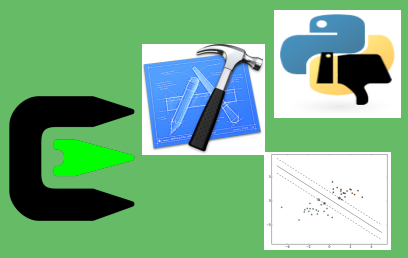
CISC 5352 FINANCIAL PROGRAMMING AND DATA ANALYTICS LECTURE NOTE (8)



Henry Han Ph.D.
Department of Computer and Information Science
Fordham University, New York NY 10023

Last class review

1. SVM learning (**from** sklearn **import** svm)
2. It has a very clear geometric interpretations:
   1. Optimal hyperplane separate data points
   2. support vectors are decision markers on the boundary
3. Kernel tricks in SVM: less work but do computing in feature-space (high dimensional space)
   1. Kernel matrices should be positive definite or semi-positive definite
   2. 'linear', 'rbf', 'poly', 'sigmoid', 'precomputed'
   3. Different kernels same label usually lead to different support vectors!

Q1: Which kernels should we use in classification/ regression?

Q1: Which kernels should we use in classification/ regression?

It will rely on your data.

Finance data: 'rbf', 'linear' ('poly' may need more time!)

Image, cybersecurity malware data: 'rbf' (very good performance)

Omics data (high-dimensional data): 'linear'

You can also make your own kernel.

Q2: Why those kernels work well for the data?

Q2: Why those kernels work well for the data?

The map created by the kernel is more similar to the implicit decision function f(x) that assigns the label for a sample x

Q3: which issue have you met in using SVM in your project?

Q2: which issue have you met in using SVM in your project?

SVM does not scale well for large datasets.

Say you have a training data 10,000x7 from option data you have

10,000 samples, each of which has 7 variables: strike price, current price,…implied volatility

SVM will have a 10000×10000 kernel matrix!

SVM with10000×10000 kernel matrix

It will ask a buge storage: suppose each entry is a double type: it will need 8×10^8 bytes almost 1 GB in storage and $10^8/2$ times kernel function evaluations!

It will make SVM running very slow! Not a good choice for online learning!

SVM with10000×10000 kernel matrix

It will ask a buge storage: suppose each entry is a double type: it will need 8×10^8 bytes almost 1 GB in storage and $10^8/2$ times kernel function evaluations!

It will make SVM running very slow! Not a good choice for online learning!

The time and space complexities both are $O(n^2)$ but n is large (e.g., n=10000)→ SVM is not desirable for large datasets!

Note: k-NN will not have this issue because the nearest-neighbor search is linear: O(n)

Idea: do sampling from training data to build an approximated kernel matrix (smaller) to save complexity

① Kernel trick actually implements an implicit map from input space to feature space.

② But it cannot scale well for large datasets

③ So idea is to build an explicit map from input space to feature space to approximate the 'original implicit map'

④ That is to approximate the kernel matrix, which represents the 'original implicit map' . A relatively new technique 'matured' around 2012

The 'original implicit map' represented by the kernel matrix is called 'kernel map'

Almost all the kernel scaling methods of SVM is based on building an explicit kernel map approximately to save complexity generated by large datasets.

An example: approximate 'rbf' kernel using Fourier transform

'rbf' kernel $\exp(-\gamma|x - x'|^2)$.

RBFSampler (Random Kitchen Sinks)→ build an explicit kernel map

SVM will do classification/regression "really" in the feature space!

It works well for large datasets, because the kernel map built from a sampling approach instead of using all data

RBFSampler(gamma=1.0, n_components=100, random_state=None)

n_components: # Monte Carlo samples per original feature

The general scaling optimization in SVM: SGDClassifier + kernel map approximation

SGDClasssifer: a linear SVM classifier with SGD (stochastic gradient descent learning (SGD)

It uses an iteration method (stochastic gradient descent learning) to find w and b of the optimal hyperplane.

It is good for online learning

An example to transform data to R^{n_d} space for classification (upper-sampling)

```
from sklearn import svm
from sklearn.kernel_approximation import RBFSampler
from sklearn.linear_model import SGDClassifier
import time

## training data
X = [[0, 0], [10,10], [10, 0], [0,10]]
## training data label
y = [0,0,1,1]
## test data
test_data = [[2.5, 8.]]

# set rbf_sample object: map each sample to R^n_d space
n_d = 50
rbf_feature = RBFSampler(n_components=n_d, gamma=1/2, random_state=1)
print(rbf_feature)
time.sleep(2)
```

```
## transform training and test data to feature space explicitly

X_features    = rbf_feature.fit_transform(X)
test_features = rbf_feature.fit_transform(test_data)

for i in range(0,len(X_features)):
    print("feature   "+ "{:d}".format(i) +" in feature space")
    print(str(X_features[i]),"\n"+"***")
    time.sleep(2)

print("The transformed test data in feature space ^^\n")
print(str(test_features) +  "\n")
time.sleep(2)
```

```
## use SGDClasssifer: a linear SVM classifier with
## SGD (stochastic gradient descent learning (SGD)
## It is good for online learning

clf=SGDClassifier()
clf.fit(X_features, y)  ## training

## prediction for test data in the feature space
predicted_label=clf.predict(test_features)

print("\n The predicted label is---->" +str(predicted_label))
time.sleep(2)
```

Change n_d=100, 200, 1000 and check your output!

Change n_d=100, 200, 500,1000 and check your output!

Output will change!

It is an approximate approach only instead of a robust accurate way!

```
kernel_list=['linear', 'rbf']

# print Input_train

for kernel in kernel_list:
    print("\n Wait, SVM  results under kernel: " + str(kernel)+ "\n")

    t = svm.SVR(kernel=kernel, cache_size=500)
    t.fit(Input_train, Response_train)

    ##predicted implied votatility
    predictedSVMIV = t.predict(Input_test)


    Error_SVM = [None] * len(Input_test)
    Error_SVM   = abs(Response_test - predictedSVMIV)

# Model Evaluation    print('\n\nThe SVM Model Peformance Summary as follows:')
    print('The MSE is            {:20.16f}'.format(get_MSE(Error_SVM)))
    print('The mean error is    {:20.16f}'.format(np.mean(Error_SVM)))
    print('The maximum error is {:20.16f}'.format(max(Error_SVM)))
    print('The minimum error is {:20.16f}'.format(min(Error_SVM)))
```

Sample codes for SVM implied volatility prediction. 'poly can lead to very slow performance

See program: cisc5352.lecture.8.demoKNNSVMIVPrecition.py

Update cisc5352.lecture.8.demoKNNSVMIVPrecition.py such that it can do kernel map approximation (Quiz 6)

You don't need to replace svr by SGDRegressor

More about SGDRegressor:

http://scikit-learn.org/stable/modules/generatedsklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor

Cross-validation methods: data partition methods to evaluate (validate) classifiers

1. hold-out cross-validation:
   1. Randomly partition a dataset with a hold-out percentage (e.g., 60% training 40% test. Usually it needs many trials)

Cross-validation methods: data partition methods to evaluate (validate) classifiers

1. hold-out cross-validation:
   1. Randomly partition a dataset with a hold-out percentage (e.g., 60% training 40% test. Usually it needs many trials)
2. K-fold cross-validation
   1. Partition a dataset into k-folds
   2. training data: k-1 folds and tst data the left→ Proceeding until all folds are used as test data

Cross-validation methods: data partition methods to evaluate (validate) classifiers

1. hold-out cross-validation:
   1. Randomly partition a dataset with a hold-out percentage (e.g., 60% training 40% test. Usually it needs many trials)
2. K-fold cross-validation
   1. Partition a dataset into k-folds
   2. training data: k-1 folds and tst data the left→ Proceeding until all folds are used as test data
3. Leave-one-out cross-validation (LOOCV)
   1. One sample is used as test data and other data used as training→ Proceeding until all samples are used as test data
   2.

```
from sklearn import datasets
from sklearn.model_selection import cross_val_score
from sklearn import svm

iris  = datasets.load_iris()
data  = iris.data
label = iris.target

kernel='rbf';
clf = svm.SVC(kernel=kernel, gamma=0.5, C=1)

k=10  # total folders
scores = cross_val_score(clf, data, label, cv=k)
print("\n k={:d}".format(k) + "  fold cross validation\n")
print("SVM classification under  kernel ** " + str(kernel) + "  ** is:\n\n" +
str(scores))
```

Demo k-fold for SVM

Independent test set method is widely used in business analytics

Find an independent test dataset to evaluate the performance of a machine learning.

As we did in our project.

Gradient Boosting (GB)

1. The 'new' state-of-the art in machine learning, invented in 1999
2. A novel ensemble learning algorithm
   1. Ensemble learning: melding results from many weak learners into one high-quality ensemble predictor
   2. Decision tree is a typical ensemble learning algorithm, in which each tree is a weak learner. An ensemble learning algorithm group and optimize results from many weak learners to get better prediction.

Gradient Boosting (GB) Cont'd

① Its main idea is to estimate a prediction function f(x) from training data with a group of base (weak) learners by searching the negative gradient directions of its loss function

① Loss function: the objective function (e.g. $L(f(x),y)=\log(f(x)-y)^2$) to be optimized: f(x) is the prediction function (unknown).

② Gradient-boosting restricts the function search space to a parametric family of functions f(x, θ) and the search direction is along the negative gradient descent direction of the loss function starting from the base learners specified by function h(x, θ) .

Gradient Boosting (GB) Cont'd

prediction function estimation in GB, in which h(..) represents The decision function from weak learners

$$\hat{f}_k = \hat{f}_{k-1} + \tau_k h(x, \theta_k)$$

$\hat{f}(x)$ is the estimation of the decision function f(x)

$$(\tau_k, \theta_k) = \arg\min_{\tau,\theta} \sum_{i=1}^{m} [-g_k(x_i) + \tau h(x_i, \theta)]^2$$

The gradient descent direction for x_i is

$$g_k(x_i) = \frac{\partial L}{\partial f}\Big|_{f(x)=\sum_{j=0}^{k} \hat{f}_j(x_i)}$$

Base learners are decision trees

① Theoretically, the base learners can be chosen as different linear regression models, decision trees or even some customized models according to different learning needs.

② However, decision trees are chosen as the base learners in most gradient boosting methods for its simplicity and advantage in modeling the interactions between predictor variables

Gradient Boosting (GB) parameters

GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2)

loss function to be optimized: loss : {'ls', 'lad', 'huber', 'quantile'}

n_estimators: # trees

max_depth : max depth of each tree

min_samples_split: The minimum number of samples required to split an internal node:

subsample: The fraction of samples to be used for fitting the individual base learners.

Gradient Boosting (GB) regression steps

① params = {'n_estimators': 1000, 'max_depth': 6, 'min_samples_split': 2, 'learning_rate': 0.01, 'loss': 'ls'}

② gb = ensemble.GradientBoostingRegressor(**params)

③ gb.fit(train_data, train_data_label)

④ gb.predict(test_data)

Update cisc5352.lecture.8.demoKNNSVMIVPrecition.py such that it can do implied volatility prediction by GB regression

```
print("GB regression implied votatility prediction\n")
params = {'n_estimators': 2000, 'max_depth': 6, 'min_samples_split': 2,
        'learning_rate': 0.01}
gb = ensemble.GradientBoostingRegressor(**params)

gb.fit(Input_train, Response_train)

Error_GB     = [None] * len(Input_test)
predictedGBIV =  Error_GB

predictedGBIV = gb.predict(Input_test)
Error_GB     = abs(Response_test - predictedGBIV)

method = 'GB regression'
print("\n\nThe' + method +' Model Peformance Summary as follows:')
print("The MSE is          {:20.16f}".format(get_MSE(Error_GB)))
print("The mean error is    {:20.16f}".format(np.mean(Error_GB)))
print("The maximum error is {:20.16f}".format(max(Error_GB)))
print("The minimum error is {:20.16f}".format(min(Error_GB)))
```

GB for implied volatility prediction

Random forest (RF) trees

1. Random forests (*RForest*) build a group of unpruned decision trees with the bagging technique
   1. Bagging constructs a large number of trees with bootstrap samples from a dataset
      1. Bootstrap: view data as a population and do random sampling with replacement. This is a way to get extra samples when dataset itself is small
2. It employs a 'feature bagging' technique, which randomly selects a subset of input variables instead of all variables to build trees in training, to lower the variances of the loss function for the sake of generalization.
   1. That is, The final predictor ensemble is constructed in a randomly selected subspaces of training data.

Given training data $x_1, x_2 \ldots x_m$, and their labels y_i in {-1,1} random forests compute an ensemble prediction function by aggregating the prediction functions of $|B| \sim 10^3$ bags, which are constructed by conducting $|B|$ times of bootstraps from the training data,

$$\hat{f}(x) = \frac{1}{|B|} \sum_{k=1}^{|B|} \hat{f}_k(x)$$

Each prediction function

$\hat{f}_k(x),\ k = 1, 2, \cdots |B|$ is fitted with a randomly selected training sample set

Random forest major parameters

sklearn.ensemble.RandomForestRegressor(n_estimators=10, max_depth=6, min_samples_split=2)

n_estimators: # trees in each forest

min_samples_split: The minimum number of samples required to split an internal node:

max_depth : max depth of each tree: default None

Default: nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples

Random forest (RF) regression steps

from sklearn.ensemble import RandomForestRegressor

① params = {'n_estimators': 20, 'min_samples_split': 2}

② rf = ensemble.GradientBoostingRegressor(**params)

③ rf.fit(train_data, train_data_response)

④ rf.predict(test_data)

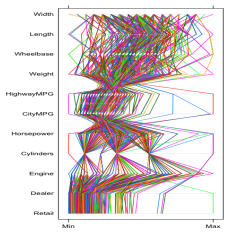
Update cisc5352.lecture.8.demoKNNSVMIVPrecition.py
such that it can do implied volatility prediction by RF regression

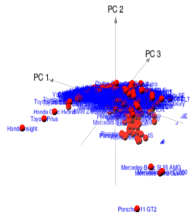
I am not going to lecture Random neural nets and deep learning

However, they will appear in your projects/homework.

Deep learning is a little bit challenging

PCA: dimension reduction and visualization tools in data analytics





PCA is a linear dimension reduction method

- It projects input data onto a lower-dimensional space that **preserves as much data variances as possible**

PCA is a linear dimension reduction method

- It projects input data to onto **a lower-dimensional space** that **preserves as much data variances as possible**
- **The lower-dimensional space is just a new coordinate system**

PCA is a linear dimension reduction method

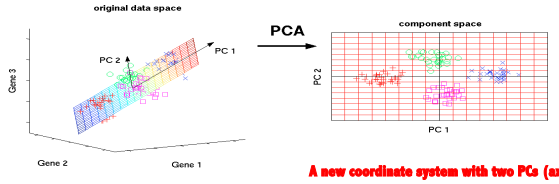
- It projects input data to onto a lower-dimensional space that **preserves as much data variances as possible**
- **The lower-dimensional space is just a new coordinate system**
  1. Its each "axis" is called **a principal component** (PC), which is the linear combination of the original variables.

PCA is a linear dimension reduction method

- It projects input data to onto a lower-dimensional space that **preserves as much data variances as possible**
- **The lower-dimensional space is just a new coordinate system**
  1. Its each "axis" is called **a principal component** (PC), which is the linear combination of the original variables.
  2. **All principal components (axes) are orthogonal to each other (No redundant information!)**

PCA is a linear dimension reduction method

- It projects input data to onto a lower-dimensional space that **preserves as much data variances as possible**
- **The lower-dimensional space is just a new coordinate system**
  1. Its each "axis" is called **a principal component** (PC), which is the linear combination of the original variables.
  2. **All principal components (axes) are orthogonal to each other (No redundant information!)**



A new coordinate system with two PCs (axes)

It is the "root" of the following more powerful data analytics methods

1. **Independent component analysis (ICA)** (Hyvärinen, 1999)
2. **Probabilistic principal component analysis (PPCA)** (Tipping and Bishop, 1999)
3. **Nonnegative matrix factorization (NMF)** (Lee and Seung, 2000)
4. **Kernel principal component analysis (KPCA)** (Schölkopf, 1999)
5. **Sparse principal component analysis (SPCA)** (Zou et al, 2004)
6. **Nonnegative principal component analysis (NPCA)** (Han, 2010)
7. **Derivative component analysis (DCA)** (Han, 2014)

Applications of Principal Component Analysis in Data analytics

Business

- **Trading**
- **Portfolio-ranking**
- **credit card ranking,**
- **security forecast**
- **CRM analysis**
- **Business intelligence**
- **Marketing**

Bioinformatics and health informatics

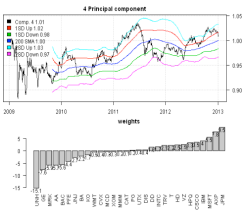
Data mining and machine learning

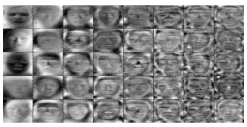
Face recognition

Social media computing

…

Almost all fields in data analytics!





What we have: input data

- Our input data is modeled as a nxp matrix:
  - **Each row is an observation** (a sample, (e.g. a stock))
  - **Each column is a variable** (feature)

What we have: input data

- Our input data is modeled as a nxp matrix:
  - **Each row is an observation** (a sample, (e.g. a stock))
  - **Each column is a variable** (feature)
- **What do we want to do by using PCA?**
  - **We want to decrease number of variables (dimensions) in a new coordinate system to get reduced-data to keep original data variances as much as possible!**

What we have: input data

- Our input data is modeled as a nxp matrix:
  - **Each row is an observation** (a sample)
  - **Each column is a variable** (feature)
- **What do we want to do by using PCA?**
  - **We want to decrease number of variables (dimensions) in a new coordinate system to get reduced-data to keep original data variances as much as possible!**
  - **We want to use the reduced-data in the new coordinate system to represent the original data.**

What we have: input data

- Our input data is modeled as a nxp matrix:
  - **Each row is an observation** (a sample)
  - **Each column is a variable** (feature)
- **What do we want to do by using PCA?**
  - **We want to decrease number of variables (dimensions) in a new coordinate system to get reduced-data to keep original data variances as much as possible!**
  - **We want to use the reduced-data in the new coordinate system to represent the original data.**
  - **We want to find more hidden information from such a dimension-reduction (e.g. outliers)**

What we have: input data

- Our input data is modeled as a nxp matrix:
  - **Each row is an observation** (a sample)
  - **Each column is a variable** (feature)
- **What do we want to do by using PCA?**
  - **We want to decrease number of variables (dimensions) in a new coordinate system to get reduced-data to keep original data variances as much as possible!**
  - **We want to use the reduced-data in the new coordinate system to represent the original data.**
  - **We want to find more hidden information from such a dimension-reduction (e.g. outliers)**
  - **We want to further do other data analytics (e.g. clustering) for the reduced-data!**

Use one sentence to summarize PCA

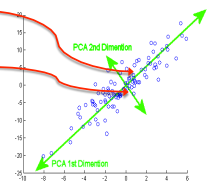
- **PCA gives you a reduced-data of the original data in a new coordinate system by keeping most of data variance information.**
- **PCA will at least give you three output items for a given input data**
  1. **Loadings**
  2. **Variances**
  3. **Scores**

1. Loadings : principal component (PC) matrix (new coordinate system)

Each column is a principal component → an axis of the new coordinate system

- The 1st PC is the 1st largest variance direction
- The 2nd PC is the 2nd largest variance direction
- The 3rd PC is the 3rd largest variance direction
- …

…



Note: we can prove it rigorously from a mathematical viewpoint.

2. Variances

The variance value for each principal component

① The 1st PC has the largest variance value (e.g., 100.1)
② The 2nd PC has the 2nd largest variance value
③ …

3. Scores

The coordinate values of the original data in the new coordinate system (new data)

Given a data set with n observations and p variables (a nxp matrix), How many PCs will we get at most?

Given a data set n observations and p variables (a nxp matrix), How many PCs will we get at most?

1. p number of PCs→ our data has *p* dimensions!
2. In other words, this new coordinate system has at most *p* axes!
3. The PC matrix (loadings) is a pxp matrix

All "mathematical steps" behind PCA

1. **Calculate the covariance matrix of input data→ get data variance information**
2. **Do eigenvalue decomposition for the covariance matrix to get** principal components (PCs)→seeking maximum variance directions
3. **Decide which dimensions to keep in the new coordinate system**

```
# Import packages
import numpy as np
from sklearn.decomposition import PCA

X = np.array([[-100, -1], [-200, -1], [-300, -2], [1, 100], [21, -1], [8.3,
9.92]])

print("\n Input data")
print(X)

## set PCA object
pca = PCA(n_components=2)
## conduct PCA
pca.fit(X)

print("\n explained variance in each PC \n")
print(str(pca.explained_variance_)+"\n")

print("explained variance ratios for all PCs\n")
print(str(pca.explained_variance_ratio_) + "\n")

print("PC components\n")
print(pca.components_)
```

Two major functions in R for PCA: they use different mathematical approaches to compute PCs

princomp()

prcomp()

The former is more basic and the latter is more robust

We can query details of prcomp from R prompt by typing

```
>?prcomp
```

Two major functions in R for PCA: they use different mathematical approaches to compute PCs

princomp()

prcomp()

The former is more basic and the latter is more robust

We can query details of prcomp from R prompt by typing

```
>?prcomp
```

prcomp {stats} R Documentation

Principal Components Analysis

Description

Performs a principal components analysis on the given data matrix and returns the results as an object of class `prcomp`.

Usage

```
prcomp(x, ...)

## S3 method for class 'formula'
prcomp(formula, data = NULL, subset, na.action, ...)

## Default S3 method:
prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE,
       tol = NULL, ...)

## S3 method for class 'prcomp'
predict(object, newdata, ...)
```

Arguments

prcomp(data, center, scale.)

Basic input:

- data: a nxp matrix (**n observations and p variables**)
- **center: if we need to center data to zero-mean data (TRUE/FALSE)**
- **scale.**: if we need to scale data to unit variance data (TRUE/FALSE)

Basic output:

- **sdev**: (square root of **variances**!)
  - the standard deviations of the principal components
- **rotation**: **loadings (PC matrix)**
- **x**: **scores** (new data in the new coordinate system)

A hands-on PCA analysis example

Now, you are a manger of a car sales company.

You collected the following information about cars in your company

1. Model: SUV, minivan, sports, pickup, …
2. Prices: dealer/retail prices
3. Gas mileage: cityMPG, highwayMPG,
4. Power (engine size, horsepower, #cylinder)
5. Size (width, length…)
6. Weight

You want to know:

1. how to remove some redundant variables→ dimension-reduction?
2. what are the relationships (correlation) between these variables?
3. what kinds of cars are very special (outliers) with respect to others?
4. What kinds of cars will be "naturally grouped together"?
5. Other following data analysis before you do a decision making
6. …

Vehicles data: 387 observations and 18 variables

- It consists of 387 vehicles from different models produced in 2004.
- Each vehicle has 18 variables: the first 7 variables describe the types of cars and the last 11 variables describe Retail, dealer prices, Engine size, Cylinder#, weight…
- "Sports" "SUV" "Wagon" "Minivan" "Pickup" "AWD" "RWD"
  - AWD→all wheel drive
  - RWD→ rear wheel drive
- "Retail" "Dealer" "Engine" "Cylinders" "Horsepower" "CityMPG" "HighwayMPG" "Weight" "Wheelbase" "Length" "Width"
- Only last 11 variables are numerical types
  - We have n=387 and p=11

The first three rows of data

	Sports	SUV	Wagon	Minivan	Pickup	AWD	RWD	Retail	Dealer	Engine	Cylinders	Horsepower	CityMPG	HighwayMPG	Weight	Wheelbase	Length	Width
Acura 3.5 RL	0	0	0	0	0	0	0	43755	39014	3.5	6	225	18	24	3880	115	197	72
Acura 3.5 RL Navigation	0	0	0	0	0	0	0	46100	41100	3.5	6	225	18	24	3893	115	197	72
Acura MDX	0	1	0	0	0	1	0	36945	33337	3.5	6	265	17	23	4451	106	189	77

We only have data: 387 vehicles and their 18 variables!
We want to listen the "latent info" disclosed by using PCA

The first three rows of data

	Sports	SUV	Wagon	Minivan	Pickup	AWD	RWD	Retail	Dealer	Engine	Cylinders	Horsepower	CityMPG	HighwayMPG	Weight	Wheelbase	Length	Width
Acura 3.5 RL	0	0	0	0	0	0	0	43755	39014	3.5	6	225	18	24	3880	115	197	72
Acura 3.5 RL Navigation	0	0	0	0	0	0	0	46100	41100	3.5	6	225	18	24	3893	115	197	72
Acura MDX	0	1	0	0	0	1	0	36945	33337	3.5	6	265	17	23	4451	106	189	77

We only have data: 387 vehicles and their 18 variables!
We want to see the "latent info" told by data by using PCA

We use R to conduct PCA analysis

- **All R scripts are application-oriented (data analysis) instead of programming oriented**
- **MBA students can view this as an alternative but more powerful VB!**

Check the 11 variables: they may have some redundant information

Price
- Retail (price $)
- Dealer (prince $)

Power
- Engine (size: liters)
- Cylinders (#cylinders)
- Horsepower

Gas mileage
- CityMPG (city gas mileage)
- HighwayMPG (highway gas mileage)

Size-weight
- Weight (pounds)
- Wheelbase (inches)
- Length (inches)
- Width (inches)

Basic steps in PCA analysis

1. **Load data and visualize data**
2. **Conduct PCA analysis**

Basic steps in PCA analysis

1. **Load data and visualize data**
2. **Conduct PCA analysis**
   1. **Normalize data (zero mean, unit variance data)**

Basic steps in PCA analysis

1. **Load data and visualize data**
2. **Conduct PCA analysis**
   1. **Normalize data (zero mean, unit variance data)**
   2. **Do PCA to the normalized data to get loadings, scores, and variances**

Basic steps in PCA analysis

① **Load data and visualize data**
② **Conduct PCA analysis**
  ① **Normalize data (zero mean, unit variance data)**
  ② **Do PCA to the normalized data to get loadings, scores, and variances**
  ③ **Conduct dimension reduction to get reduced-data**

Basic steps in PCA analysis

① **Load data and visualize data**
② **Conduct PCA analysis**
  ① **Normalize data (zero mean, unit variance data)**
  ② **Do PCA to the normalized data to get loadings, scores, and variances**
  ③ **Conduct dimension reduction to get dimension-reduced-data**
  ④ **Do visualization with 2 PCs or 3 PCs (2D or 3D visualization) for the reduced data→SEE dimension-reduced data in the new coordinate system!**

```
> ## Step 1: load data and visualize data
>  rm(list=ls()) # clear memory
>  vehicles <-read.csv("vehicles.csv")
>  dim(vehicles)  # dimension of data
```

```
> ## Step 1: load data and visualize data
>  rm(list=ls()) # clear memory
>  vehicles <-read.csv("vehicles.csv")
>  dim(vehicles)  # dimension of data

[1] 387  18
```

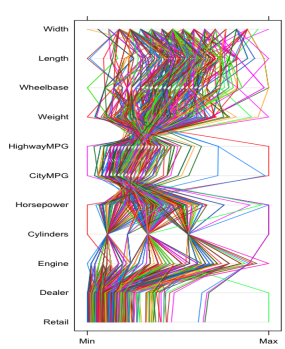
```
> ## Step 1: load data and visualize data
>  rm(list=ls()) # clear memory
>  vehicles <-read.csv("vehicles.csv")
>  dim(vehicles)  # dimension of data

[1] 387  18

>  # Summarize data
>  data<-vehicles[,8:18]  # only last 11 variables are useful
>  summary(data)     # data summary
```

```
> ## Step 1: load data and visualize data
>  rm(list=ls()) # clear memory
>  vehicles <-read.csv("vehicles.csv")
>  dim(vehicles)  # dimension of data

[1] 387  18

>  # Summarize data
>  data<-vehicles[,8:18]  # only last 11 variables are useful
>  summary(data)     # data summary

     Retail            Dealer            Engine          Cylinders        Horsepower       CityMPG         HighwayMPG         Weight       Wheelbase
 Min.   : 10280   Min.   :  9875   Min.   :1.400   Min.   : 3.000   Min.   : 73.0   Min.   :10.00   Min.   :12.00   Min.   :1850   Min.   : 89.0
 1st Qu.: 20997   1st Qu.: 19575   1st Qu.:2.300   1st Qu.: 4.000   1st Qu.:165.0   1st Qu.:18.00   1st Qu.:24.00   1st Qu.:3107   1st Qu.:103.0
 Median : 28495   Median : 26155   Median :3.000   Median : 6.000   Median :210.0   Median :19.00   Median :27.00   Median :3469   Median :107.0
 Mean   : 33231   Mean   : 30441   Mean   :3.127   Mean   : 5.757   Mean   :214.4   Mean   :20.31   Mean   :27.26   Mean   :3532   Mean   :107.2
 3rd Qu.: 39552   3rd Qu.: 36124   3rd Qu.:3.800   3rd Qu.: 6.000   3rd Qu.:250.0   3rd Qu.:21.50   3rd Qu.:30.00   3rd Qu.:3922   3rd Qu.:112.0
 Max.   :192465   Max.   :173560   Max.   :6.000   Max.   :12.000   Max.   :493.0   Max.   :60.00   Max.   :66.00   Max.   :6400   Max.   :130.0
     Length          Width
 Min.   :143   Min.   :64.00
 1st Qu.:177   1st Qu.:69.00
 Median :186   Median :71.00
 Mean   :185   Mean   :71.28
 3rd Qu.:193   3rd Qu.:73.00
 Max.   :221   Max.   :81.00
```

```
   Retail          Dealer          Engine        Cylinders       Horsepower       CityMPG        HighwayMPG       Weight       Wheelbase
Min.   : 10280   Min.   :  9875   Min.   :1.400   Min.   : 3.000   Min.   : 73.0   Min.   :10.00   Min.   :12.00   Min.   :1850   Min.   : 89.0
1st Qu.: 20997   1st Qu.: 19575   1st Qu.:2.300   1st Qu.: 4.000   1st Qu.:165.0   1st Qu.:18.00   1st Qu.:24.00   1st Qu.:3107   1st Qu.:103.0
Median : 28495   Median : 26155   Median :3.000   Median : 6.000   Median :210.0   Median :19.00   Median :27.00   Median :3469   Median :107.0
Mean   : 33231   Mean   : 30441   Mean   :3.127   Mean   : 5.757   Mean   :214.4   Mean   :20.31   Mean   :27.26   Mean   :3532   Mean   :107.2
3rd Qu.: 39552   3rd Qu.: 36124   3rd Qu.:3.800   3rd Qu.: 6.000   3rd Qu.:250.0   3rd Qu.:21.50   3rd Qu.:30.00   3rd Qu.:3922   3rd Qu.:112.0
Max.   :192465   Max.   :173560   Max.   :6.000   Max.   :12.000   Max.   :493.0   Max.   :60.00   Max.   :66.00   Max.   :6400   Max.   :130.0
    Length        Width
Min.   :143   Min.   :64.00
1st Qu.:177   1st Qu.:69.00
Median :186   Median :71.00
Mean   :185   Mean   :71.28
3rd Qu.:193   3rd Qu.:73.00
Max.   :221   Max.   :81.00
```

```
> # Visualize data
> library(lattice) # a graphics package
> parallelplot(data)
```

```
> # Visualize data
> library(lattice) # a graphics package
> parallelplot(data)
```



Vehicle data has very few choices for the variable Cylinders.

But it has big differences in retail prices, weights, horsepower… and width!

```
> # Step 2: PCA analysis
> # Normalize data to zero mean and unit variance data for PCA
> vehicles.pca = prcomp(data, center=TRUE, scale.=TRUE)
```

Why do we need to normalize input data to a zero mean and unit variance data (standard form) before PCA analysis?

```
> # Step 2: PCA analysis
> # Normalize data to zero mean and unit variance data for PCA
>  vehicles.pca = prcomp(data, center=TRUE, scale.=TRUE)
```

Why do we need to normalize input data to a zero mean and unit variance data (standard form) before PCA analysis?

Since the variables are heterogeneous (e.g. retail price, cylinder#, cityMPG), such a normalization process is required. Otherwise PCA will give us wrong results!

```
>  # 1) Retrieve loadings, variances, and scores
>
>  # All principal components (PCs)
>   loadings<-vehicles.pca$rotation
>   loadings[,1:2] # show the first two PCs
```

```
> # 1) Retrieve loadings, variances, and scores
>
> # All principal components (PCs)
>  loadings<-vehicles.pca$rotation
>  loadings[,1:2] # show the first two PCs

                PC1          PC2
Retail     -0.2637504 -0.468508698
Dealer     -0.2623186 -0.470146585
Engine     -0.3470805  0.015347186
Cylinders  -0.3341888 -0.078032011
Horsepower -0.3186023 -0.292213476
CityMPG     0.3104817  0.003365936
HighwayMPG  0.3065886  0.010964460
Weight     -0.3363294  0.167463572
Wheelbase  -0.2662100  0.418177107
Length     -0.2567902  0.408411381
Width      -0.2960546  0.312891350
```

The retail variable 's contribution to the first PC is -0.2637504

What does this mean?

What does this mean?

```
                PC1          PC2
Retail     -0.2637504 -0.468508698
Dealer     -0.2623186 -0.470146585
Engine     -0.3470805  0.015347186
Cylinders  -0.3341888 -0.078032011
Horsepower -0.3186023 -0.292213476
CityMPG     0.3104817  0.003365936
HighwayMPG  0.3065886  0.010964460
Weight     -0.3363294  0.167463572
Wheelbase  -0.2662100  0.418177107
Length     -0.2567902  0.408411381
Width      -0.2960546  0.312891350
```

All the variables except CityMPG and HighwayMPG (the gas-mileages variables) have a negative contribution on to the first PC – the direction with the largest variance.

This means the gas-mileage variables have negative correlations with other variables.

Gas-mileage variables probably should be treated differently than others in data analysis

What does this mean?

```
                PC1          PC2
Retail     -0.2637504 -0.468508698
Dealer     -0.2623186 -0.470146585
Engine     -0.3470805  0.015347186
Cylinders  -0.3341888 -0.078032011
Horsepower -0.3186023 -0.292213476
CityMPG     0.3104817  0.003365936
HighwayMPG  0.3065886  0.010964460
Weight     -0.3363294  0.167463572
Wheelbase  -0.2662100  0.418177107
Length     -0.2567902  0.408411381
Width      -0.2960546  0.312891350
```

Price: Retail, Dealer

Power: Engine, Cylinders, Horsepower

Gas mileage: CityMPG, HighwayMPG

Size: Weight, Wheelbase, Length, Width

It tells us coordinate values on the first PC can separate two types of vehicles

1) **expensive, powerful engine, oil-inefficient, and big vehicles or**
2) **cheap, less-powerful engine, oil-efficient, and small vehicles**

We are going to see

The expensive, powerful engine, oil-inefficient, and big vehicles will have small values on the first PC.

The cheap, less-powerful engine, oil-efficient, and small vehicles will have large values on the first PC

What does this mean for the 2nd PC?

	PC1	PC2
Retail	-0.2637504	-0.468508698
Dealer	-0.2623186	-0.470146585
Engine	-0.3470805	0.015347186
Cylinders	-0.3341888	-0.078032011
Horsepower	-0.3186023	-0.292213476
CityMPG	0.3104817	0.003365936
HighwayMPG	0.3065886	0.010964460
Weight	-0.3363294	0.167463572
Wheelbase	-0.2662100	0.418177107
Length	-0.2567902	0.408411381
Width	-0.2960546	0.312891350

Price

engine

The gas-mileage and engine size variables have little contribution to the 2nd PC.

Gas mileage

Car size

Vehicle size variables have negative correlation with the vehicle price features

It means big vehicles but less expensive (e.g. mini-van, pickup) will contribute more to the 2nd PC than small but expensive ones (e.g. sport cars)

In other words

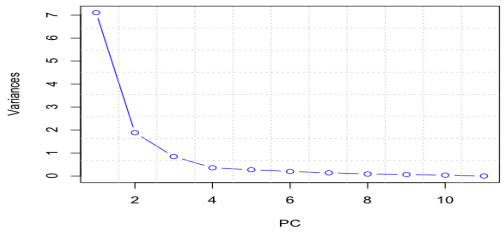
Big and **less expensive** cars (e.g. Pickup, mini-vans) will have large values on the 2nd PC (2nd axis of the new coordinate system)

```
>   # Variance on each PC
>   variances<-vehicles.pca$sdev^2
> variances
```

```
>   # Variance on each PC
>   variances<-vehicles.pca$sdev^2
> variances
[1] 7.1046384308 1.8839247679 0.8497282852 0.3570154894 0.2754355932 0.1979437155 0.1405192086 0.0866388119 0.0663879807 0.0369773622 0.0007903547
```

```
>   # Variance on each PC
>   variances<-vehicles.pca$sdev^2
> variances
[1] 7.1046384308 1.8839247679 0.8497282852 0.3570154894 0.2754355932 0.1979437155 0.1405192086 0.0866388119 0.0663879807 0.0369773622 0.0007903547

> plot(variances, type="b", ylab="Variances", xlab="PC", col="blue")
>   grid(nx=11,ny=8)
```

```
>   # Variance on each PC
>   variances<-vehicles.pca$sdev^2
> variances
[1] 7.1046384308 1.8839247679 0.8497282852 0.3570154894 0.2754355932 0.1979437155 0.1405192086 0.0866388119 0.0663879807 0.0369773622 0.0007903547

> plot(variances, type="b", ylab="Variances", xlab="PC", col="blue")
>   grid(nx=11,ny=8)
```




```
>   # Scores in the new coordinate system
>   scores<-vehicles.pca$x
>   head(scores)  # First 6 rows
```

```
>   # Scores in the new coordinate system
>   scores<-vehicles.pca$x
>   head(scores)  # First 6 rows

                       PC1       PC2       PC3       PC4       PC5       PC6       PC7       PC8       PC9      PC10       PC11
Acura 3.5 RL      -1.5654166 0.44669421 -0.2870171 0.6098600 0.46565572 -0.3261484 -0.1710511 0.02844808 0.22029751 -0.20332981 -0.048230584
Acura 3.5 RL Navigation -1.6335337 0.33929273 -0.3452409 0.6788642 0.44813694 -0.3456422 -0.2237604 0.03471782 0.23499520 -0.20390280 -0.050407699
Acura MDX         -1.9041537 0.41060707 0.5519813 0.2956175 -1.23445167 0.5834249 0.1956721 -0.34753490 -0.43226542 0.12294837 0.003284217
Acura NSX S       -1.5881428 -3.85758573 -0.3563574 1.1480124 -0.20949332 0.5350684 -0.5601955 0.35653160 0.37990771 -0.20028128 -0.073888305
Acura RSX          2.6513314 -0.65360524 0.1731496 0.2890318 0.096999110 0.1412389 0.5194514 -0.06043627 -0.01390064 -0.05476417 -0.010159741
Acura TL          -0.4406721 -0.08100219 -0.1895334 -0.1317345 0.02068582 0.2309155 0.6178140 -0.03623753 -0.20235842 0.11871858 -0.002230600
```

Scores: the coordinate values of the original data in the new coordinate system (new data!)

```
> # 2) Dimension reduction
>
> #Calculate explained variance ratios
> explainedVarianceRatios <- variances/sum(variances)
> explainedVarianceRatios
```

Explained variance ratios are ratios between each PC's variance over the total variances

The explained variance ratio on the ith PC: $\frac{\lambda_i}{\sum_{i=1}^{p} \lambda_i}$

Suppose there are total four PCs and corresponding variances are 6.0, 3.0, 0.9 0.1:

The total variances: 6.0+3.0+0.9+0.1=10.

The explained variance ratios on each PC: 60%, 30% , 9% and 1%


```
> # 2) Dimension reduction
>
> #Calculate explained variance ratios
> explainedVarianceRatios <- variances/sum(variances)
> explainedVarianceRatios
```

Explained variance ratios are ratios between each PC's variance over the total variances

The explained variance ratio on the ith PC: $\frac{\lambda_i}{\sum_{i=1}^{p} \lambda_i}$

Suppose there are total four PCs and corresponding variances are 6.0, 3.0, 0.9 0.1:

The total variances: 6.0+3.0+0.9+0.1=10.

The explained variance ratios on each PC: 60%, 30% , 9% and 1%

If a PC has a low explained variance ratio, then it will be viewed as an less important one.

Thus, we can just drop the PCs with small explained variance ratios

A related measure always used together with the explained variance ratio : Cumulative explained variance ratios:

Cumulative explained variance ratios are ratios between cumulative sum of PC variances over the total variances

The cumulative explained variance ratio on first k PCs: $\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$

- Suppose there are total four PCs and corresponding variances are 6.0, 3.0, 0.9 0.1
- The explained variance ratios: 60%, 30% , 9% and 1%
- The cumulative explained variance ratios on the first 3 PCs are 99% (60% +30%+9%)

```
>   # 2) Dimension reduction
>
>   #Calculate explained variance ratios
>   explainedVarianceRatios <- variances/sum(variances)
>   explainedVarianceRatios
```

```
>   # 2) Dimension reduction
>
>   #Calculate explained variance ratios
>   explainedVarianceRatios <- variances/sum(variances)
>   explainedVarianceRatios

[1] 6.458762e-01 1.712659e-01 7.724803e-02 3.245595e-02 2.503960e-02 1.799488e-02 1.277447e-02 7.876256e-03 6.035271e-03 3.361578e-03 7.185043e-05
```

```
>   # 2) Dimension reduction
>
>   #Calculate explained variance ratios
>   explainedVarianceRatios <- variances/sum(variances)
>   explainedVarianceRatios

[1] 6.458762e-01 1.712659e-01 7.724803e-02 3.245595e-02 2.503960e-02 1.799488e-02 1.277447e-02 7.876256e-03 6.035271e-03 3.361578e-03 7.185043e-05


>   # Calculate the cumulative explained variance ratios
>   cumulativeExplainedVarianceRatios<-cumsum(variances)/sum(variances)
>   cumulativeExplainedVarianceRatios
```

```
> # 2) Dimension reduction
>
> #Calculate explained variance ratios
> explainedVarianceRatios <- variances/sum(variances)
> explainedVarianceRatios

[1] 6.458762e-01 1.712659e-01 7.724803e-02 3.245595e-02 2.503960e-02 1.799488e-02 1.277447e-02 7.876256e-03 6.035271e-03 3.361578e-03 7.185043e-05

> # Calculate the cumulative explained variance ratios
> cumulativeExplainedVarianceRatios<-cumsum(variances)/sum(variances)
> cumulativeExplainedVarianceRatios

[1] 0.6458762 0.8171421 0.8943901 0.9268461 0.9518857 0.9698806 0.9826550 0.9905313 0.9965666 0.9999281 1.0000000
```

```
> # 2) Dimension reduction
>
> #Calculate explained variance ratios
> explainedVarianceRatios <- variances/sum(variances)
> explainedVarianceRatios

[1] 6.458762e-01 1.712659e-01 7.724803e-02 3.245595e-02 2.503960e-02 1.799488e-02 1.277447e-02 7.876256e-03 6.035271e-03 3.361578e-03 7.185043e-05

> # Calculate the cumulative explained variance ratios
> cumulativeExplainedVarianceRatios<-cumsum(variances)/sum(variances)
> cumulativeExplainedVarianceRatios

[1] 0.6458762 0.8171421 0.8943901 0.9268461 0.9518857 0.9698806 0.9826550 0.9905313 0.9965666 0.9999281 1.0000000

> # Keep the first 5 PCs because they count >95% data variance!
> k<-5
> reduced_data<-scores[,1:k]
> dim(reduced_data)
```

```
> # 2) Dimension reduction
>
> #Calculate explained variance ratios
> explainedVarianceRatios <- variances/sum(variances)
> explainedVarianceRatios

[1] 6.458762e-01 1.712659e-01 7.724803e-02 3.245595e-02 2.503960e-02 1.799488e-02 1.277447e-02 7.876256e-03 6.035271e-03 3.361578e-03 7.185043e-05

> # Calculate the cumulative explained variance ratios
> cumulativeExplainedVarianceRatios<-cumsum(variances)/sum(variances)
> cumulativeExplainedVarianceRatios

[1] 0.6458762 0.8171421 0.8943901 0.9268461 0.9518857 0.9698806 0.9826550 0.9905313 0.9965666 0.9999281 1.0000000

> # Keep the first 5 PCs because they count >95% data variance!
> k<-5
> reduced_data<-scores[,1:k]
> dim(reduced_data)

[1] 387   5
```

Dimension reduced but keep the most data variances!
Original data is a 387x11 matrix

```
> head(reduced_data) # first 6 rows
                          PC1         PC2        PC3        PC4         PC5
Acura 3.5 RL        -1.5654166  0.44669421 -0.2870171  0.6098609  0.46565572
Acura 3.5 RL Navigation -1.6335337  0.33929273 -0.3452409  0.6788642  0.44813694
Acura MDX           -1.9041537  0.41060707  0.5519813  0.2956175 -1.23445167
Acura NSX S         -1.5881428 -3.85758573 -0.3563574  1.1480124 -0.20949332
Acura RSX            2.6513314 -0.65360524  0.1731496  0.2890318  0.09699110
Acura TL            -0.4406721 -0.08100219 -0.1895334 -0.1317345  0.02068582
```

PCA provides powerful visualization capabilities to let us see data in the new coordinate system in 2D and 3D.

It is still a dimension reduction by only keeping 2/3 PCs in the new coordinate system.

That is, see dimension-reduced data in the new coordinate system!

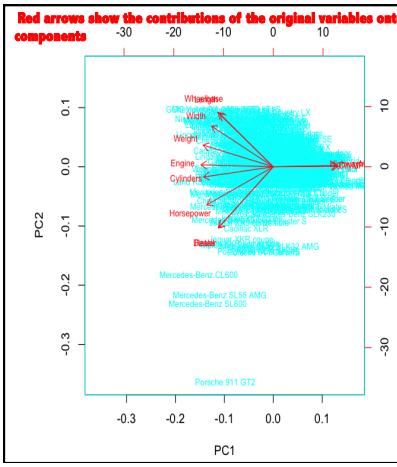
```
> # 3) Visualization
>
>  # visualize data in the new coordinate system with two PCs)
>   biplot(vehicles.pca, choices=c(1,2),  cex=0.7, xlab="PC1", ylab="PC2", col = c("cyan", "red"))
```

A biplot visualizes

1. **the magnitude of each variable's contribution to the first two principal components**
2. **how each observation is represented in terms of those components.**

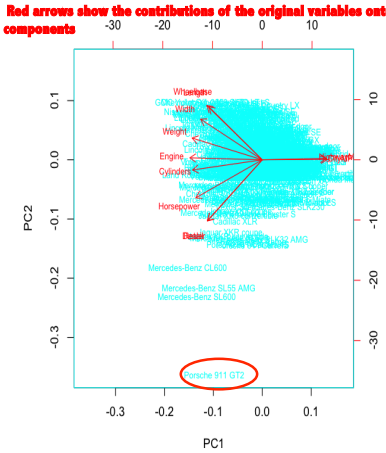
We can find outliers or even clusters from such a biplot visualization!

Red arrows show the contributions of the original variables onto the two principal components



1) **The gas-mileages variables (cityMPG, highwayMPG) have negative correlations with other variables.**

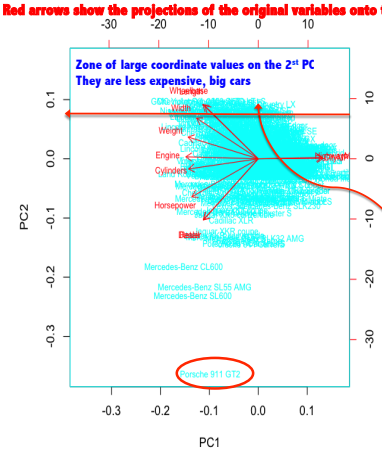
Red arrows show the contributions of the original variables onto the two principal components



1) The gas-mileages features (cityMPG, highwayMPG) have negative correlations with other variables.

2) Porsche 911 GT2 (small, expensive) is an outlier that has the lowest value with respect to the PC2 (2nd axis).

Red arrows show the projections of the original variables onto the two principal components

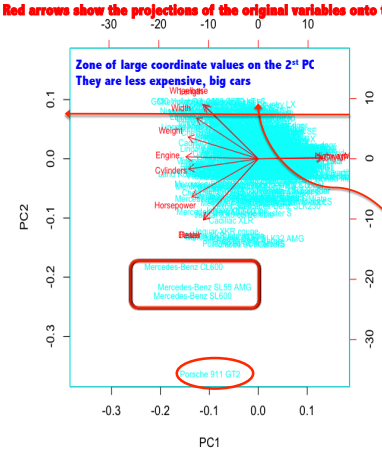


1) The gas-mileages features (cityMPG, highwayMPG) have negative correlations with other variables.

2) Porsche 911 GT2 (small, expensive) is an outlier that has the lowest value with respect to the PC2 (2nd axis).

3) Pickup trucks, mini-vans have large values with respect to the 2nd PC

Red arrows show the projections of the original variables onto the two principal components



1) The gas-mileages features (cityMPG, highwayMPG) have negative correlations with other variables.

2) Porsche 911 GT2 (small, expensive) is an outlier that has the lowest value with respect to the PC2 (2nd axis).

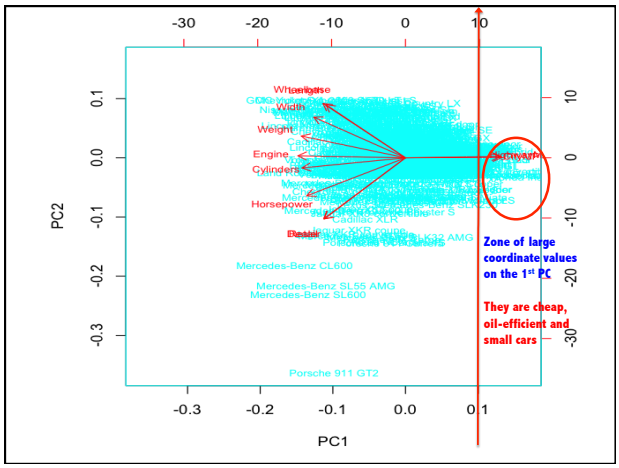
3) Pickup trucks, mini-vans have large values with respect to the 2nd PC

4) M-benz CL600, SL55 AMG, SL600 can be viewed as a naturally grouped cluster: they all have low values on the 1st and 2nd PCs

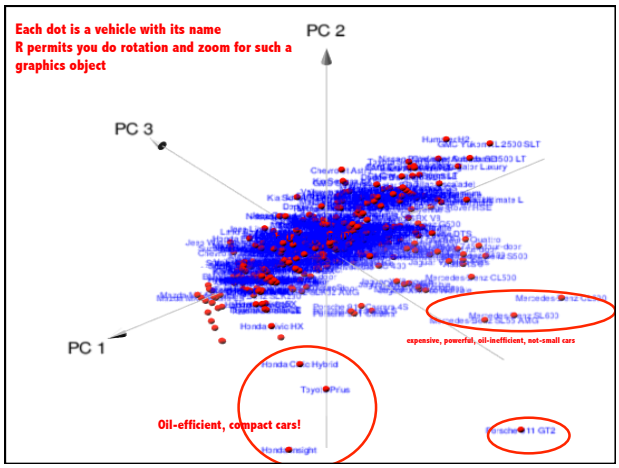
Three Benz cars "naturally grouped" together: all have low values on the 1st PC
They are expensive, powerful, oil-inefficient, not-small cars

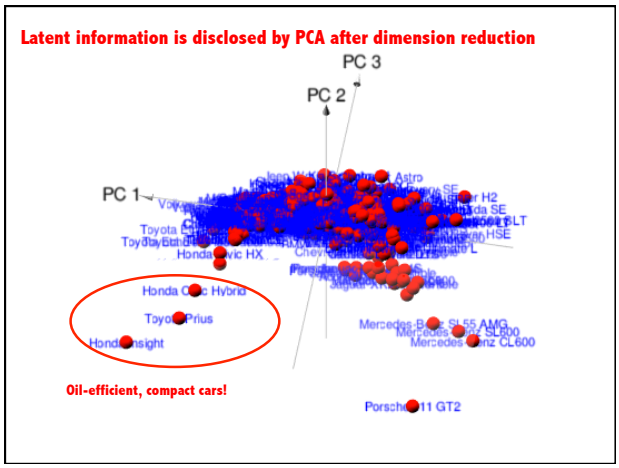
```
                          Retail Dealer Engine Cylinders Horsepower CityMPG HighwayMPG Weight Wheelbase Length Width
Mercedes-Benz CL600       128420 119600    5.5        12        493      13         19   4473       114    196    73
Mercedes-Benz SL55 AMG    121770 113388    5.5         8        493      14         21   4235       101    179    72
Mercedes-Benz SL600       126670 117854    5.5        12        493      13         19   4429       101    179    72
```

```
    Retail          Dealer          Engine       Cylinders       Horsepower       CityMPG        HighwayMPG       Weight       Wheelbase
 Min.   : 10280   Min.   :  9875   Min.   :1.400   Min.   : 3.000   Min.   : 73.0   Min.   :10.00   Min.   :12.00   Min.   :1850   Min.   : 89.0
 1st Qu.: 20997   1st Qu.: 19575   1st Qu.:2.300   1st Qu.: 4.000   1st Qu.:165.0   1st Qu.:18.00   1st Qu.:24.00   1st Qu.:3107   1st Qu.:103.0
 Median : 28495   Median : 26155   Median :3.000   Median : 6.000   Median :210.0   Median :19.00   Median :27.00   Median :3469   Median :107.0
 Mean   : 33231   Mean   : 30441   Mean   :3.127   Mean   : 5.757   Mean   :214.4   Mean   :20.31   Mean   :27.26   Mean   :3532   Mean   :107.2
 3rd Qu.: 39552   3rd Qu.: 36124   3rd Qu.:3.800   3rd Qu.: 6.000   3rd Qu.:250.0   3rd Qu.:21.50   3rd Qu.:30.00   3rd Qu.:3922   3rd Qu.:112.0
 Max.   :192465   Max.   :173560   Max.   :6.000   Max.   :12.000   Max.   :493.0   Max.   :60.00   Max.   :66.00   Max.   :6400   Max.   :130.0
     Length          Width
 Min.   :143   Min.   :64.00
 1st Qu.:177   1st Qu.:69.00
 Median :186   Median :71.00
 Mean   :185   Mean   :71.28
 3rd Qu.:193   3rd Qu.:73.00
 Max.   :221   Max.   :81.00
```




```
>   # visualize new data in the new coordinate system with 3 PCs (3D visualization)
>
>   library(pca3d) # 3D PCAvisualization package
>   library(rgl)   # 3D visualization package using openGL
>
>   x<-rownames(vehicles)      # Labels for observations
>   pca3d(scores,radius=2.0, col="red")          #3D PCA plot
[1] 0.3473586 0.3928380 0.2541229
Creating new device
>   text3d(scores, texts=x, cex=0.7, col="blue") # Add labels
```





After PCA, you have a reduced-data: less noise, low-dimensional, redundant-information removed, but with most data variances!)

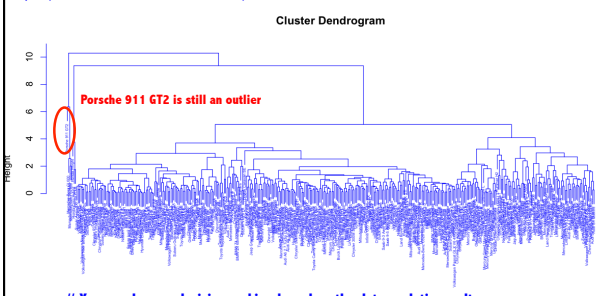
Clustering data by using the reduced-data to get more information

Do hierarchical clustering from reduced Data

```
vehicles.pdist<-dist(reduced_data, method="euclidean")
vehicles.hc <- hclust(vehicles.pdist, method="average")
plot(vehicles.hc, cex=0.5, col="blue")
```

Do hierarchical clustering from reduced Data

```
vehicles.pdist<-dist(reduced_data, method="euclidean")
vehicles.hc <- hclust(vehicles.pdist, method="average")
plot(vehicles.hc, cex=0.5, col="blue")
```



 # You can do your decision making based on the data analytic results (e.g., order more big, less expensive cars in store…)

Why using reduced data to do HC will be better than using the original data?

Why using reduced data to do HC will be better than using the original data?

1. The reduced data removes redundant information by doing dimension reduction
2. Original data is even before normalization!

Besides clustering, you can do more analysis after PCA

① Use the reduced-data to rank each observation's significance (e.g. stock-ranking) or conduct other hypothesis testing.

② Use the reduced data as a training data to conduct classification/regression by integrating other classification/ regression algorithms

① support vector machines (SVM)
② linear discriminant analysis (LDA)
③ logistic regression
④ random forest tree (RF)
⑤ gradient boosting

③ Use the reduced data to do other data analysis according to your needs….

Q1: Our first PC has 65% explained variance ratio. Is it possible that the first PC will have an almost 100% explained variance ratio and other PCs has almost tiny or even a zero explained variance ratio?

Q1: Our first PC has 65% explained variance ratio. Is it possible that the first PC will have an almost 100% explained variance ratio and other PCs has almost tiny or even a zero explained variance ratio?

A: Yes. It relies on data. It often happens to some high-dimensional omics data.

Q2: Our current reduced data is based on 5 PCs with a cumulative explained variance ratio: 95%. Can I get a reduced-data based on only 2 or 3 PCs?

Q2: Our current reduced data is based on 5 PCs with cumulative explained variance ratio: 95%. Can I get a reduced-data based on only 2 or 3 PCs?

A: Sure. The reduced-data with 2/3 PCs have 81% and 89% cumulative explained variance ratios.

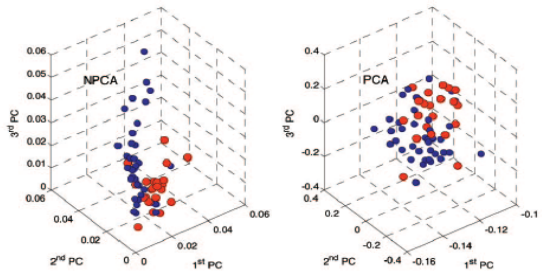
The general rule is the reduced-data should have at least 50% cumulative explained variance ratios.

Q3: PCA is so powerful. However, what are the weakness of PCA?

Q3: What are the weakness of PCA?

1. Principle components **are only uncorrelated (orthogonal)** instead of "independent"
   - Independent component analysis (ICA)
     - Widely used in Financial data analytics

2. PCA is **not "purely additive"**: PCs consist of both positive and negative entries.
   - Nonnegative matrix factorization (NMF)
   - Nonnegative principal component analysis (NPCA) (all PCs have nonnegative entries)
     - Widely used in health informatics, bioinformatics and image data analytics
     - Also used in business data analytics
     - **Both need to solve corresponding nonlinear programming problems!**



The visualization of 62 observations (40 cancer ones + 22 normal ones) from a benchmark data with 2000 variables by using first three PCs from NPCA and PCA, respectively.

Cancer samples are colored by "blue" and normal samples are colored by "red." (Han 2010)

NPCA is more representative: samples are much less inter-wined under NPCA than PCA!

What are the weakness of PCA cont'd?

3. PCA is not rigorously built upon a probability model

- Probabilistic principle component analysis (PPCA)

4. PCA is only a linear dimension-reduction method instead of a nonlinear one.

- Kernel principal component analysis (KPCA): a nonlinear extension of PCA using kernel tricks

What are the weakness of PCA? Cont'd

5. PCA is only based on single-resolution data analysis and each variable is viewed as indivisible information unit. It can't capture "data derivatives".

- Derivative component analysis (DCA): PCA-based data reconstruction in a multi-resolution data analysis

6. PCA lacks a sparse representation, which is essential for data locality. Each PC is "jammed with" nonzero entries

- Sparse principal component analysis (SPCA): enhance sparse representation for PCA by solving semi-definite programming

A little bit formal description

- Given a zero mean data $X = \begin{pmatrix} x_1^t \\ x_2^t \\ \vdots \\ x_n^t \end{pmatrix}$, $x_i \in R^p$, PCA is equivalent to solving the problem

$$\max J(U) = \frac{1}{2}||U^t X||_F^2, s.t. U^t U = I$$

- $||U^t X||_F^2$ is the mathematical representation of the sum of all eigenvalues of the covariance matrix C:

$$trace(C) = \sum_{i=1}^{p} \lambda_i$$

- $U = [u_1, u_2, ...u_p]$ is the PC matrix: all maximum variance directions