- ผ่านทวน
- ทบทวนการบ้าน

DS : be able to perform ML pipeline

- data prep (impute missing, train/test split
- traditional ML Decision tree, Linear, Logistic, Gridsearch, pipeline, Scikit, kmean
- model evaluation / parameter tuning
- MLflow

Visual : Hint ข้อสอบส่วน อ.วีระ ฝากมาแจ้งนะครับ

ข้อสอบส่วนของอ.วีระ
ส่วนแรก Spatial analysis จะให้นิสิตวิเคราะห์ข้อมูลเชิงพื้นที่ โดยจะมีโปรแกรมให้บางส่วน นิสิตจะต้องเขียนโปรแกรมเพิ่มเติมเพื่อตอบคำถาม สิ่งที่ควรรู้คือ streamlit, pydeck, dbscan

ส่วนที่สอง network analysis จะให้นิสิตวิเคราะห์ข้อมูลเครือข่ายโดยใช้โปรแกรม gephi โดยให้ตอบคำถามคล้ายๆกับที่ให้โจทย์ตัวอย่างไว้ใน github ของรายวิชา (ใน folder code)

MLFlow,
Spark, RedDB, kafka, Web

Gephi, Spatial analysis (pydeck, dbscan)

DATAQUEST BLOG

28 DECEMBER 2016 / DATA CLEANING

Preparing and Cleaning Data for Machine Learning

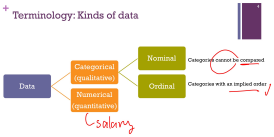
- 1) Examining the Data Set
- 2) Narrowing down columns manually
  - Remove Id's
  - Irrelevant variables
  - Remove zipcode & date
  - Temporal infidelity (data from future)
  - Calculated variables
  - Decide target
    - Select studied cases
    - Distribution of target variables
  - Remove flat values
- 3) Preparing features for ML
  - Preview data
  - Handling missing values df.fillna(0)
    - Drop unqualified features
  - Investigate categorical features
    - Drop too many unique values (treat as Id)
    - Convert ordinal to numeric
    - Convert categorical to numeric
  - Check all numeric variables
- 4) Other preprocessing steps:
  - Train/Test/Validate

9

```
# Forward fill missing values
df_ffill = df.fillna(method='ffill')

# Backward fill missing values
df_bfill = df.fillna(method='bfill')
```

df. median
df. mean()
mode = (df. iloc[0])



ne-learning-preparing-data

1) Examining the Data Set

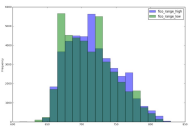
10

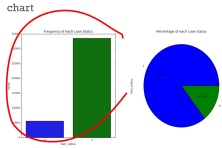
df. describe()
df. info()

- Numerical variables
  - Out of ranges
  - Distribution: histogram
- Categorical variables
  - Miscodes
  - Distribution: frequency table, bar chart
- Target variable
  - Understand class distribution (proportion of each class): bar chart, pie chart

df. value_count

ตัวแปรตาม





↳ ปัญหา: imbalance data

2) Narrowing down columns (cont.): Remove unqualified features

18

ของ data ต้องการ transform

- Remove unqualified features
  1. ID's (lack of generalization; overfit)
  2. Variables with missing values > 50%
  3. Categorical variables
     - Too many unique values (treat as ID's)
     - Flat values (underfit)
- Special ways to treat these data
  - High cardinalities of categorical inputs
    - Recode, consolidation (grouping)
  - Zip code → อำเภอ → รวม
    - Distance to closet branch
  - Date/time
    - Recency
    - Month, day of week, year
    - Hours, period of days

df. value_counts()
df. isnull(). sum()

3) Preparing features for ML (cont.): 3.1 Impute missing values

ตำแหน่ง

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

$$Spend = 500 + 10 \times IncomeK - 20 \times Age$$

- For missing in target, those examples must be removed.
- For missing in inputs, either remove those examples or impute missing values

- 1) Statistical approach
  - Numerical variables:
    - Mean
    - Median
  - Categorical variables:
    - Mode (most-frequent)
  - Stats by group can improve the performance, e.g., income by age group.
- 2) Model-based approach
  - x1 ~ (x2, x3, x4, …)
  - Income ~ Age
  - Tree-based imputation

ตัวแปรต้น missing เพิ่มได้

ตัวแปรตาม missing เพิ่มได้ด้วย

$$Spend = 500 + 10 \times IncomeK - 20 \times Age + 3 \times Province$$

? วิธีการ ที่ใช้ได้

3) Preparing features for ML (cont.): 3.2 Categorical to numeric variables

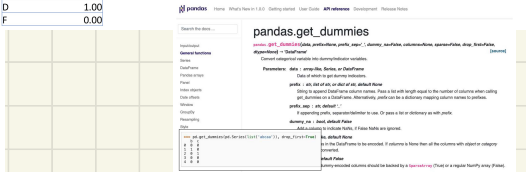
convert

- Ordinal variable
  - Enumerate

Grade	GradeNum
A	4.00
B+	3.50
B	3.00
C+	2.50
C	2.00
D+	1.50
D	1.00
F	0.00

Size	SizeNum
XL	5
L	4
M	3
S	2
XS	1

- Nominal variable
  - (1) One-hot vector (dummy codes)
  - (2) Target Averaging (prob)
  - (3) Weight of Evidence (WoE)
  - (4) Smoothed weight of evidence (SWoE)



X_train,X_test,y_train,y_test = train_test_split(X,y,stratify=y,test_size=0.25, random_state=42)

(1) One-hot vector (dummy codes)

17

- Dummy coding = (n-1) dummy codes

3 level = 2 dummy code

Branch	BranchNum	D_BKK	D_Patum	D_Non
BKK	1	1	0	0
Patumtani	2	0	1	0
Nontaburi	3	0	0	1

D_BKK	B_Patum
1	0
0	1
0	0 reference

4) Other preprocessing steps: Train/Test/Validate – Overfitting issue on train

29

Training Data

Validation Data

Testing Data

inputs / target

Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	Yes
35	50,000	Female	Nontaburi	Yes
32	35,000	Male	Bangkok	No

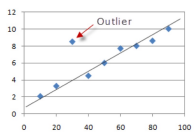
Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	Yes
35	50,000	Female	Nontaburi	Yes

Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	?

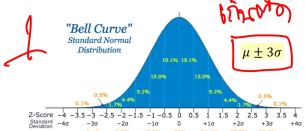
stratification

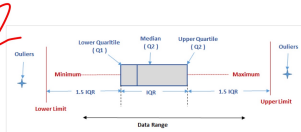
train	test
Yes 80%	Yes 80%
No 20%	No 20%

3) Preparing features for ML (cont.): 3.3 Truncate outliers



- Outlier, leverage points, extreme values



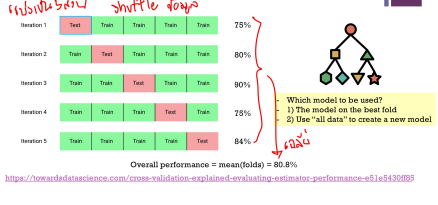


Percentile
1st
2.5th
5th
10th
25th
50th
75th
90th
95th
97.5th
99th

K-Fold Cross Validation

How to fix overfitting issue on test

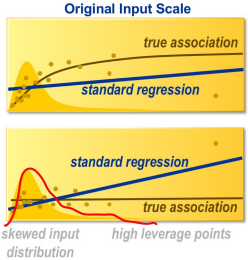
40



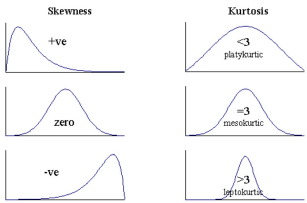
https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

3) Preparing features for ML (cont.): 3.4 Feature transformation



- Skewness
- Example: Salary, Balance in bank account
- **Solutions: Log, Binning**

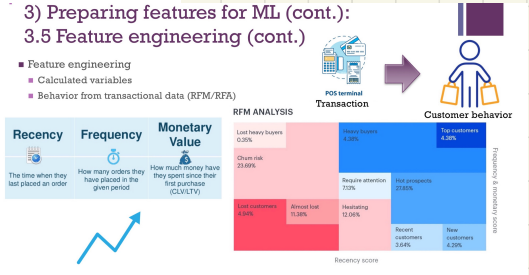


3) Preparing features for ML (cont.): 3.4 Feature transformation - Log

	Spending	Spending with outliers	LOG10(Spending)	LOG10(Spending with outliers)
	2,500.00	2,500.00	3.40	3.40
	2,900.00	2,900.00	3.46	3.46
	3,200.00	3,200.00	3.51	3.51
	4,000.00	4,000.00	3.60	3.60
	4,500.00	4,500.00	3.65	3.65
	6,200.00	6,200.00	3.79	3.79
		10,000,000.00		7.00
mean	3,883.33	1,431,900.00	3.57	4.06

3) Preparing features for ML (cont.): 3.5 Feature engineering (cont.)

- Feature engineering
  - Calculated variables
  - Behavior from transactional data (RFM/RFA)



Data leaking from testing data in training data

ข้อมูล เดียวกัน ไม่ควรอยู่ ทั้ง train และ test

- Split by subjects/videos rather than individual images
- For time series data, split by period of time

from ข้อมูลนี้ ใช้ เทคนิค คล้ายๆ train ได้อีกนะ

Remark: Random Seed

- The experiment must be able to reconstruct (replicate).
- All randoms must be assigned a radom seed.
  - random.seed(12345)
  - random_state option

ใช้ seed อ้างอิง

Feature selection Helper

```python
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif  # or any other scoring function

# Create SelectKBest object with scoring function
selector = SelectKBest(score_func=f_classif, k=3)  # Select top 3 features

# Fit the selector to your data
selector.fit(X, y)

# Transform your data to include only the selected features
X_selected = selector.transform(X)
```

parameter

```python
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import f_classif  # or any other scoring function

# Create SelectPercentile object with scoring function
selector = SelectPercentile(score_func=f_classif, percentile=20)  # Keep top 20% of f

# Fit the selector to your data
selector.fit(X, y)

# Transform your data to include only the selected features
X_selected = selector.transform(X)
```

Classification Problems:
For classification tasks, where the target variable is categorical, some common score_func options are:

- f_classif:
  Computes the ANOVA F-value for the provided samples.
  Useful when the features are numeric.
  Assumes the features are normally distributed and independent.
- chi2:
  Computes the chi-squared statistic between each non-negative feature and the target.
  Suitable for categorical features.
  Assumes the features are non-negative.
- mutual_info_classif:
  Estimates mutual information for a discrete target variable.
  Works for both continuous and categorical features.
  Captures any kind of statistical dependency, linear or non-linear.

Regression Problems:
For regression tasks, where the target variable is continuous, some common score_func options are:

- f_regression:
  Computes the F-value between each feature and the target.
  Suitable for linear regression tasks.
  Assumes the features are normally distributed and independent.
- mutual_info_regression:
  Estimates mutual information for a continuous target variable.
  Useful for capturing non-linear dependencies.
  Works for both continuous and categorical features.

```python
from sklearn.feature_selection import SelectKBest, f_classif, chi2, mutual_info_class

# Create SelectKBest objects with different scoring functions
selector_f_classif = SelectKBest(score_func=f_classif, k=3)
selector_chi2 = SelectKBest(score_func=chi2, k=3)
selector_mutual_info = SelectKBest(score_func=mutual_info_classif, k=3)

# Fit and transform using each selector
X_selected_f_classif = selector_f_classif.fit_transform(X, y)
X_selected_chi2 = selector_chi2.fit_transform(X, y)
X_selected_mutual_info = selector_mutual_info.fit_transform(X, y)
```

```python
from sklearn.feature_selection import SelectKBest, f_regression, mutual_info_regressi

# Create SelectKBest objects with different scoring functions
selector_f_regression = SelectKBest(score_func=f_regression, k=3)
selector_mutual_info_regression = SelectKBest(score_func=mutual_info_regression, k=3)

# Fit and transform using each selector
X_selected_f_regression = selector_f_regression.fit_transform(X, y)
X_selected_mutual_info_regression = selector_mutual_info_regression.fit_transform(X,
```

Chi-square (χ^2) is a statistical test commonly used to determine whether there is a significant association between two categorical variables. It assesses whether the observed frequencies of categorical data differ from the expected frequencies under the assumption of independence between the variables.

ML

Linear Regression

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Generate random data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 3 * X.squeeze() + np.random.randn(100) * 2  # True relationship: y = 3*X + noise

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Create and train the model
model = LinearRegression().fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("R-squared (R²) Score:", r2)
```

Sequential Feature Selection (SFS) is a technique used for selecting a subset of features from a larger set of features based on their contribution to the performance of a machine learning model. The idea is to iteratively build a model with an increasing number of features, evaluating the performance at each step, and selecting the best subset of features.

Steps in Sequential Feature Selection:
Initialization: Start with an empty set of features.
Selection: Iteratively add or remove features based on a criterion (forward, backward, or bidirectional).
Forward Selection: Start with an empty set and add the feature that provides the best improvement in model performance.
Backward Elimination: Start with all features and remove the least important feature at each step.
Bidirectional: A combination of forward and backward selection, adding and removing features alternately.
Stop Criterion: Stop when a predetermined condition is met (e.g., a specific number of features, optimal performance, etc.).
Benefits of Sequential Feature Selection:
Improves Model Performance: Focuses on the most relevant features, potentially improving the model's performance.
Reduces Overfitting: Helps in reducing overfitting by selecting only the most informative features.
Example (Forward Selection):
Let's say we have a dataset with multiple features and a target variable. We can use forward selection as follows:

```
python
Copy code
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# Load Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a logistic regression model
model = LogisticRegression()

# Forward Selection
sfs = SFS(model,
          k_features=3,  # Select 3 features
          forward=True,  # Forward selection
          floating=False,
          scoring='accuracy',
          cv=5)

# Fit the model
sfs.fit(X_train, y_train)

# Get the selected feature indices and names
selected_features_idx = sfs.k_feature_idx_
selected_features_names = [iris.feature_names[i] for i in selected_features_idx]

print("Selected Features Indices:", selected_features_idx)
print("Selected Features Names:", selected_features_names)
In this example:

We use the Iris dataset and split it into training and testing sets.
We create a logistic regression model.
We use SequentialFeatureSelector from mlxtend to perform forward selection.
The k_features parameter specifies the number of features to select (in this case, 3).
The selected feature indices and names are printed at the end.
Important Notes:
Computational Cost: Sequential feature selection can be computationally expensive, especially for large datasets or when using complex models.
Tuning Parameters: You might need to tune parameters such as the number of features to select, the scoring metric, and cross-validation settings.
Model Selection: It's essential to consider the model's performance on a validation set or using cross-validation after feature selection.
Sequential Feature Selection is a powerful technique for feature selection, especially when dealing with datasets with many features. It helps in improving model performance, reducing overfitting, and gaining insights into the most relevant features for the task at hand.
```

Confusion Matrix

84

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

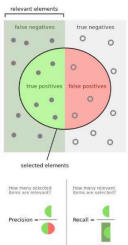
- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)

- Accuracy = (TP + TN) / total
- Misclassification = (FP + FN) / total

Precision, Recall, F1

88

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	



- Precision = correctly predict = TP / (TP + FP)
- Recall = coverage = TP / (TP + FN)
- F1 = (2*pre*rec) / (pre + rec)

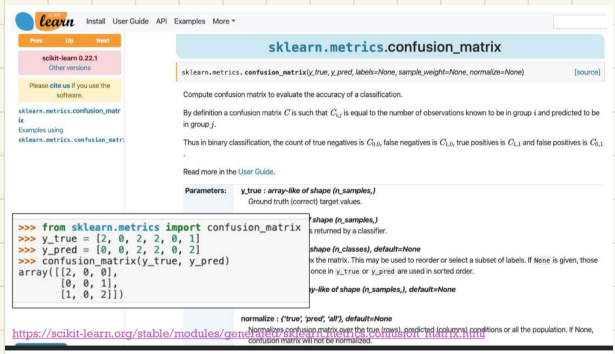
Precision, Recall, F1: Average

88

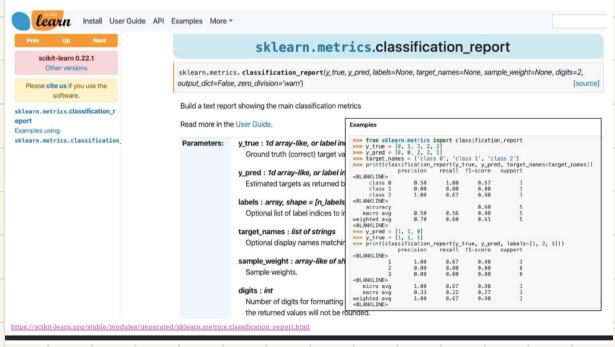
n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

- What is a positive class?
  - 1) Direct target marketing? Yes
  - 2) Intelligent diagnosis to predict "Corona" or "Flu" – both are important.
- If there is no positive (all classes are important)
  - Macro Average
  - Weighted Average (Micro Average) by the amount of data in that class

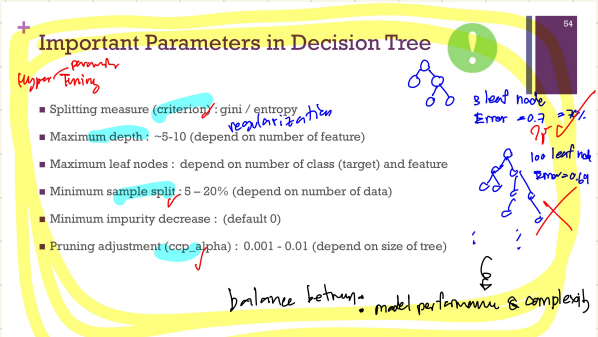
- Precision = correctly predict = TP / (TP + FP)
- Recall = coverage = TP / (TP + FN)
- F1 = (2*pre*rec) / (pre + rec)

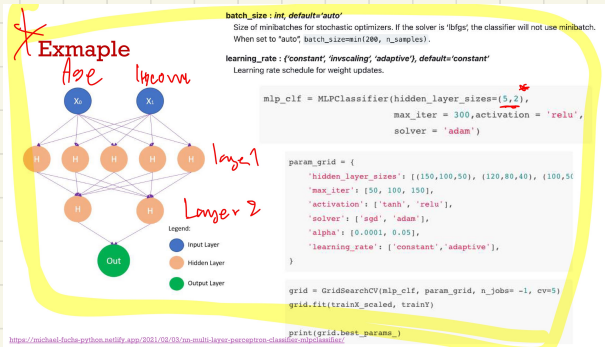


https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html



https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html





Sure, let's break down CAP theorem and ACID in a beginner-friendly way:

CAP Theorem:
CAP theorem, also known as Brewer's theorem, deals with distributed data stores, like databases, in computer science. It states that it's impossible for a distributed system to simultaneously provide all three of the following guara

1. **Consistency (C)**: Every read receives the most recent write or an error. In other words, all nodes in the system have the same data at the same time.

2. **Availability (A)**: Every request to the system gets a response, even if one of the nodes in the system fails.

3. **Partition tolerance (P)**: The system continues to operate despite network partitions (communication failures) between nodes.

Explanation:
- **Consistency (C)** means that all nodes in a distributed system have the same data at the same time. So if you write data to one node and then immediately read from another, you should get the same data, not something outda

- **Availability (A)** means that every request to the system gets a response, even if some nodes are down or unreachable. Users can always read and write to the system.

- **Partition Tolerance (P)** means that the system continues to operate even if there are network issues causing some nodes to be unreachable by others. It ensures that the system can handle network failures without data loss.

Implications:
- If you choose Consistency and Partition Tolerance (CP), then during a network partition, the system will stop taking requests to maintain consistency.

- If you choose Availability and Partition Tolerance (AP), then during a network partition, the system will still be available but may return data that is not fully up-to-date (as it prioritizes availability over consistency).

- If you choose Consistency and Availability (CA), then during a network partition, the system will sacrifice availability to ensure that all nodes have consistent data.

ACID:
ACID is a set of properties that ensure that database transactions are processed reliably. It stands for:

1. **Atomicity**: Ensures that either all operations within a transaction succeed, or if any operation fails, the entire transaction fails, and the database state is left unchanged.

2. **Consistency**: Guarantees that only valid data is written to the database. If a transaction starts with the database in a consistent state, it will end with the database in a consistent state.

3. **Isolation**: Transactions are executed in isolation from each other, so the result of a transaction is not visible to other transactions until it is committed. This prevents interference between transactions.

4. **Durability**: Once a transaction is committed, its changes are permanent and will not be lost, even in the event of a system crash.

Explanation:
- **Atomicity**: Think of this as an "all or nothing" rule. If you're transferring money from one account to another, either the full amount is moved successfully, or nothing happens at all. No partial transfers.

- **Consistency**: This ensures that your data remains in a valid state at all times. For example, if you have a rule that a person's age can't be negative, any transaction that tries to set a person's age to -5 would be rejected, maint

- **Isolation**: Transactions should not interfere with each other. If two transactions are happening at the same time, they should not impact each other until they are both completed. This prevents scenarios like one transaction rea

- **Durability**: Once a transaction is confirmed (like clicking "submit" on an order), the changes are saved permanently, even if the system crashes immediately after. So, you won't lose your order confirmation even if the power go

Conclusion:
- **ACID** properties are mainly concerned with single-node (or single-database) transactions, ensuring reliability and integrity.

- **CAP theorem** deals with distributed systems, emphasizing the trade-offs between consistency, availability, and partition tolerance.

In practical terms, designers and engineers often need to balance these principles based on the needs of their application. Some applications may prioritize data consistency and be willing to sacrifice availability during network iss at times.

network measure

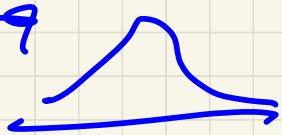
- Degree
- Avg path
- Clustering coefficient

1 = เกาะกลุ่มดี

0 = ไม่

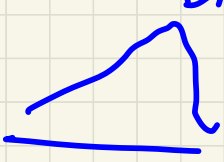
→ Random network Dist

- เหมือน random →



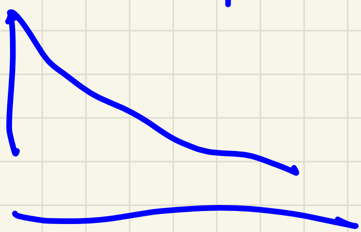
- เกาะกลุ่ม

→ Small-world network Dist



→ Scale-Free

โอกาสที่ node ใหญ่ จะเกิด
กับคนที่เก่งมาก



Central

Scale-Free degree central Random

Small closeness central Random

Small Betwen central
(ดูว่าในpath ผ่านจุด

- Eigenvector (เหมือนศักย์ไว

CSV = import spreadsheet

statistics → avg path length → Dist
(v07) → avg Degree

Modularity (v07)