

CAPP
30122
March 8

OUTSIDE TRADER

Hyun Ki Kim

Jessica Jee Yoon Song

Weiwei Zheng

Outside trader

1.Trading strategy: {"Ki": "1st year MACSS"}

- Motivation
- Web scraping

2. Modeling: {"Weiwei": "1st year MACSS"}

- Machine learning
- Model selection

3. Demo: {"Jessica": "1st year CAPP"}

- Django

Motivation

Goal:

- Building an algorithmic trading software

Insider trading:

- Easiest way to make money
- Illegal to reveal price sensitive information
- Secret may be revealed by collective search

Search ranking:

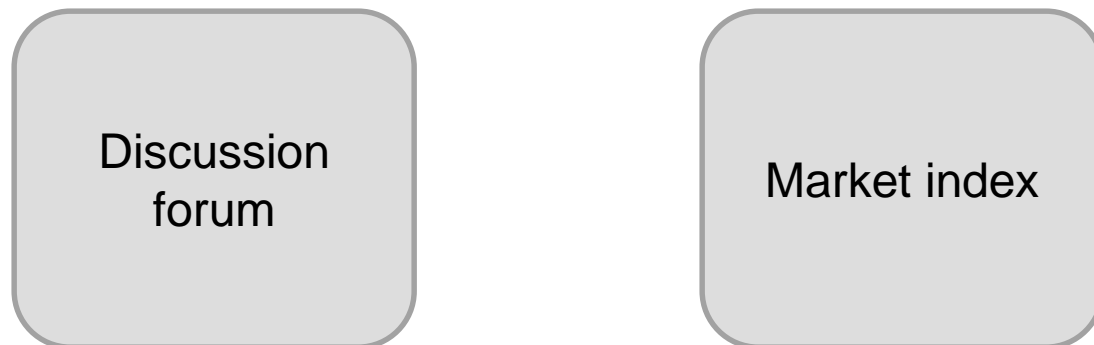
- Preceding indicator for stock price
- Need real time web scraping
- Naver Finance (Korea-based web portal)

Web scraping

Filtering process (overnight)



Real-time scraping (9:01am ~ 3:30 pm)



CRON

APScheduler

- Executes job function periodically

```
from apscheduler.schedulers.blocking import BlockingScheduler

def job_function:
    """
    define the task you wanna run.
    """

    print("hello world!")

    return

sched = BlockingScheduler()
# set the parameters as you want
sched.add_job(job_function, 'cron', month='2', day='7-28', hour='9-12', \
              minute='0-59/15')

sched.start()
```

Difficulties

```
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 18:40:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 18:50:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 19:00:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 19:10:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 19:20:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 19:30:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 19:40:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 19:50:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 20:00:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 20:10:00 CST)" skipped: maximum number of running instances reached (1)
Execution of job "job_function (trigger: cron[month='3', day='1-7', hour='0-23']
-03-01 20:20:00 CST)" skipped: maximum number of running instances reached (1)
```


Difficulties



Modeling

Data

- variables

Method

- machine learning
- model selection

Results

- performance of best models
- next step

Variables

Dependent variable

- predicted if price increases by more than 0.33% after an hour
- made prediction every ten minutes

Predictors (in total 164)

- use data at the moment, ten and twenty minutes earlier

Discussion

trend of posts number, unique participants and click, etc.

Price

closing price the day before, price volatility, price trend, difference between selling and buying price of each period, price gap volatility, trade volume trend, etc.

Variables

Dependent variable

- predicted if price increases by thirty percent one hour later
- made prediction every ten minutes

Predictors (in total 164)

- use data at the moment, ten and twenty minutes earlier

Time

different time slots (early morning, morning, lunch, etc.)


Market index

KOSPI and KOSDAQ index and their changes

Different square and interaction terms

Predicting methods

scikit-learn API



scikit-learn
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction.
Activate Windows — Examples
Go to Settings to activate Windows.

Predicting methods

Logistic regression

- *from* sklearn.linear_model *import* LogisticRegression

K Nearest Neighbors (KNN)

- *from* sklearn.neighbors *import* KNeighborsClassifier

Principal component regression (PCR)

- *from* sklearn.decomposition *import* PCA

Partial least squares regression (PLS)

- *from* sklearn.cross_decomposition *import* PLSRegression

Tree-based methods

Support vector machines (SVM)

- *from* sklearn.svm *import* SVC

Model selection

Validation set approach

- *training set: February 14, 20, 21*
- *validation set: February 22, 23, 26*
- *testing set: February 27, 28, March 2, 3, 6, 7*
- *Observations in total: 19786*

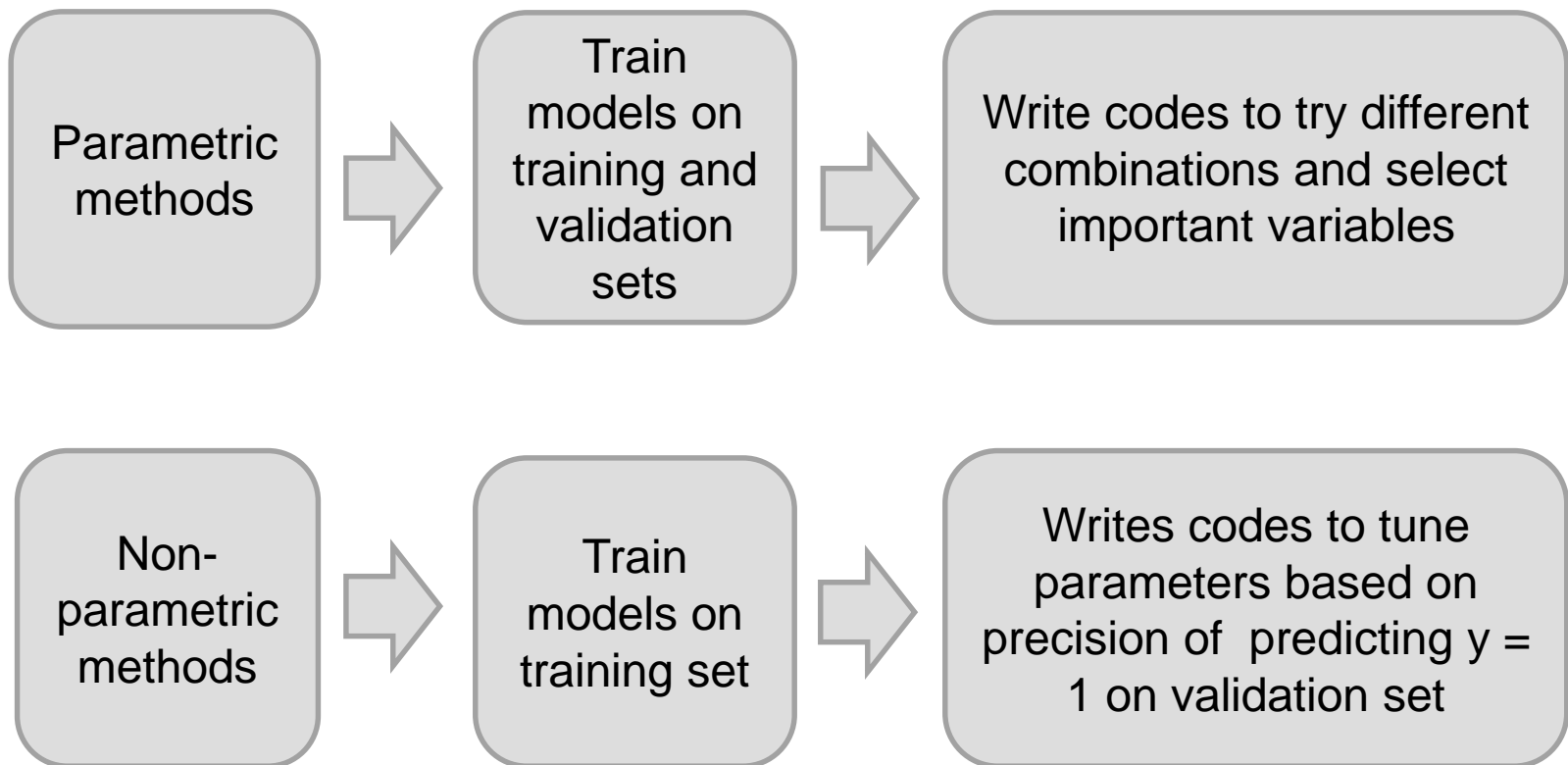
Tuning process

- *Regression vs non-linear models*
- *Penalty for non-linearity*
- *Using mean square error*

from sklearn.metrics import mean_squared_error, classification_report

Model selection

Fitting process



Tree-based methods

decision tree, bagging, random forest, boosting

Decision Tree

- *from* sklearn.tree *import* DecisionTreeClassifier, export_graphviz
- *import* graphviz

Important variables

price volatility: (maximum price – minimum price) / minimum price

average price volatility: price volatility / increase count

alpha: price increase – market increase

variation ten minutes before squared: the squared price variation between ten minutes before and eleven minutes before

Tree-based methods

