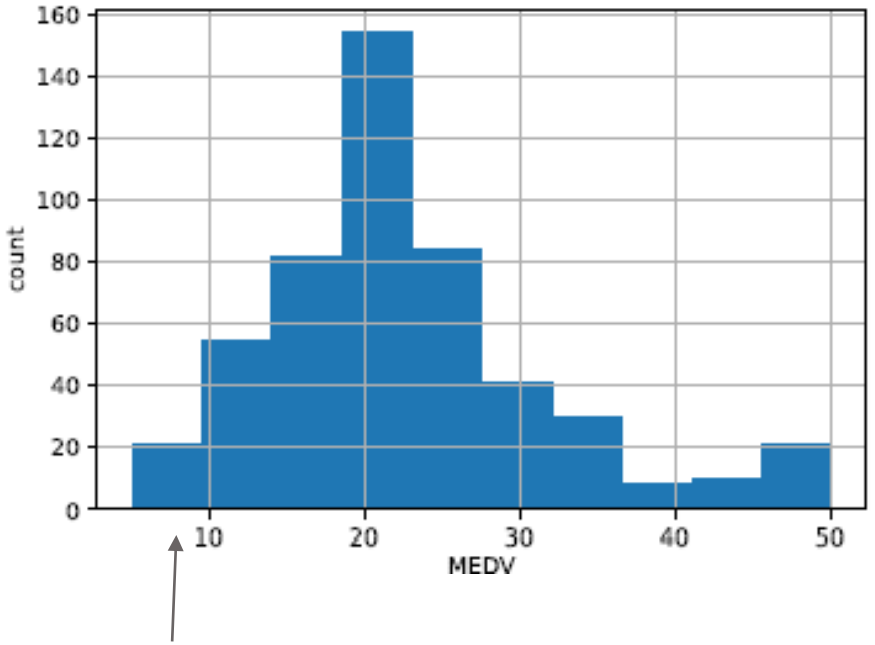
# Distribution Plots

- Display “how many” of each value occur in a data set
- Or, for continuous data or data with many possible values, “how many” values are in each of a series of ranges or “bins”

# Histograms

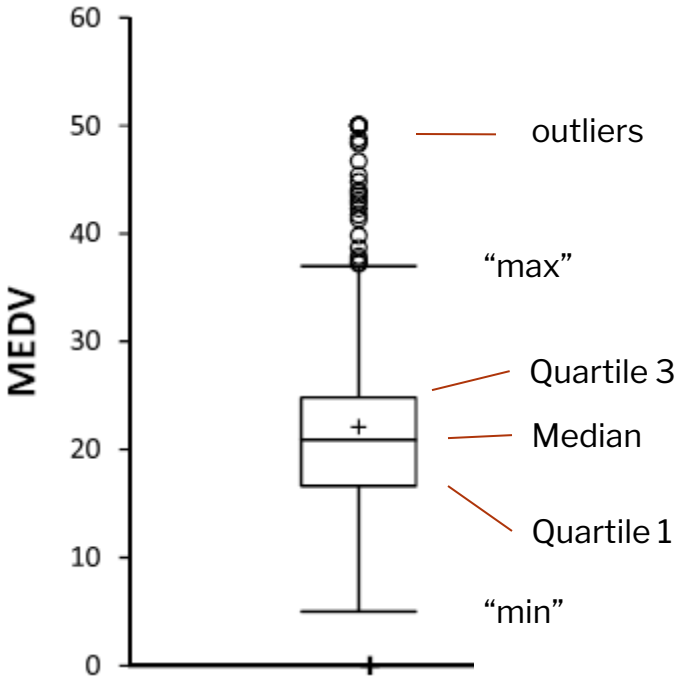
Boston Housing example:

Histogram shows the distribution of the outcome variable (median house value)




About 20 neighborhoods had a median house value in the lowest bin, about $4000K to $9500 (these data are from mid-20th century)

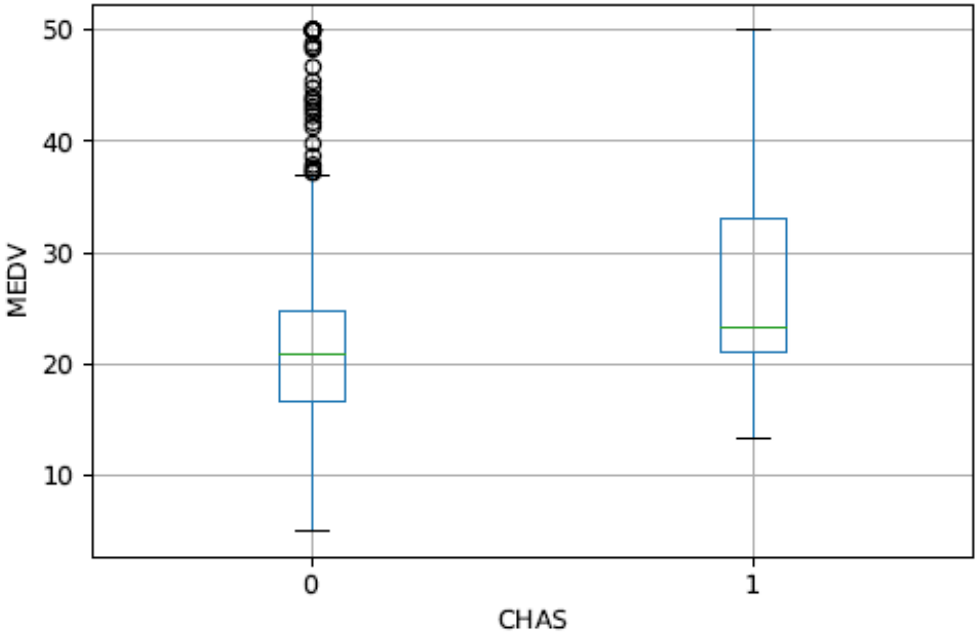
# Box Plot




- Top outliers defined as those above Q3+1.5(Q3-Q1).
- “max” = maximum of non-outliers
- Analogous definitions for bottom outliers and for “min”
- Details may differ across software

# Boxplots

Side-by-side boxplots are useful for comparing subgroups

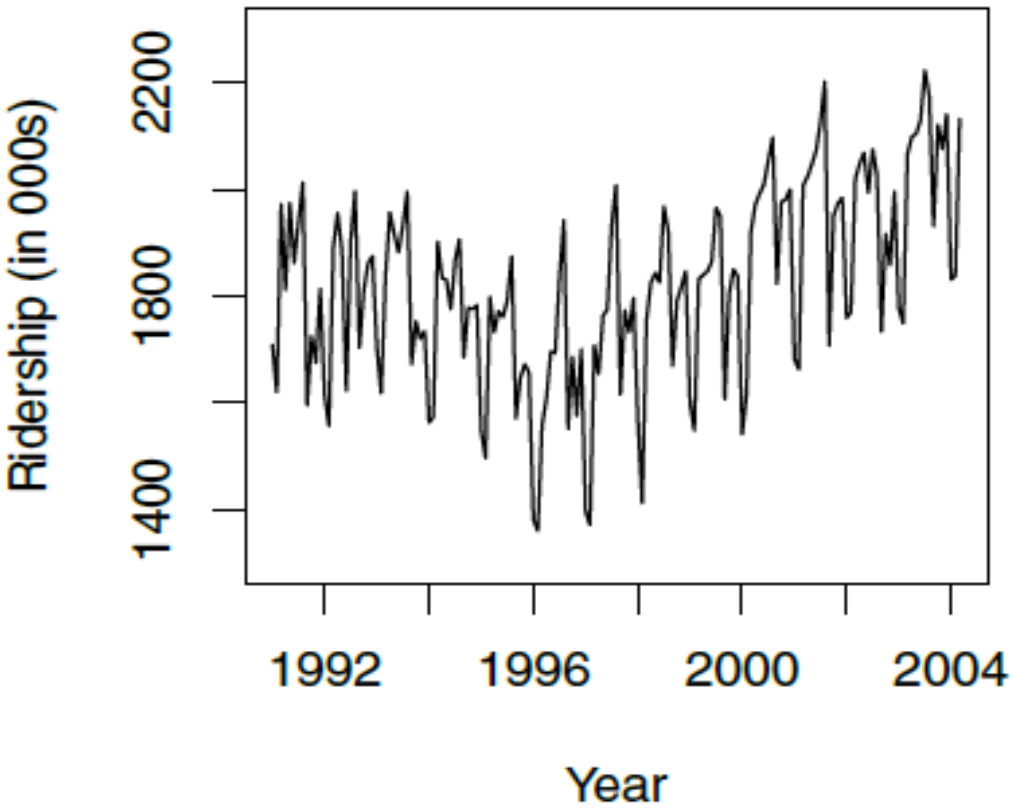



Houses in neighborhoods on Charles river (1) are more valuable than those not (0)

```
ax = housing_df.boxplot(column='MEDV', by='CHAS')
ax.set_ylabel('MEDV')
plt.suptitle('')  # Suppress the titles
plt.title('')
```

# Line Plot for Time Series

## Amtrak Ridership



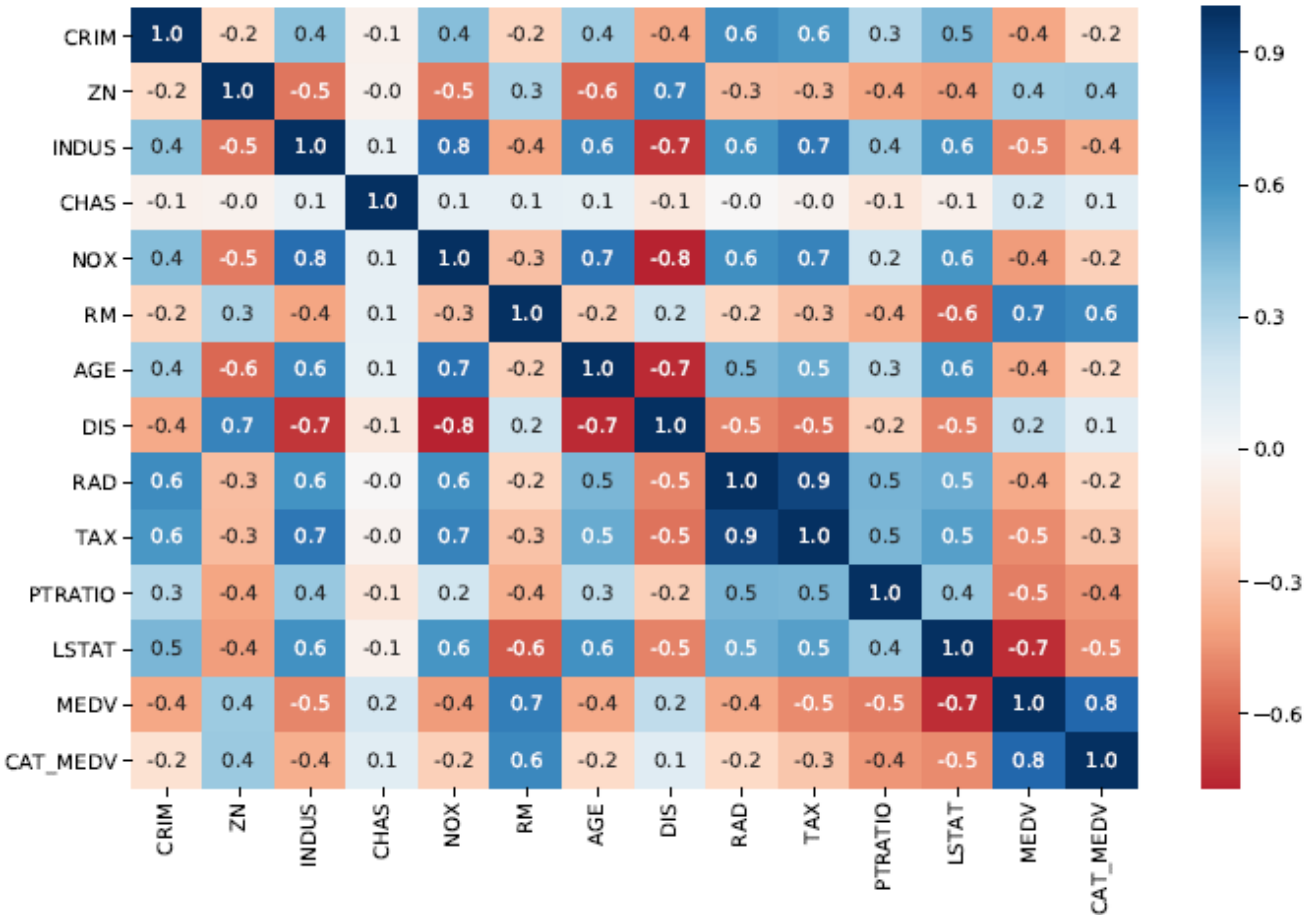
# Heat Maps

Color conveys information

In data mining, used to visualize
- Correlations
- Missing Data

# Heatmap to highlight correlations

Darker and bluer = stronger positive correlation
Darker & redder = stronger negative correlation



# Multidimensional Visualization

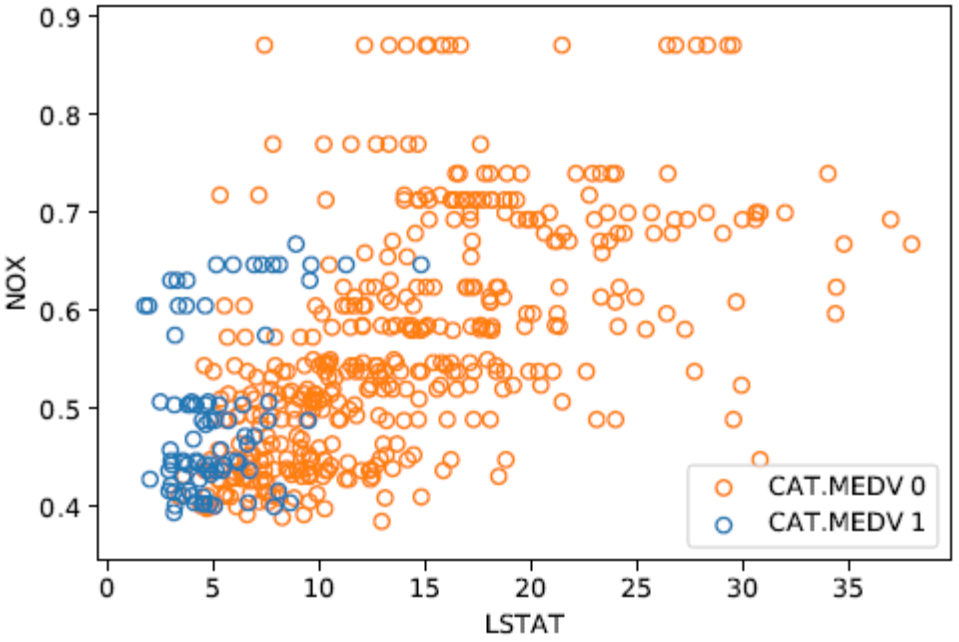
# Scatterplot with color added

Boston Housing

NOX vs. LSTAT

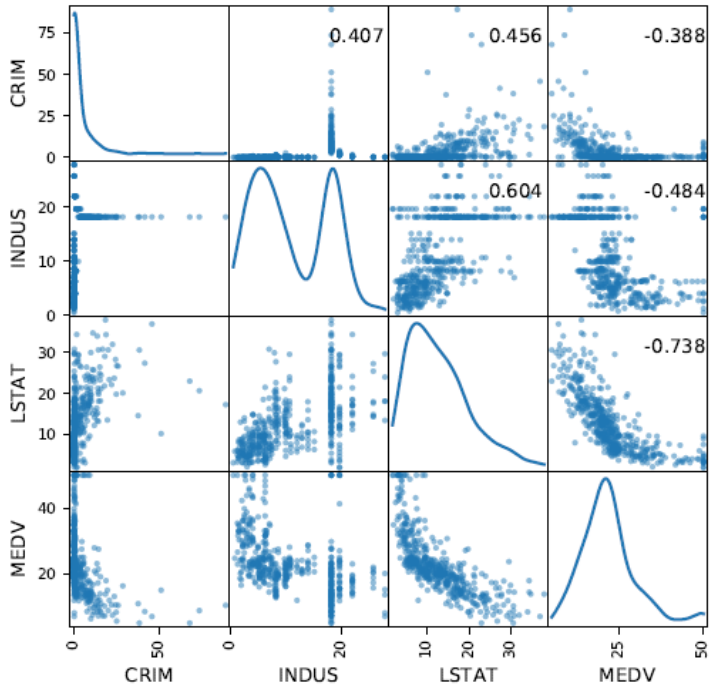
low median value

high median value



# Matrix Scatterplot

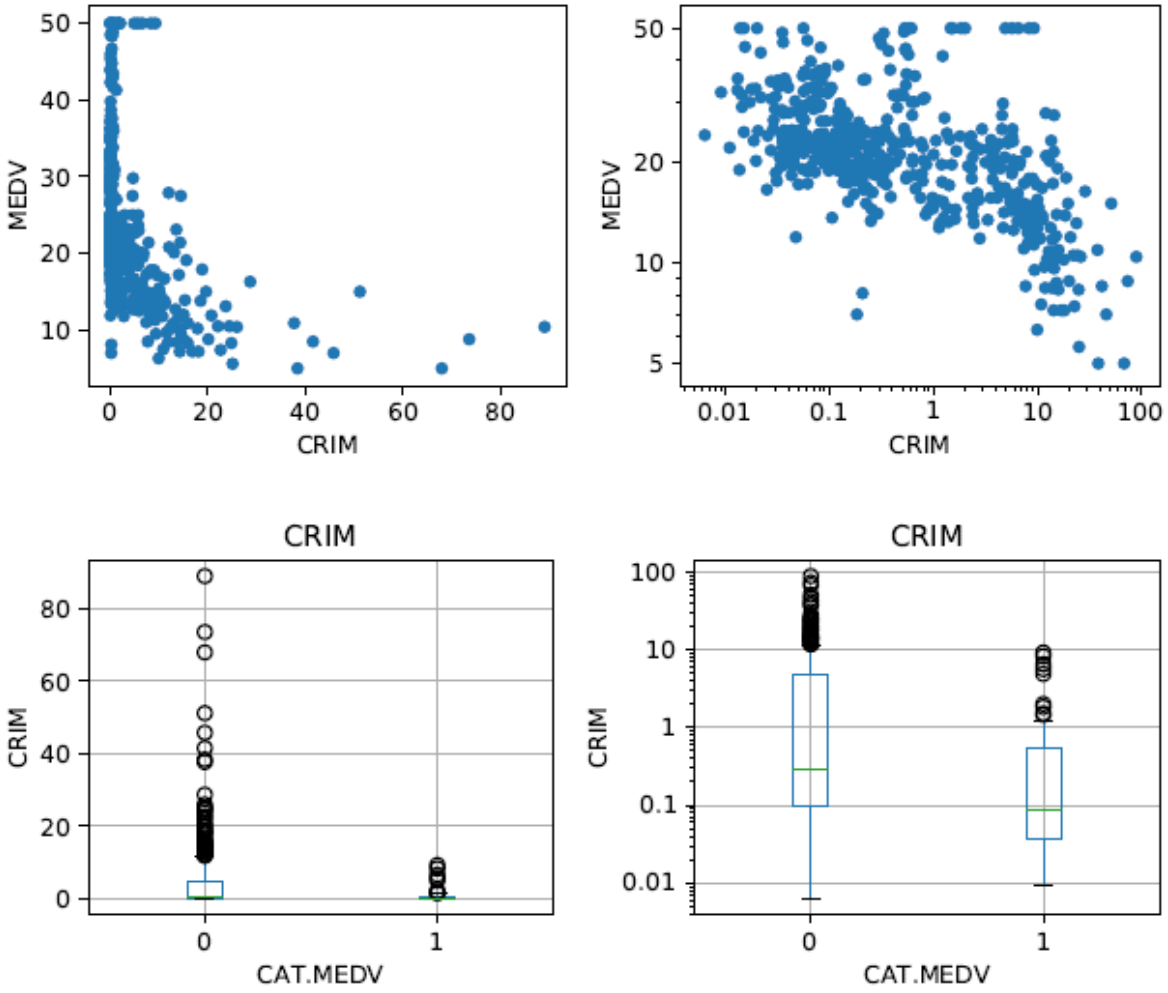
Diagonal plot is the frequency distribution for the variable



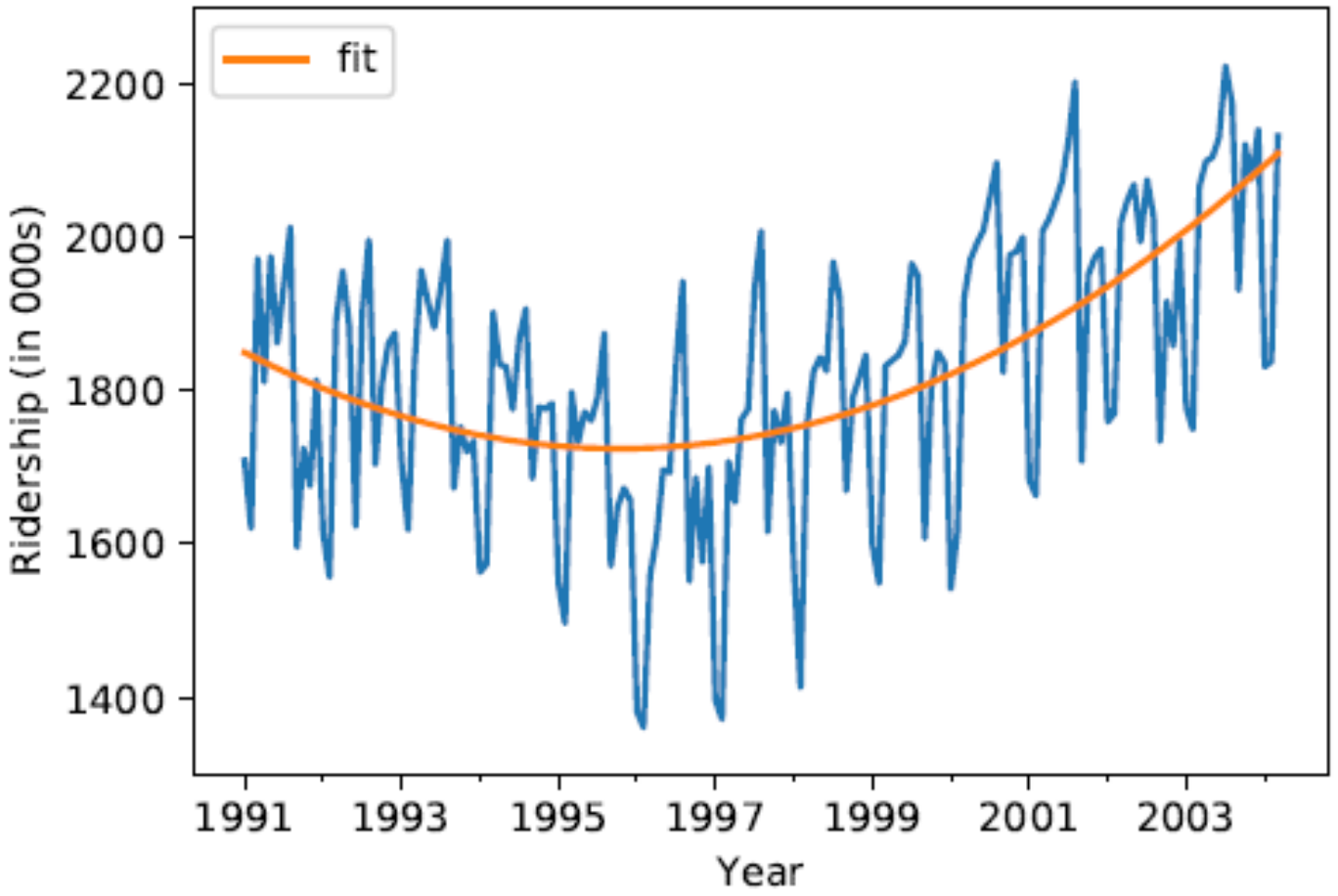
Manipulation:

- Rescaling
- Aggregation
- Zooming
- Filtering

# Rescaling “CRIM” to log scale (on right) “uncrowds” the data

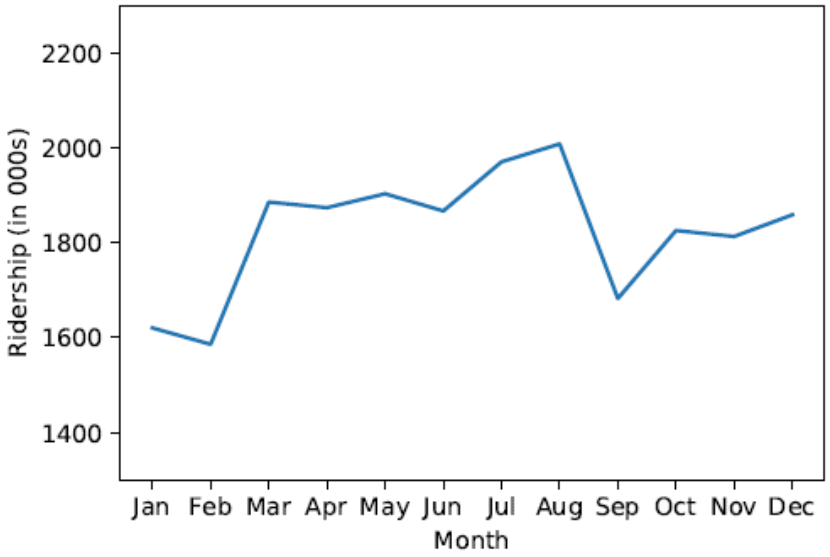


# Amtrak Ridership – Monthly Data – Curve Added

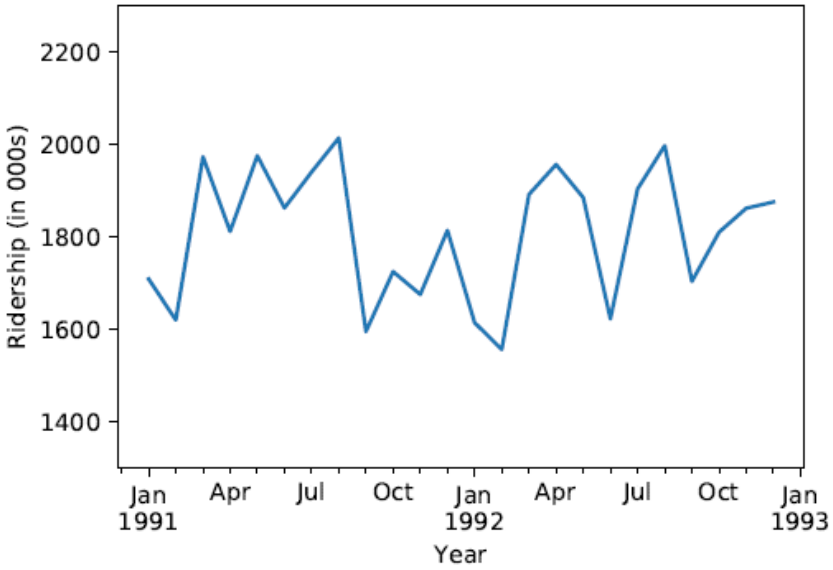


# Amtrak Ridership – Different Aggregations

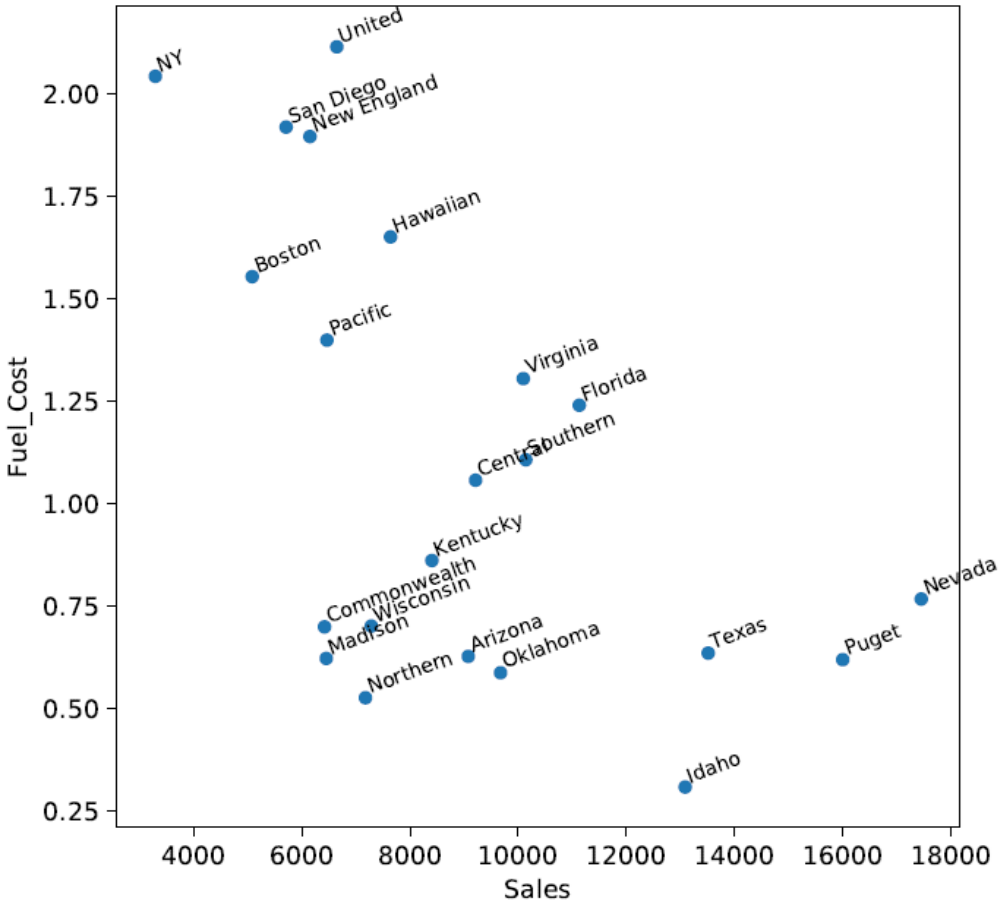
Monthly Averages



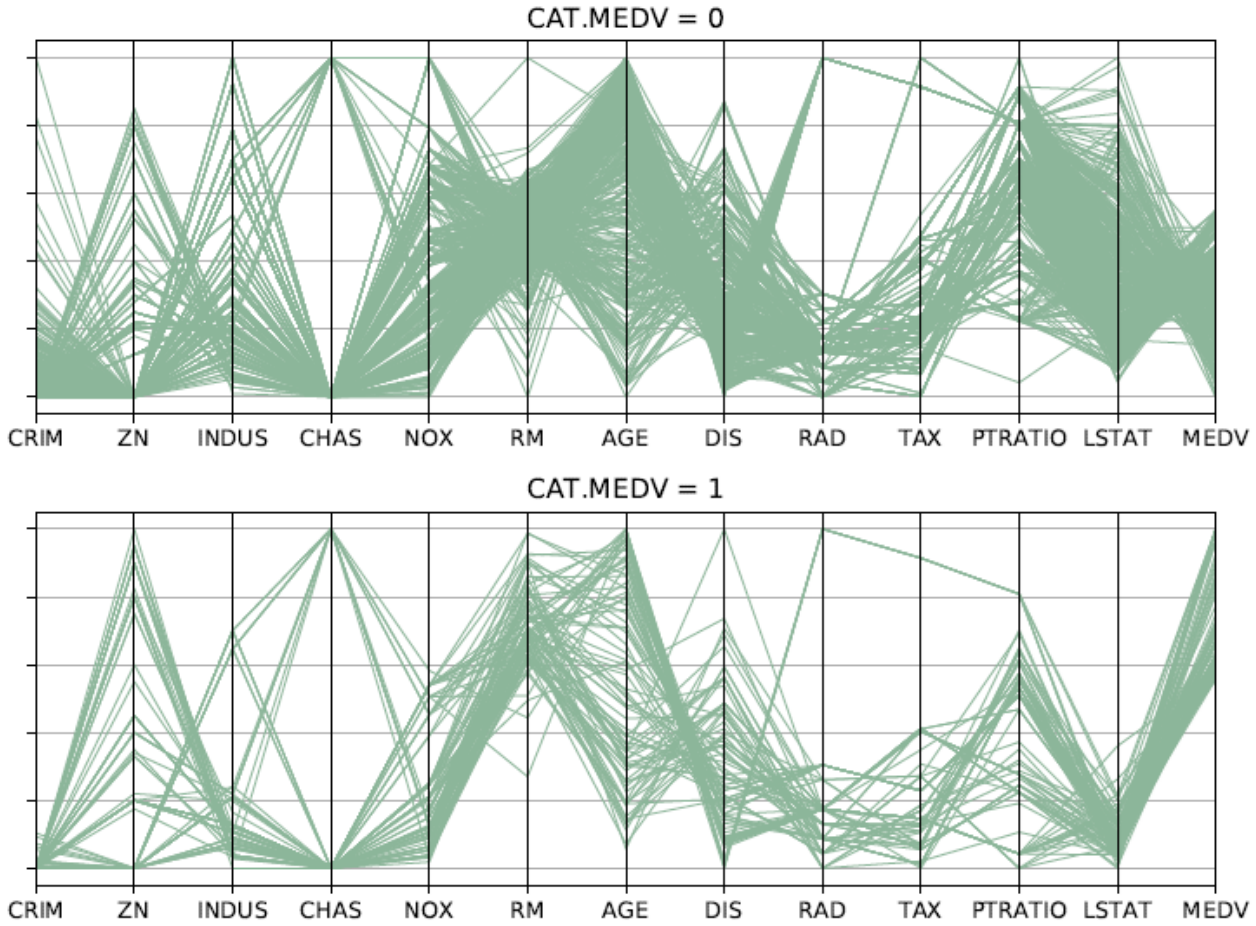

Zoom In



# Scatter Plot with Labels (Utilities)



# Parallel Coordinate Plot (Boston Housing)




All variables are rescaled to 0-1 scale

Each line is a single record

# Chapter 4 – Dimension Reduction

## Data Mining for Business Analytics in Python

**Shmueli, Bruce, Gedeck & Patel**

 rev 10/17/19

# Exploring the data

Statistical summary of data: common metrics

- Average
- Median
- Minimum
- Maximum
- Standard deviation
- Counts & percentages

# Summary Statistics for Boston Housing Data

| | mean | sd | min | max | median | length | miss.val |
|---|---|---|---|---|---|---|---|
| CRIM | 3.61352356 | 8.6015451 | 0.00632 | 88.9762 | 0.25651 | 506 | 0 |
| ZN | 11.36363636 | 23.3224530 | 0.00000 | 100.0000 | 0.00000 | 506 | 0 |
| INDUS | 11.13677866 | 6.8603529 | 0.46000 | 27.7400 | 9.69000 | 506 | 0 |
| CHAS | 0.06916996 | 0.2539940 | 0.00000 | 1.0000 | 0.00000 | 506 | 0 |
| NOX | 0.55469506 | 0.1158777 | 0.38500 | 0.8710 | 0.53800 | 506 | 0 |
| RM | 6.28463439 | 0.7026171 | 3.56100 | 8.7800 | 6.20850 | 506 | 0 |
| AGE | 68.57490119 | 28.1488614 | 2.90000 | 100.0000 | 77.50000 | 506 | 0 |
| DIS | 3.79504269 | 2.1057101 | 1.12960 | 12.1265 | 3.20745 | 506 | 0 |
| RAD | 9.54940711 | 8.7072594 | 1.00000 | 24.0000 | 5.00000 | 506 | 0 |
| TAX | 408.23715415 | 168.5371161 | 187.00000 | 711.0000 | 330.00000 | 506 | 0 |
| PTRATIO | 18.45553360 | 2.1649455 | 12.60000 | 22.0000 | 19.05000 | 506 | 0 |
| LSTAT | 12.65306324 | 7.1410615 | 1.73000 | 37.9700 | 11.36000 | 506 | 0 |
| MEDV | 22.53280632 | 9.1971041 | 5.00000 | 50.0000 | 21.20000 | 506 | 0 |
| CAT.MEDV | 0.16600791 | 0.3724560 | 0.00000 | 1.0000 | 0.00000 | 506 | 0 |

# Reducing Categories

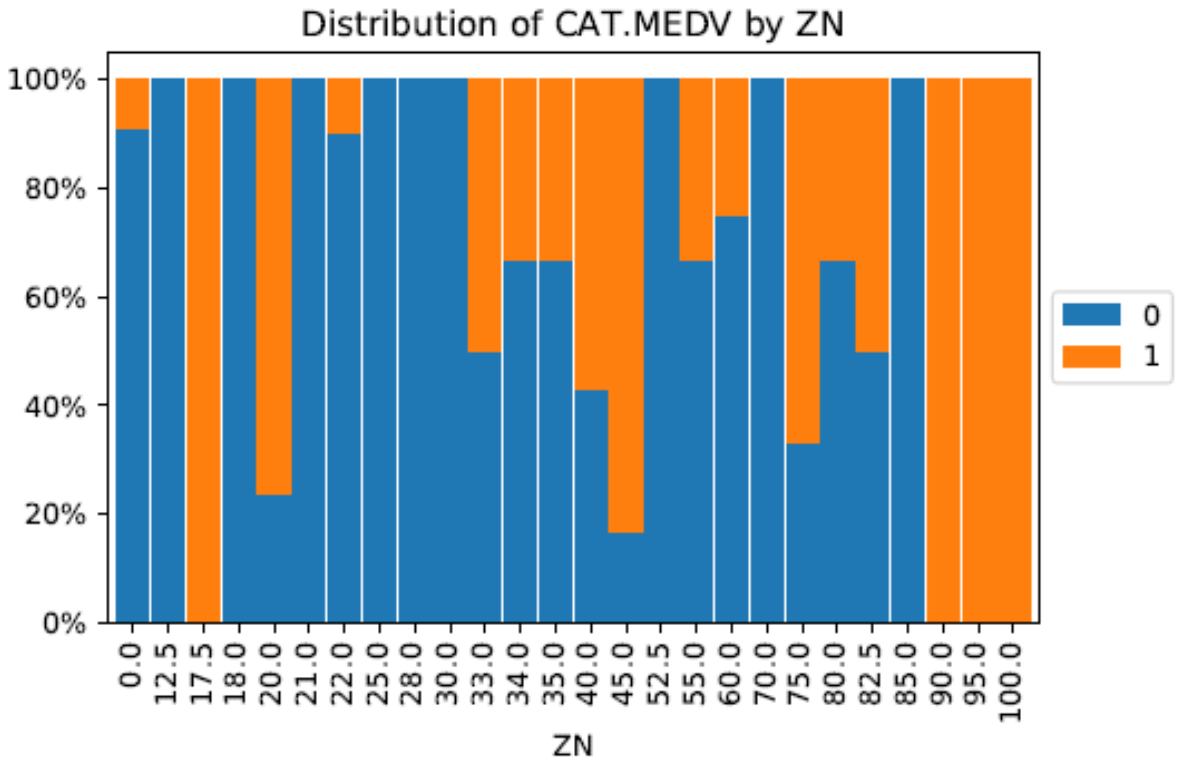
- A single categorical variable with *m* categories is typically transformed into *m* or *m-1* dummy variables (handled automatically by most R/Python modeling functions
- Each dummy variable takes the values 0 or 1

  0 = “no” for the category

  1 = “yes”
- Problem: Can end up with too many variables
- Solution: Reduce by combining categories that are close to each other
- Use pivot tables to assess outcome variable sensitivity to the dummies
- Exception: Naïve Bayes can handle categorical variables without transforming them into dummies

# Combining Categories

Stacked bar chart: Many zoning categories are the same or similar with respect to CATMEDV



# Principal Components Analysis

**Goal:** Reduce a set of **numerical** variables.

**The idea**: Remove the overlap of information between these variable. [“Information” is measured by the sum of the variances of the variables.]

**Final product:** A smaller number of numerical variables that contain most of the information

# Principal Components Analysis

**How does PCA do this?**

- Create new variables that are linear combinations of the original variables (i.e., they are weighted averages of the original variables).

- These linear combinations are uncorrelated (no information overlap), and only a few of them contain most of the original information.

- The new variables are called *principal components*

# Example – Breakfast Cereals (excerpt)

| name | mfr | type | calories | protein | … | rating |
|---|---|---|---|---|---|---|
| 100%_Bran | N | C | 70 | 4 | … | 68 |
| 100%_Natural_Bran | Q | C | 120 | 3 | … | 34 |
| All-Bran | K | C | 70 | 4 | … | 59 |
| All-Bran_with_Extra_Fiber | K | C | 50 | 4 | … | 94 |
| Almond_Delight | R | C | 110 | 2 | … | 34 |
| Apple_Cinnamon_Cheerios | G | C | 110 | 2 | … | 30 |
| Apple_Jacks | K | C | 110 | 2 | … | 33 |
| Basic_4 | G | C | 130 | 3 | … | 37 |
| Bran_Chex | R | C | 90 | 2 | … | 49 |
| Bran_Flakes | P | C | 90 | 3 | … | 53 |
| Cap'n'Crunch | Q | C | 120 | 1 | … | 18 |
| Cheerios | G | C | 110 | 6 | … | 51 |
| Cinnamon_Toast_Crunch | G | C | 120 | 1 | … | 20 |

# Description of Variables

**Name**: name of cereal

**mfr**: manufacturer

**type**: cold or hot

**calories**: calories per serving

**protein**: grams

**fat**: grams

**sodium**: mg.

**fiber**: grams

**carbo**: grams complex carbohydrates

**sugars**: grams

**potass**: mg.

**vitamins**: % FDA rec

**shelf**: display shelf

**weight**: oz. 1 serving

**cups**: in one serving

**rating**: consumer reports

# Principal Component Scores for the First Five Records

| | PC1 | PC2 |
|---|---|---|
| [1,] | 44.921528 | 2.1971833 |
| [2,] | -15.725265 | -0.3824165 |
| [3,] | 40.149935 | -5.4072123 |
| [4,] | 75.310772 | 12.9991256 |
| [5,] | -7.041508 | -5.3576857 |

# The Weightings for the First Five Components

| | PC1 | PC2 | PC3 | PC4 | PC5 |
|---|---|---|---|---|---|
| calories | -0.077984 | -0.009312 | 0.629206 | -0.601021 | 0.454959 |
| protein | 0.000757 | 0.008801 | 0.001026 | 0.003200 | 0.056176 |
| fat | 0.000102 | 0.002699 | 0.016196 | -0.025262 | -0.016098 |
| sodium | -0.980215 | 0.140896 | -0.135902 | -0.000968 | 0.013948 |
| fiber | 0.005413 | 0.030681 | -0.018191 | 0.020472 | 0.013605 |
| carbo | -0.017246 | -0.016783 | 0.017370 | 0.025948 | 0.349267 |
| sugars | -0.002989 | -0.000253 | 0.097705 | -0.115481 | -0.299066 |
| potass | 0.134900 | 0.986562 | 0.036782 | -0.042176 | -0.047151 |
| vitamins | -0.094293 | 0.016729 | 0.691978 | 0.714118 | -0.037009 |
| shelf | 0.001541 | 0.004360 | 0.012489 | 0.005647 | -0.007876 |
| weight | -0.000512 | 0.000999 | 0.003806 | -0.002546 | 0.003022 |
| cups | -0.000510 | -0.001591 | 0.000694 | 0.000985 | 0.002148 |
| rating | 0.075296 | 0.071742 | -0.307947 | 0.334534 | 0.757708 |

# Generalization

$X_1, X_2, X_3, \ldots X_p$, original *p* variables

$Z_1, Z_2, Z_3, \ldots Z_p$, weighted averages of original variables

All pairs of Z variables have 0 correlation

Order Z's by variance ($z_1$ largest, $Z_p$ smallest)

Usually the first few Z variables contain most of the information, and so the rest can be dropped.

# Normalizing data

- In these results, sodium dominates first PC
- Just because of the way it is measured (mg), its scale is greater than almost all other variables
- Hence its variance will be a dominant component of the total variance
- Normalize each variable to remove scale effect

  Divide by std. deviation (may subtract mean first)
- Normalization (= standardization) is usually performed in PCA; otherwise measurement units affect results

```
pcs = PCA()
pcs.fit(preprocessing.scale(cereals_df.iloc[:, 3:].dropna(axis=0)))
```

Normalize the variables

# PCA Output Using all 13 ***Normalized*** Numerical Variables

| | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 |
|---|---|---|---|---|---|---|
| Standard deviation | 1.9192 | 1.7864 | 1.3912 | 1.0166 | 1.0015 | 0.8555 |
| Proportion of variance | 0.2795 | 0.2422 | 0.1469 | 0.0784 | 0.0761 | 0.0555 |
| Cumulative proportion | 0.2795 | 0.5217 | 0.6685 | 0.7470 | 0.8231 | 0.8786 |

| | PC7 | PC8 | PC9 | PC10 | PC11 | PC12 | PC13 |
|---|---|---|---|---|---|---|---|
| Standard deviation | 0.8251 | 0.6496 | 0.5658 | 0.3051 | 0.2537 | 0.1399 | 0.0 |
| Proportion of variance | 0.0517 | 0.0320 | 0.0243 | 0.0071 | 0.0049 | 0.0015 | 0.0 |
| Cumulative proportion | 0.9303 | 0.9623 | 0.9866 | 0.9936 | 0.9985 | 1.0000 | 1.0 |

# Weightings for the First Five ***Normalized*** Components

| | PC1 | PC2 | PC3 | PC4 | PC5 |
|---|---|---|---|---|---|
| calories | -0.299542 | -0.393148 | 0.114857 | -0.204359 | 0.203899 |
| protein | 0.307356 | -0.165323 | 0.277282 | -0.300743 | 0.319749 |
| fat | -0.039915 | -0.345724 | -0.204890 | -0.186833 | 0.586893 |
| sodium | -0.183397 | -0.137221 | 0.389431 | -0.120337 | -0.338364 |
| fiber | 0.453490 | -0.179812 | 0.069766 | -0.039174 | -0.255119 |
| carbo | -0.192449 | 0.149448 | 0.562452 | -0.087835 | 0.182743 |
| sugars | -0.228068 | -0.351434 | -0.355405 | 0.022707 | -0.314872 |
| potass | 0.401964 | -0.300544 | 0.067620 | -0.090878 | -0.148360 |
| vitamins | -0.115980 | -0.172909 | 0.387859 | 0.604111 | -0.049287 |
| shelf | 0.171263 | -0.265050 | -0.001531 | 0.638879 | 0.329101 |
| weight | -0.050299 | -0.450309 | 0.247138 | -0.153429 | -0.221283 |
| cups | -0.294636 | 0.212248 | 0.140000 | -0.047489 | 0.120816 |
| rating | 0.438378 | 0.251539 | 0.181842 | -0.038316 | 0.057584 |

# PCA in Classification/Prediction

- Apply PCA to training data
- Decide how many PC's to use
- Use variable weights in those PC's with validation/new data
- This creates a new reduced set of predictors in validation/new data

# Regression-Based Dimension Reduction

- Multiple Linear Regression or Logistic Regression
- Use subset selection
- Algorithm chooses a subset of variables
- This procedure is integrated directly into the predictive task

# Summary

- **Data summarization** is an important component of data exploration
- **Data summaries** include numerical metrics (average, median, etc.) and graphical summaries
- **Data reduction** is useful for compressing the information in the data into a smaller subset
  - Categorical variables can be reduced by combining similar categories
  - Principal components analysis transforms an original set of numerical data into a smaller set of weighted averages of the original data that contain most of the original information in less variables.

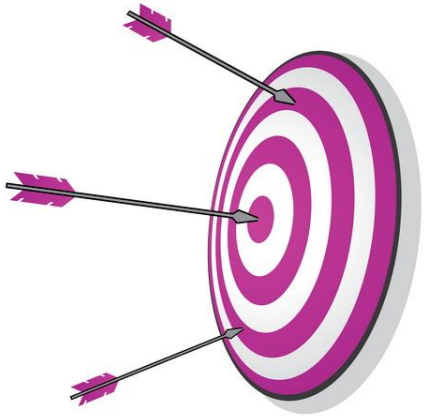
# Chapter 5 – Evaluating Classification & Predictive Performance

## Data Mining for Business Analytics in Python

**Shmueli, Bruce, Gedeck & Patel**

 rev 10/17/19

# Why Evaluate?



- Multiple methods are available to classify or predict
- For each method, multiple choices are available for settings
- To choose best model, need to assess each model's performance

# Evaluating Predictive Performance

# Outcomes of Interest

- Numerical value
- Membership in a class
- “Propensity” - probability of belonging to a class

# Measuring Predictive error - Numerical Value

- Not the same as “goodness-of-fit”
- We want to know how well the model predicts **new data**, not how well it fits the data it was trained with
- Key component of most measures is difference between actual *y* and predicted *y* (“error,” “loss”)

# Some measures of error

**MAE or MAD**: Mean absolute error (deviation)
Gives an idea of the magnitude of errors

**Average error**
Gives an idea of systematic over- or under-prediction

**MAPE**: Mean absolute percentage error

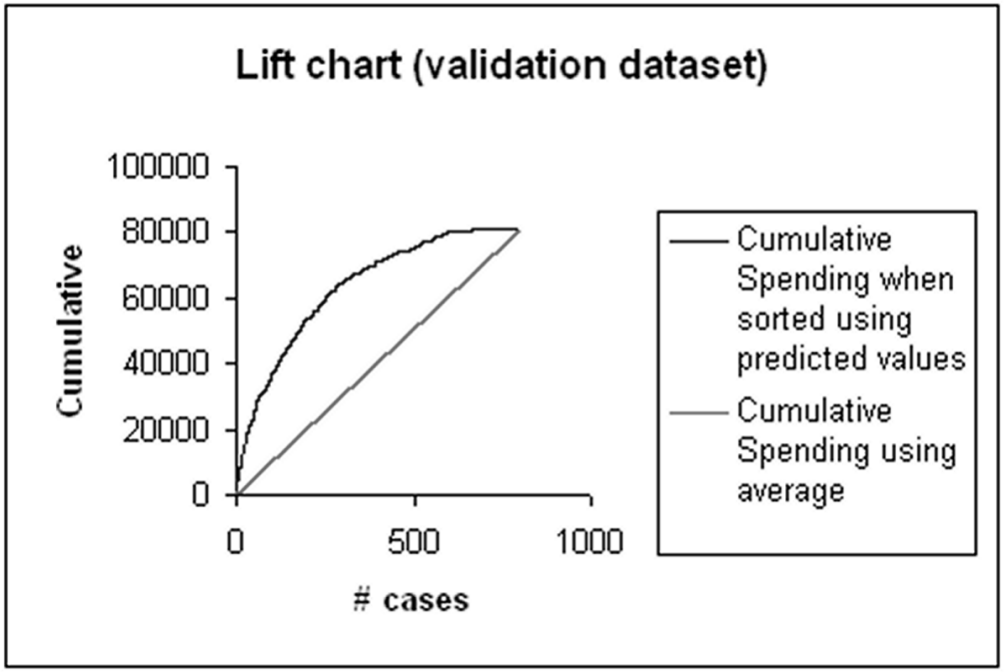
**RMSE** (root-mean-squared-error): Square the errors, find their average, take the square root

**Total SSE**: Total sum of squared errors

# Lift Chart for Predictive Error

- Y axis is cumulative value of numeric target variable (e.g., revenue), instead of cumulative count of “responses”
- X axis is cumulative number of cases, sorted left to right in order of predicted value
- Benchmark is average numeric value per record, i.e. not using model

# Lift chart example – spending



# Accuracy Measures (Classification)

# Misclassification error

- Error = classifying a record as belonging to one class when it belongs to another class.
- Error rate = percent of misclassified records out of the total records in the validation data

# Naïve Rule

**Naïve rule:** classify all records as belonging to the most prevalent class

- Often used as benchmark: we hope to do better than that
- Exception: when goal is to identify high-value but rare outcomes, we may do well by doing worse than the naïve rule (see “lift” – later)

# Separation of Records

“High separation of records” means that using predictor variables attains low error

“Low separation of records” means that using predictor variables does not improve much on naïve rule

# Confusion Matrix, 3000 cases

## function: "`classificationSummary`" found in Appendix

| | | Actual Class | |
|---|---|---|---|
| | | 0 | 1 |
| Predicted Class | 0 | 2689 | 85 |
| | 1 | 25 | 201 |

**201** 1's correctly classified as "1"

**85** 1's incorrectly classified as "0"

**25** 0's incorrectly classified as "1"

**2689** 0's correctly classified as "0"

# Error Rate

| | | Actual Class | |
|---|---|---|---|
| | | 0 | 1 |
| Predicted Class | 0 | 2689 | 85 |
| | 1 | 25 | 201 |

**Overall error rate** = (25+85)/3000 = 3.67%

**Accuracy** = 1 – err = (201+2689) = 96.33%

If there are multiple classes, the error rate is:

(sum of misclassified records)/(total records)

# Cutoff for classification

Most DM algorithms classify via a 2-step process:

For each record,

1. Compute **probability of belonging to class "1"**
2. Compare to cutoff value, and classify accordingly

- Default cutoff value is 0.50
  - If >= 0.50, classify as "1"
  - If < 0.50, classify as "0"
- Can use different cutoff values
- Typically, error rate is lowest for cutoff = 0.50

# When One Class is More Important

In many cases it is more important to identify members of one class

- Tax fraud
- Credit default
- Response to promotional offer
- Detecting electronic network intrusion
- Predicting delayed flights

In such cases, we are willing to tolerate greater overall error, in return for better identifying the important class for further attention

# Alternate Accuracy Measures

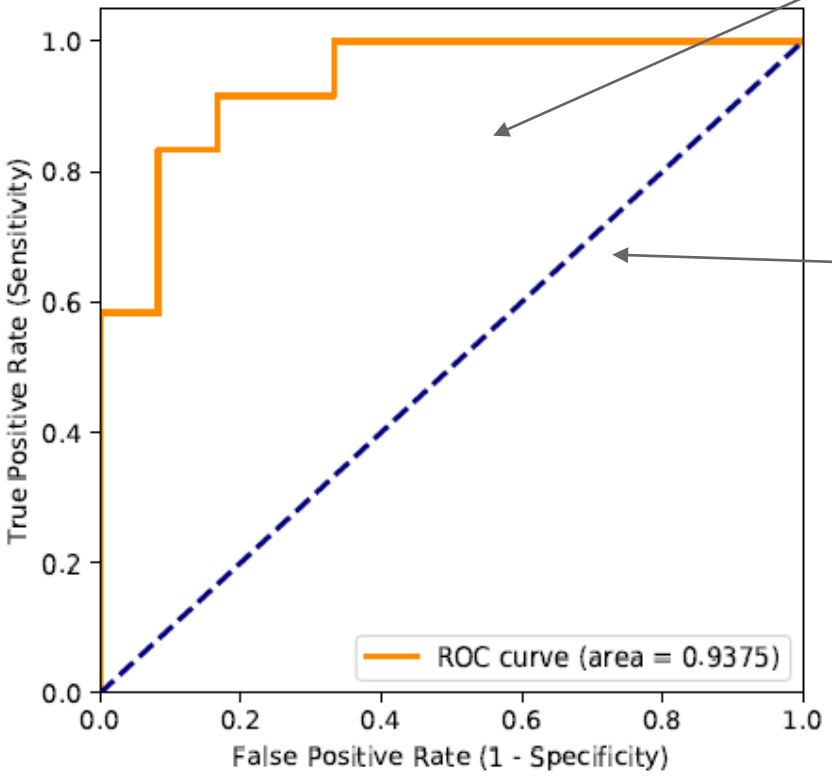
If "$C_1$" is the important class,

**Sensitivity (also called "recall)** = % of "$C_1$" class correctly classified

**Specificity** = % of "$C_0$" class correctly classified

**Precision**= % of predicted "$C_1$'s" that are actually"$C_1$'s"

# ROC Curve (library `roc_curve`)




Area under the ROC curve ("AUC," function `auc`) is a useful metric

Diagonal reflects random classification on a sliding probability, from 0 to 1, of labeling a record as a "1"

Other software sometimes plots x-axis as "specificity" with 1 on left, 0 on right.

# ROC = “Receiver Operating Characteristics”

Term originated in WWII, applied to radar

US sought to measure effectiveness in identifying enemy aircraft, i.e. distinguishing signal from noise

**ROC's** are one way to measure a model's effectiveness in separating the "wheat from the chaff"

"**Lift**" ("**gains**") is a similar metric, but measuring "how much does the model improve on random chance in finding the class of interest"

# Lift (also termed “gains”): Goal

Evaluates how well a model identifies the most important class

Helps evaluate, e.g.,

- How many tax records to examine
- How many loans to grant
- How many customers to mail offer to

# Lift (gains) and Decile Charts – Cont.

Compare performance of DM model to “no model, pick randomly”

Measures ability of DM model to identify the important class, relative to the average prevalence of the class

Charts give explicit assessment of results over a large number of cutoffs

# Lift and Decile Charts: How to Use

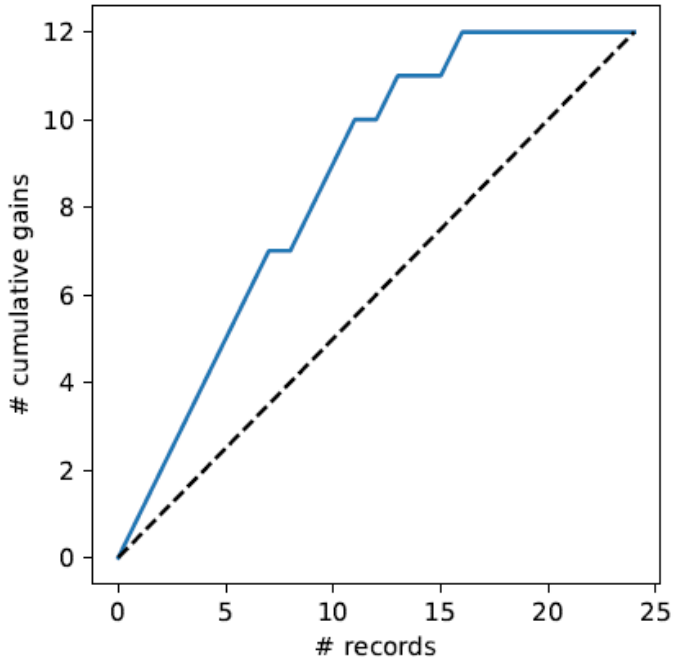
Sort records by predicted probability of belonging to the important class ("1's")

Move down the list, noting actual class

As you go, compare the number of actual 1's to the number of 1's you would expect with no model

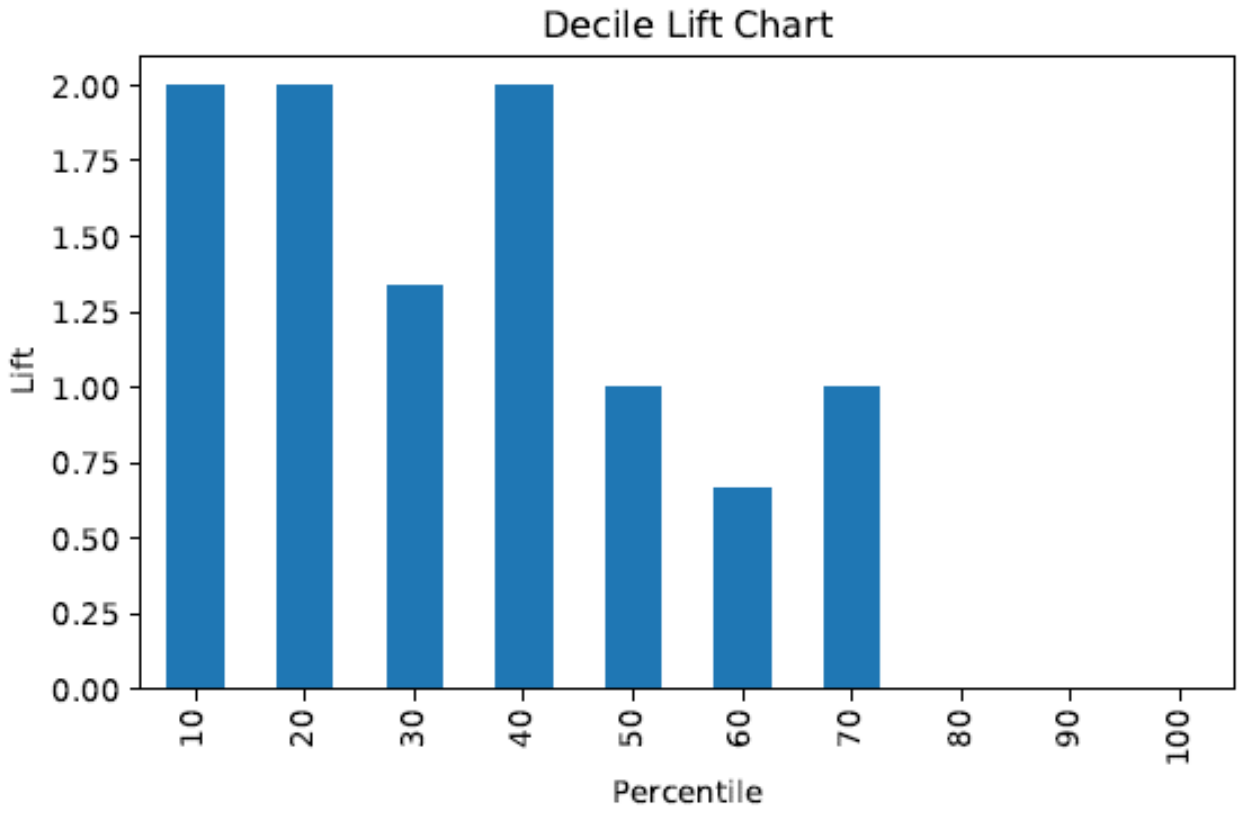
- In lift chart: compare step function to straight line
- In decile chart compare to ratio of 1

# Lift (Gains) Chart – cumulative performance




After examining (e.g.) 10 cases (x-axis), 9 owners (y-axis) have been correctly identified

# Decile Chart




In “most probable” (top) decile, the model is twice as likely to identify the important class compared to just picking at random.

# Lift (Gains): How to Compute

- Using the model's output, sort records from most likely to least likely members of the important class

- Compute lift: Accumulate the correctly classified "important class" records (Y axis) and compare to number of total records (X axis)

# Lift (gains) curve vs. Decile Charts

Both embody concept of “moving down” through the records, starting with the most probable 1’s

Decile chart does this in decile chunks of data

- Y axis shows ratio of decile mean to overall mean

Lift chart shows continuous cumulative results

- Y axis shows number of important class records identified

# Lift (gains) vs. ROC curve

Gains and ROC curves have similar appearance

ROC curve and AUC provide a widely used single metric and visual to assess a model's ability to separate classes **overall**

Lift (gains) yields intuitive measure of model **performance at different cutoffs**, but no overall metric

# Asymmetric Costs

# Misclassification Costs May Differ

The cost of making a misclassification error may be higher for one class than the other(s)

Looked at another way, the benefit of making a correct classification may be higher for one class than the other(s)

# Example – Response to Promotional Offer

Suppose we send an offer to 1000 people, with 1% average response rate ("1" = response, "0" = nonresponse)

- "Naïve rule" (classify everyone as "0") has error rate of 1% (seems good)
- Using DM we can correctly classify eight 1's as 1's

  It comes at the cost of misclassifying twenty 0's as 1's and two 1's as 0's.

# Example (cont.)

## The Confusion Matrix*

| | Actual 0 | Actual 1 |
|---|---|---|
| Predicted 0 | 970 | 2 |
| Predicted 1 | 20 | 8 |

Error rate = (2+20) = 2.2% (higher than naïve rate)

*confusion matrix is often shown with predictions as rows, actuals as columns, the reverse of what Python produces

# Introducing Costs & Benefits

**Suppose:**

- Profit from a “1” is $10
- Cost of sending offer is $1

**Then:**

- Under naïve rule, all are classified as “0”, so no offers are sent: no cost, no profit
- Under DM predictions, 28 offers are sent.
  - 8 respond with profit of $10 each
  - 20 fail to respond, cost $1 each
  - 972 receive nothing (no cost, no profit)
- Net profit = $60

# Profit Matrix

| | Actual 0 | Actual 1 |
|---|---|---|
| Predicted 0 | $0 | $0 |
| Predicted 1 | ($20) | $80 |

# Lift (again)

Adding costs to the mix, as above, does not change the actual classifications

Better: Use the lift curve and change the cutoff value for “1” to maximize profit

# Generalize to Cost Ratio

Sometimes actual costs and benefits are hard to estimate

- Need to express everything in terms of costs (i.e., cost of misclassification per record)
- Goal is to minimize the average cost per record

A good practical substitute for individual costs is the **ratio** of misclassification costs (e,g,, “misclassifying fraudulent firms is 5 times worse than misclassifying solvent firms”)

# Minimizing Cost Ratio

$q_1$ = cost of misclassifying an actual “1”,

$q_0$ = cost of misclassifying an actual “0”

Minimizing the **cost ratio** $q_1/q_0$ is identical to minimizing the average cost per record

Software may provide option for user to specify cost ratio

# Note: Opportunity costs

- As we see, best to convert everything to costs, as opposed to a mix of costs and benefits
- E.g., instead of “benefit from sale” refer to “opportunity cost of lost sale”
- Leads to same decisions, but referring only to costs allows greater applicability

# Cost Matrix
# (inc. opportunity costs)

| | Predict as 1 | Predict as 0 |
|---|---|---|
| Actual 1 | $8 | $20 |
| Actual 0 | $20 | $0 |

**Recall original confusion matrix (profit from a "1" = $10, cost of sending offer = $1):**

| | Predict as 1 | Predict as 0 |
|---|---|---|
| Actual 1 | 8 | 2 |
| Actual 0 | 20 | 970 |

# Multiple Classes

For *m* classes, confusion matrix has *m* rows and *m* columns

- Theoretically, there are *m*(*m*-1) misclassification costs, since any case could be misclassified in *m*-1 ways

- Practically, too many to work with

- In decision-making context, though, such complexity rarely arises – one class is usually of primary interest

# Adding Cost/Benefit to Lift Curve

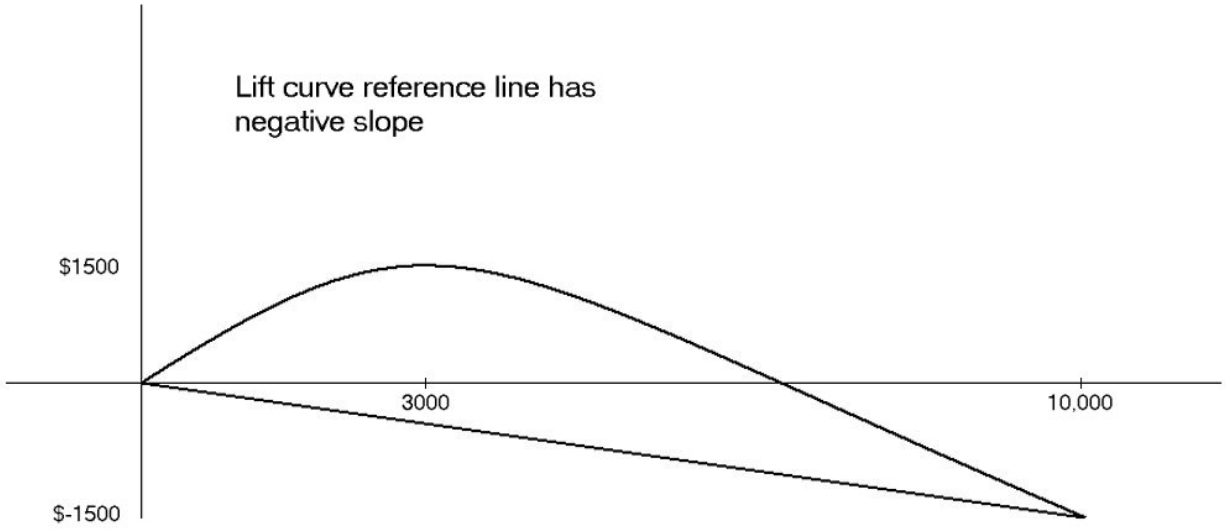
- Sort records in descending probability of success
- For each case, record cost/benefit of actual outcome
- Also record cumulative cost/benefit
- Plot all records
  - X-axis is index number (1 for 1$^{st}$ case, n for n$^{th}$ case)
  - Y-axis is cumulative cost/benefit
  - Reference line from origin to $y_n$ ($y_n$ = total net benefit)

# Lift Curve May Go Negative

If total net benefit from all cases is negative, reference line will have **negative slope**

Nonetheless, goal is still to use cutoff to select the point where net benefit is at a maximum

# Negative slope to reference curve



# Oversampling and Asymmetric Costs

# Rare Cases

Asymmetric costs/benefits typically go hand in hand with presence of rare but important class

- Responder to mailing
- Someone who commits fraud
- Debt defaulter

- Often we oversample rare cases to give model more information to work with

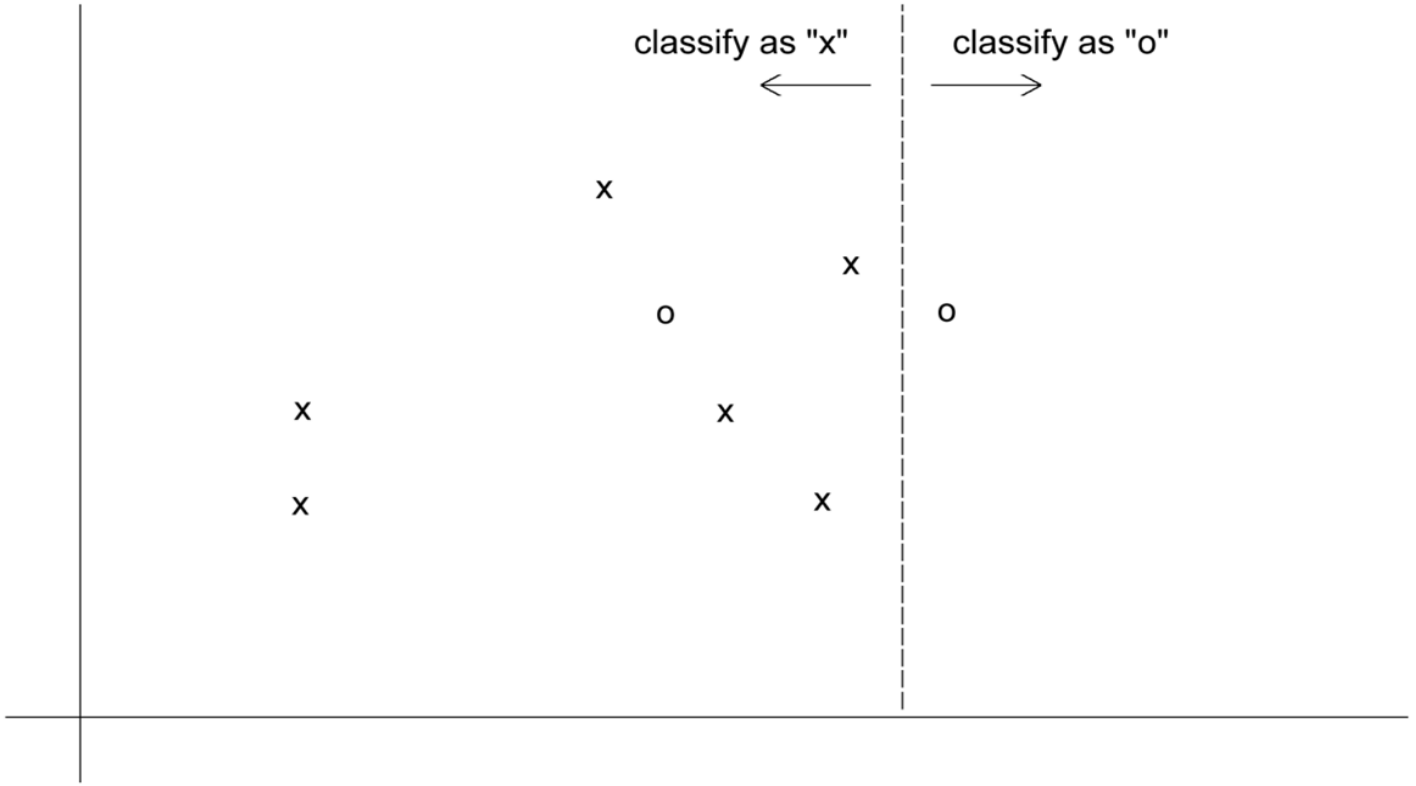
- Typically use 50% “1” and 50% “0” for training

# Example

Following graphs show optimal classification under three scenarios:

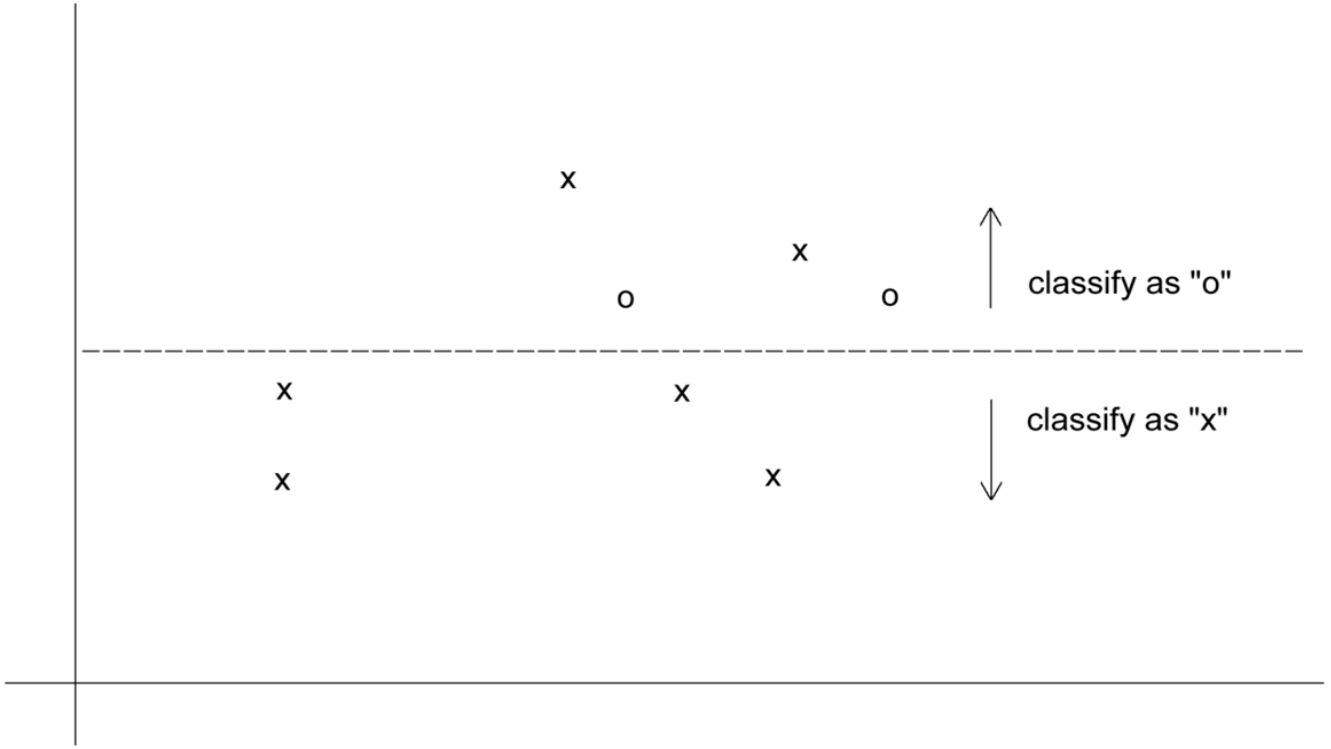
- assuming equal costs of misclassification
- assuming that misclassifying “o” is five times the cost of misclassifying “x”
- Oversampling scheme allowing DM methods to incorporate asymmetric costs

# Classification: equal costs



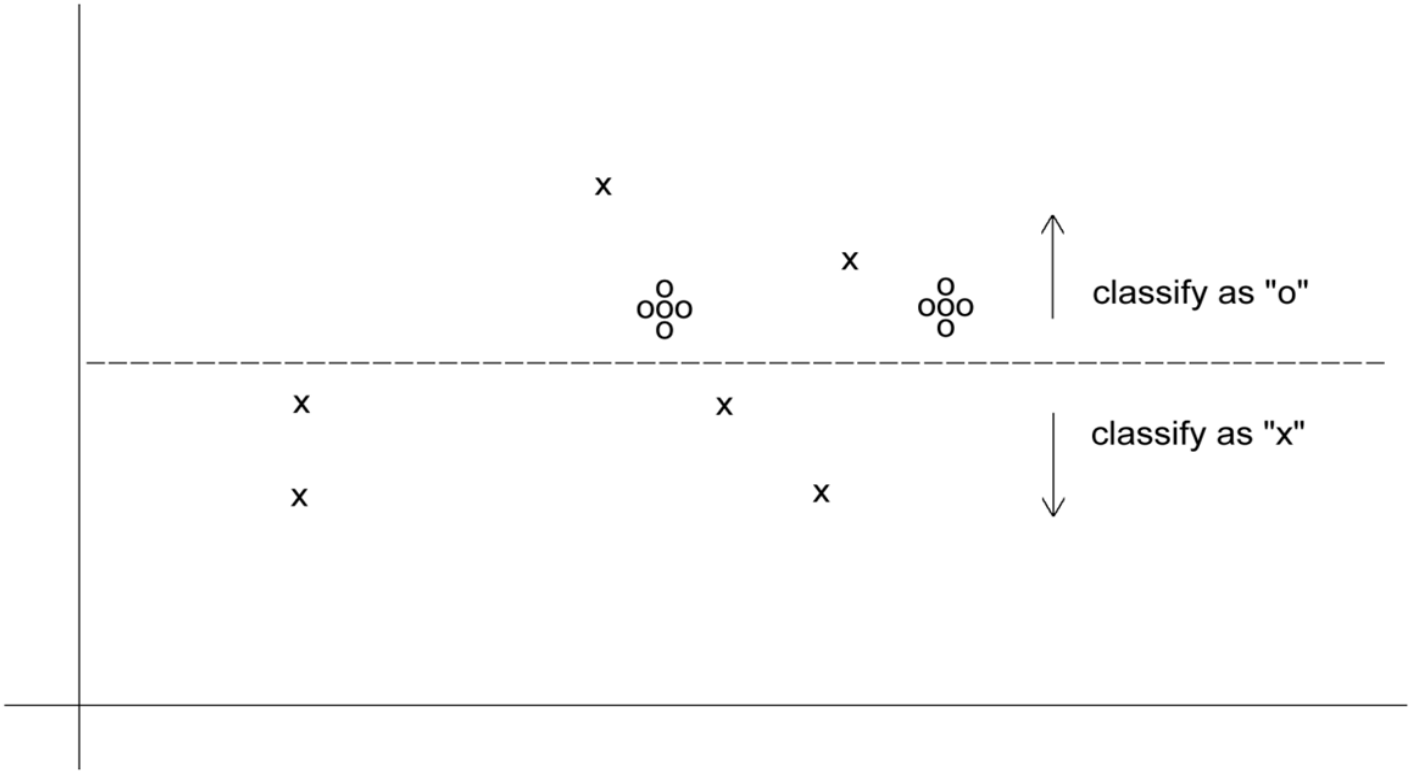
# Classification: Unequal costs

Suppose misclassifying “o” 5 times as costly as misclassifying “x”



# Oversampling Scheme

Oversample “o” to appropriately weight misclassification costs



# An Oversampling Procedure

1. Separate the responders (rare) from non-responders
2. Randomly assign half the responders to the training sample, plus equal number of non-responders
3. Remaining responders go to validation sample
4. Add non-responders to validation data, to maintain original ratio of responders to non-responders
5. Randomly take test set (if needed) from validation

# Classification Using Triage

Take into account a gray area in making classification decisions

- Instead of classifying as $C_1$ or $C_0$, we classify as
  - $C_1$
  - $C_0$
  - Can't say

The third category might receive special human review

# Summary

- Evaluation metrics are important for comparing across DM models, for choosing the right configuration of a specific DM model, and for comparing to the baseline (“no model”)
- Major metrics: confusion matrix, error rate, predictive error
- Other metrics when
  - one class is more important
  - asymmetric costs
- When important class is rare, use oversampling
- In all cases, metrics computed from validation data