A Tutorial on
Predictive Modeling with Python

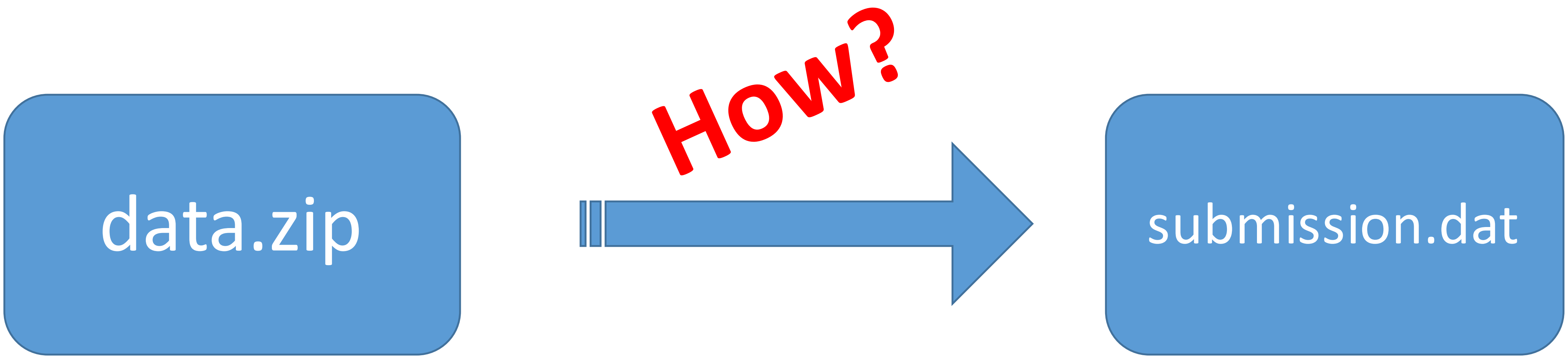
Predictive Modeling Challenge

@Statistical Learning Theory, 2017

Jiuding Duan (TA)

dj@ml.ist.i.kyoto-u.ac.jp

This Tutorial

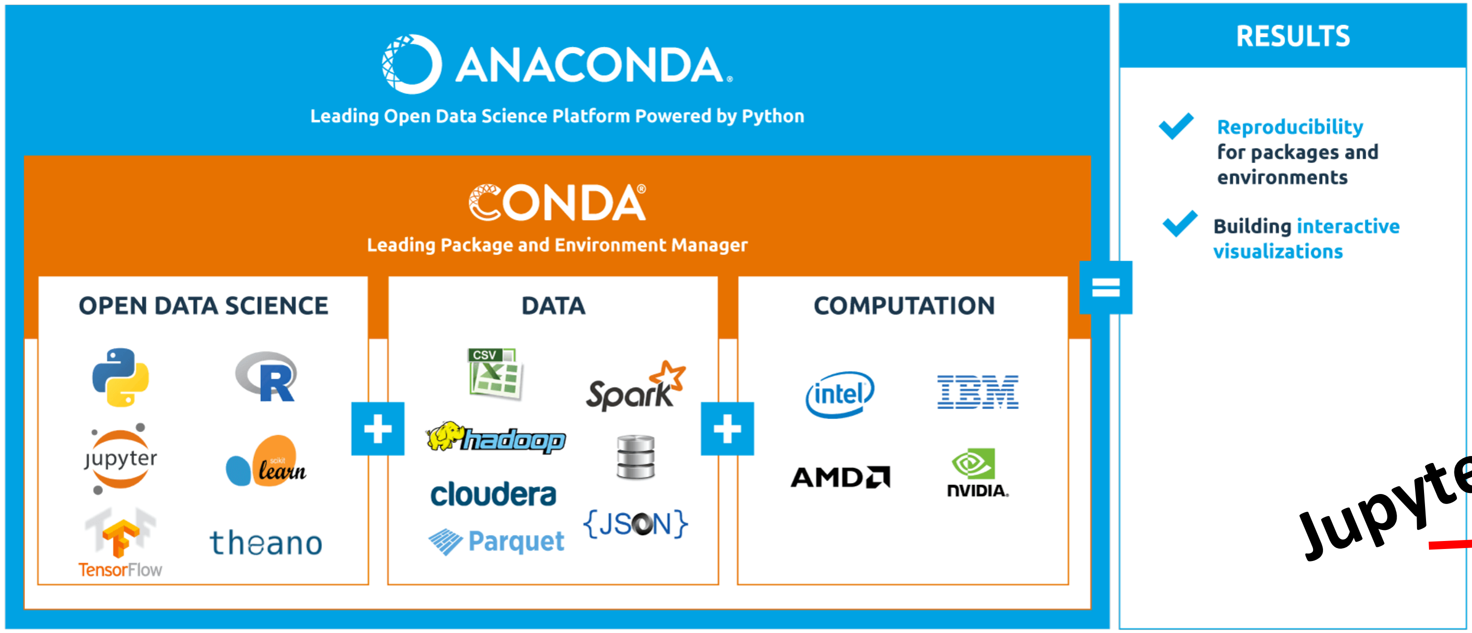


Python

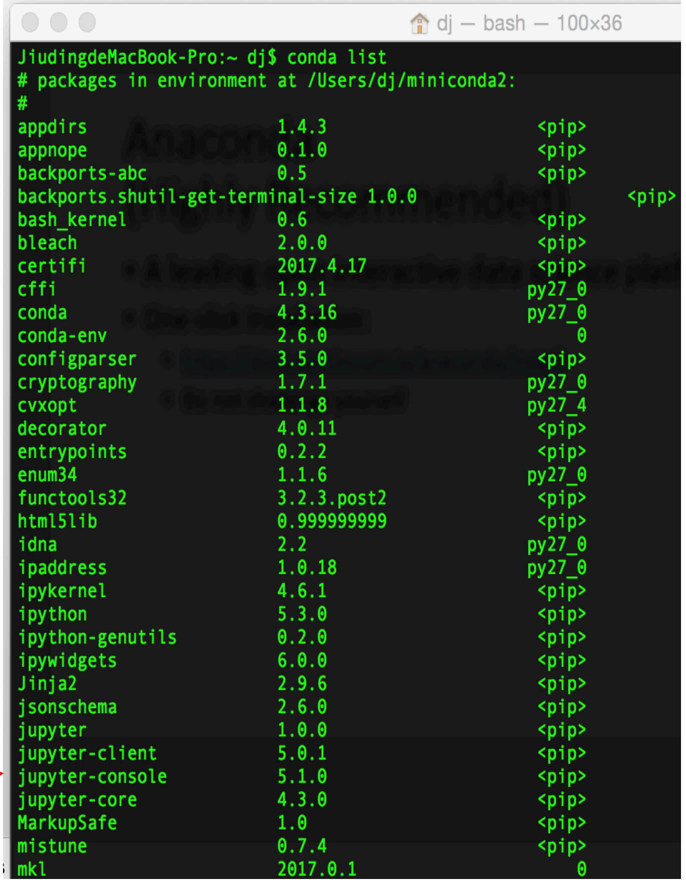
- Install:
  - Windows / Mac OS X/ Linux
  - management tools: pip, homebrew
- Libraries:
  - Numpy, Scipy: for numerical computation
  - Pandas: for data manipulation
  - Matlibplot: for visualization
  - scikit learn: for machine learning
- Simple python tutorial:
  - https://learnxinyminutes.com/docs/python/

Anaconda
(Highly Recommended)

- A leading open interactive data science platform powered by Python
- One-click Installation:
  - https://docs.continuum.io/anaconda/install
  - Do not challenge yourself



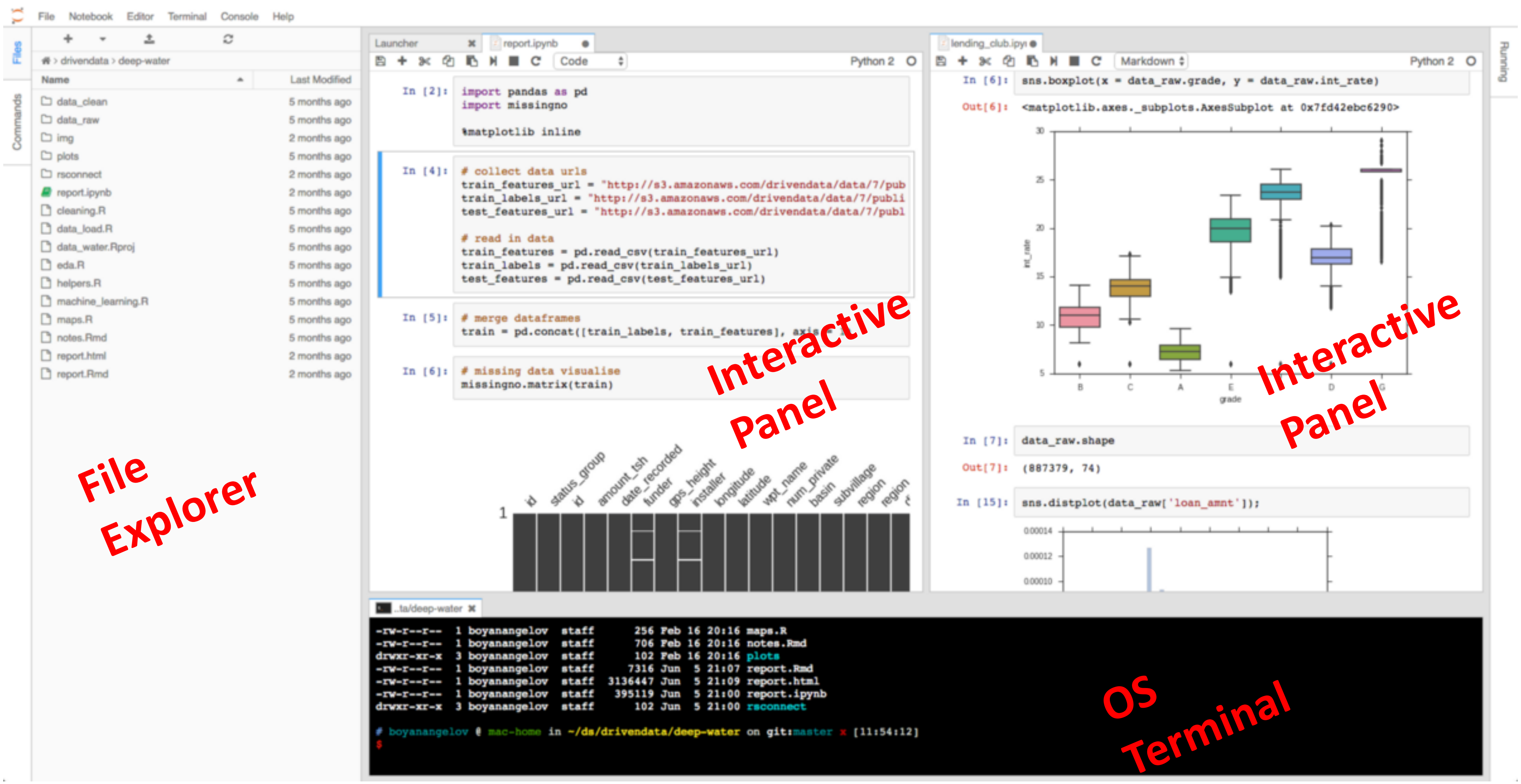
Jupyter



Jupyter lab (Optional)

- An extensible open-source web application for Jupyter notebook

- Installation guide:
  - [Jupyter lab] : https://github.com/jupyterlab/jupyterlab


File Explorer
Interactive Panel
Interactive Panel
OS Terminal

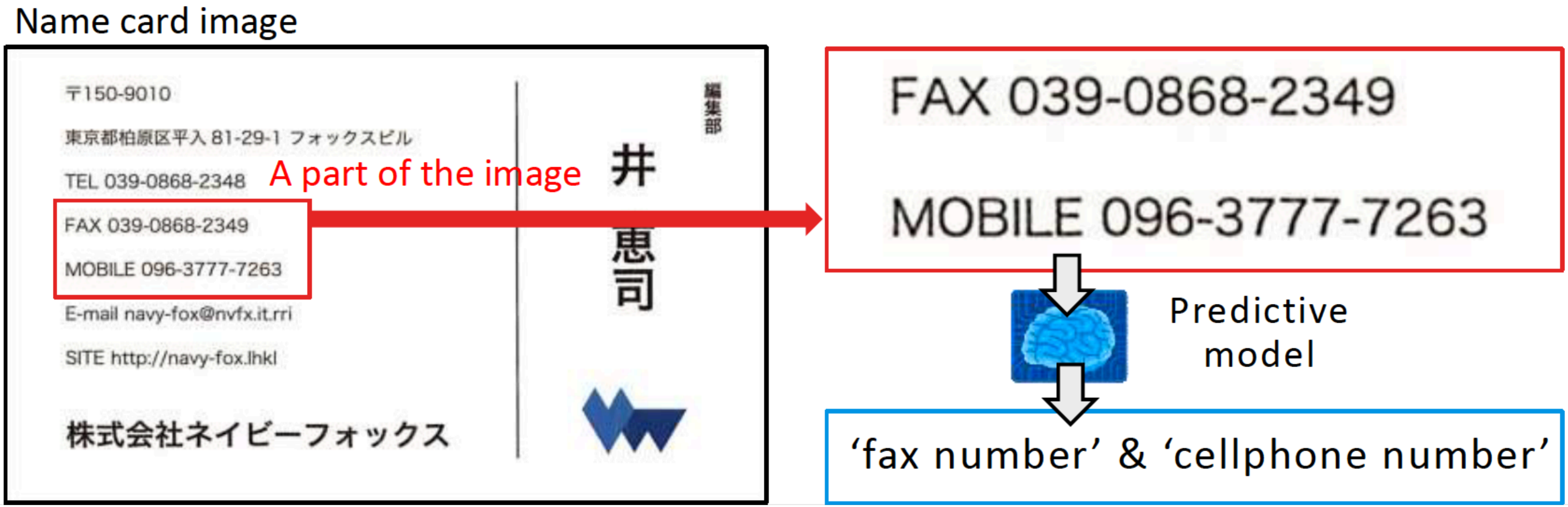
This Tutorial

- The ‘Sansan Data Analysis Challenge’
  - Business card field labeling

- A hand-on Python workflow for
  - basics of predictive modeling
  - construct a basic predictive model pipeline
  - select the best predictive model
  - A hand-on workflow for the above

- See python notebook:
  - [日本語] : http://universityofbigdata.net/competition/tutorial/5723788444434432
  - [English]: http://universityofbigdata.net/competition/tutorial/5723788444434432?lang=en

Multi-label classification



Preparation

$$x = [x_1, x_2, x_3, \ldots, x_d]^T$$

Clean the data - 1

- Load and see what's inside

```
In [3]: df_train.head()
```

Out[3]:

	filename	left	top	right	bottom	company_name	full_name	position_name	address	phone_number	fax	mobil
0	2842.png	491	455	796	485	0	0	0	0	0	0	1
1	182.png	24	858	311	886	0	0	0	0	0	0	1
2	95.png	320	498	865	521	0	0	0	0	0	1	1
3	2491.png	65	39	497	118	1	0	0	0	0	0	0
4	3301.png	271	83	333	463	0	1	1	0	0	0	0

- size
- ranges of variables

```
In [4]: df_train.shape
Out[4]: (25357, 14)
```

Clean the data - 2

- Zoom-in a single sample

- the meaning of each attribute
  - X
  - y

In [5]:
```
row = df_train.iloc[0, :]
print row
```

```
filename         2842.png
left                  491
top                   455
right                 796
bottom                485
company_name            0
full_name               0
position_name           0
address                 0
phone_number            0
fax                     0
mobile                  1
email                   0
url                     0
Name: 0, dtype: object
```

x

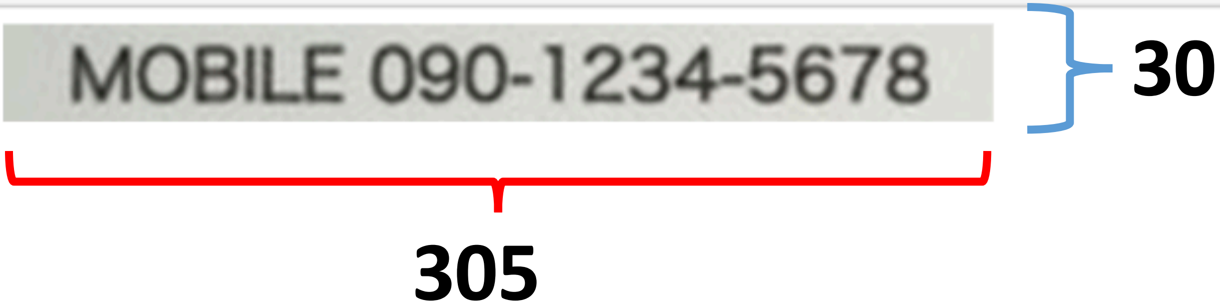
y

Clean the data - 3

- Extract the useful part

In [6]:
```python
DIR_IMAGES = 'images'
img = Image.open(os.path.join(DIR_IMAGES, row.filename))
img = img.crop((row.left, row.top, row.right, row.bottom))
img
```

Out[6]:



- General treatment in computer vision :
    - Flatten it into 1 dimension
    - But, the clipped images are in different scale…

Generate feature vectors - 1

- Resize the image into 100 * 100

$$x = [x_1, x_2, x_3, ..., x_d]^T$$

In [12]:
```
IMG_SIZE = 100
img = img.resize((IMG_SIZE, IMG_SIZE), resample=Image.BICUBIC)
img
```

Out[12]:

100

100

Apply this to all the images.

Generate feature vectors - 2

- Normalize the entries

```
In [14]: x

Out[14]: array([[ 204.,   203.,   203., ...,   222.,   223.,   223.],
                [ 204.,   203.,   203., ...,   222.,   223.,   223.],
                [ 204.,   203.,   203., ...,   222.,   223.,   223.],
                ...,
                [ 204.,   204.,   205., ...,   223.,   223.,   224.],
                [ 204.,   204.,   205., ...,   223.,   223.,   224.],
                [ 204.,   204.,   205., ...,   223.,   223.,   224.]])
```

Generate feature vectors - 2

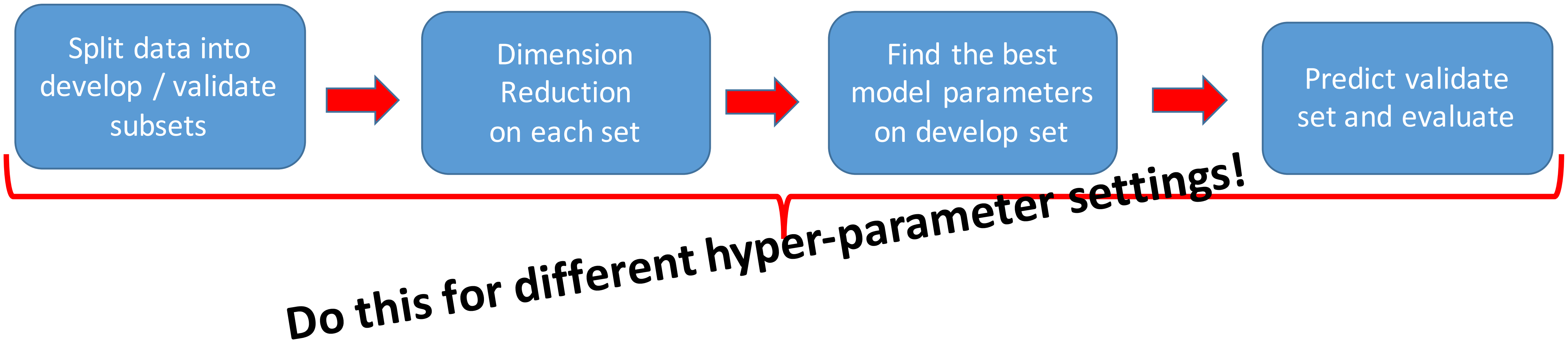
- After normalization, all entries are within [0, 1]

```
In [15]: x = (x - np.min(x)) / (np.max(x)-np.min(x))
         x

Out[15]: array([[ 0.82513661,  0.81967213,  0.81967213, ...,  0.92349727,
                  0.92896175,  0.92896175],
                [ 0.82513661,  0.81967213,  0.81967213, ...,  0.92349727,
                  0.92896175,  0.92896175],
                [ 0.82513661,  0.81967213,  0.81967213, ...,  0.92349727,
                  0.92896175,  0.92896175],
                ...,
                [ 0.82513661,  0.82513661,  0.83060109, ...,  0.92896175,
                  0.92896175,  0.93442623],
                [ 0.82513661,  0.82513661,  0.83060109, ...,  0.92896175,
                  0.92896175,  0.93442623],
                [ 0.82513661,  0.82513661,  0.83060109, ...,  0.92896175,
                  0.92896175,  0.93442623]])
```

Training predictive model

- Input:
  - (X_develop, Y_develop), (X_validate, Y_validate)
  - Hyper-parameter of Model/Pipeline
    - i.e. the weight of regularization term, the bandwidth of Gaussian kernels, etc.
  - Model parameters
    - i.e. the coefficients in linear regression or SVMs



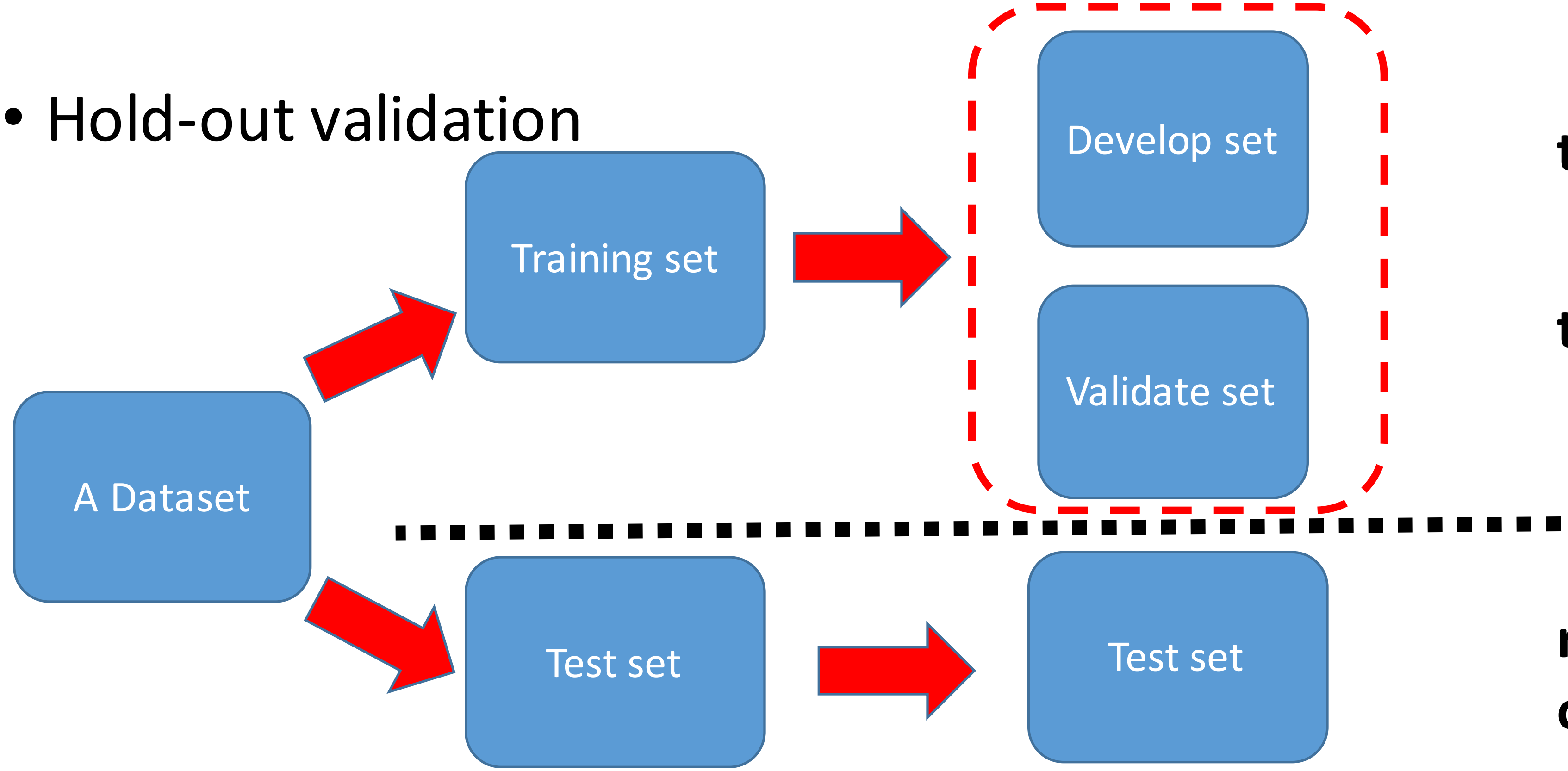
Do not let your training process see what it will test.

That is cheating.

The predictive model will be meaningless.

Splitting Data

- Hold-out validation



train on Dev set

test on Val set

report final prediction on Test set

- Two frequent mistakes:
  - 1 . learn and predict on the same dataset (over-fitting)
  - 2. expose the test data during training

Split training data into Dev / Val subsets

- 80% develop, 20% validate

In [22]:
```
from sklearn.model_selection import train_test_split
X_dev, X_val, Y_dev, Y_val = train_test_split(X_train, Y_train, train_size=0.8,
random_state=0)
```

The total 500 training data has been splited into 400 as development set and 100 as evaluation set.

In [23]:
```
print X_dev.shape, Y_dev.shape
print X_val.shape, Y_val.shape
```

```
(400, 10000) (400, 9)
(100, 10000) (100, 9)
```

Curse of dimensionality

- Data lies in a low dimensional subspace
- Axes of this subspace are more effective indicators

- Need for dimension reduction
  - discover hidden correlations
  - remove redundant features
  - interpretation and visualization
  - easier storage and processing

5 dimensional?

No

	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	0	0	4	4
Jenny	0	0	0	5	5
Jane	0	0	0	2	2

Find the genuine dimension

- Rank = 2 < 5
  - Joe : [1 1 1 0 0] = 1 * [1 1 1 0 0]
  - Jim : [3 3 3 0 0] = 3 * [1 1 1 0 0]
  - John : [4 4 4 0 0] = 4 * [1 1 1 0 0]

  - Jill : [0 0 0 4 4] = 4 *[0 0 0 1 1]
  - Jenny : [0 0 0 5 5] = 5 *[0 0 0 1 1]

	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	0	0	4	4
Jenny	0	0	0	5	5
Jane	0	0	0	2	2

- The genuine dimension is 2!
  - A method to find an efficient projection: Principal Component Analysis (PCA)

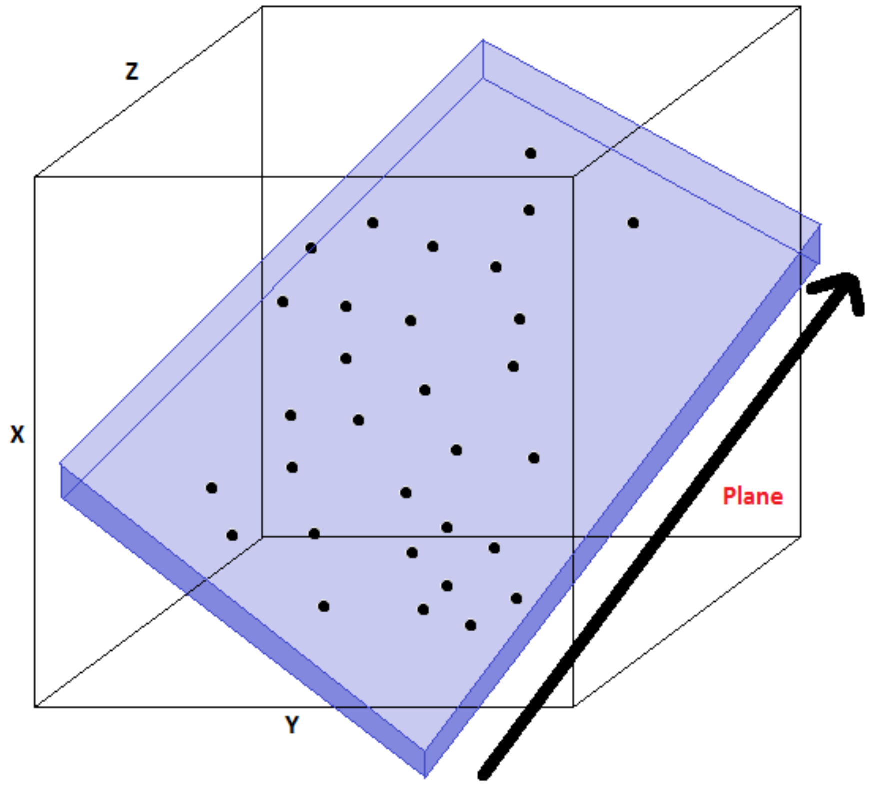
Principal component analysis (PCA)

$$X_{lowDim} = \underline{W}^T X_{highDim}$$

PCA tells you a good *W*

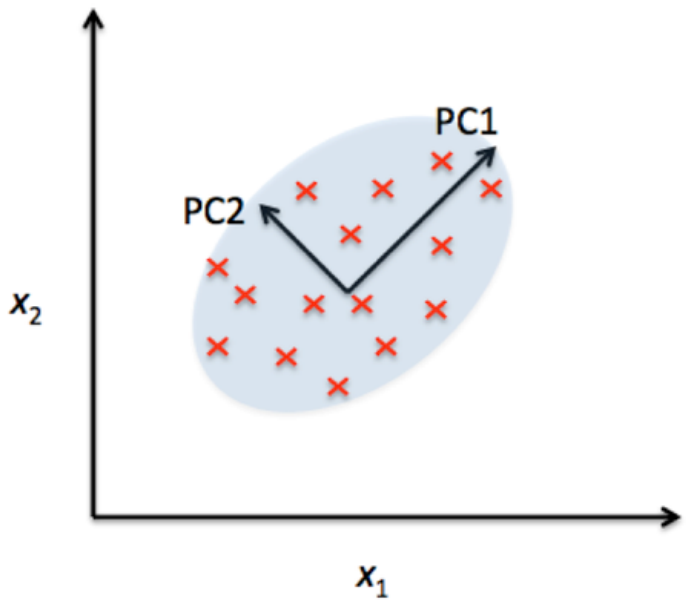
- A linear projection
  - from coordinate system [1 0 0], [0 1 0], [0 0 1]
  - to new coordinate system [1 2 1] [-2 -3 1]

- Project a sample linearly:
  - from x_a = [1 2 1]
  - to x_a_pca = [1 0]



Dimension reduction

- Principal Component Analysis (PCA)




```
In [24]: from sklearn.decomposition import PCA
         decomposer = PCA(n_components=10, random_state=0)
         decomposer.fit(X_dev)

Out[24]: PCA(copy=True, iterated_power='auto', n_components=10, random_state=0,
           svd_solver='auto', tol=0.0, whiten=False)
```

We can apply PCA as a method for dimension reduction, on both development set X_dev and X_dev_pca and X_val_pca. Specifically, we use decomposer.transform(X).

```
In [25]: X_dev_pca = decomposer.transform(X_dev)
         X_val_pca = decomposer.transform(X_val)
```

Project, don't do PCA again!

X_dev_pca, X_val_pca are indeed 10 dimensional feature vectors.

```
In [26]: print X_dev_pca.shape
         print X_val_pca.shape

(400, 10)
(100, 10)
```

Classification

- Logistic Regression

(1) define x and y →
(2) define a classifier →
(3) fit it →

```
y = Y_dev[:, j]
classifier = LogisticRegression(penalty='l2', C=0.01)
classifier.fit(X_dev_pca, y)
```

(4) Get a cup of coffee, done!

- Default hyper-parameter:
  - Regularization = L2-norm
  - C = 0.01

Multi-label classification

- A naïve solution:
  - Treat each label independently

loop over all labels

In [27]:
```
from sklearn.linear_model import LogisticRegression

classifiers = []
for j in range(Y_dev.shape[1]):
    y = Y_dev[:, j]
    classifier = LogisticRegression(penalty='l2', C=0.01)
    classifier.fit(X_dev_pca, y)
    classifiers.append(classifier)
```

Make prediction

- For all labels

```
In [28]: Y_val_pred = np.zeros(Y_val.shape)
         for j in range(Y_dev.shape[1]):
             classifier = classifiers[j]
             y = classifier.predict_proba(X_val_pca)[:, 1]
             Y_val_pred[:, j] = y
```

Double check the results

```
In [29]: Y_val_pred.shape

Out[29]: (100, 9)

In [30]: Y_val_pred

Out[30]: array([[ 0.3028575 ,  0.22535636,  0.19299058,  0.22895185,  0.20035639,
                  0.28620408,  0.23830063,  0.65474453,  0.5790035 ],
                [ 0.22904739,  0.24642238,  0.34446014,  0.24784898,  0.58169305,
                  0.66908365,  0.26408045,  0.13458627,  0.16573239],
                [ 0.25088454,  0.18977441,  0.25086316,  0.3291364 ,  0.2580537 ,
                  0.26655858,  0.28930265,  0.52152202,  0.56373377],
                [ 0.32761155,  0.43064428,  0.43842012,  0.21176827,  0.24395992,
                  0.17400567,  0.17683373,  0.43748784,  0.36377521],
                [ 0.23734343,  0.34666775,  0.27659202,  0.47624   ,  0.32644031,
                  0.29955068,  0.40354904,  0.21382397,  0.30855398],
                [ 0.2522688 ,  0.39599945,  0.32287796,  0.32408376,  0.4669947 ,
                  0.31923533,  0.24730576,  0.22885354,  0.27911869],
```

Evaluate the results

- Use Marco-Averaged-AUC

```
In [31]: from sklearn.metrics import roc_auc_score
         roc_auc_score(Y_val, Y_val_pred, average='macro')

Out[31]: 0.7900549356140124
```

Let's make it better!

Simplify the above... (in 3 lines)

```
In [32]: from sklearn.multiclass import OneVsRestClassifier

         classifier = OneVsRestClassifier(LogisticRegression(penalty='l2', C=0.01))
         classifier.fit(X_dev_pca, Y_dev)
         Y_val_pred = classifier.predict_proba(X_val_pca)

In [33]: roc_auc_score(Y_val, Y_val_pred, average='macro')

Out[33]: 0.7900549356140124l
```

the same score as we've achieved!

Tuning Hyper-parameters

- So far we've finished a **pipeline** with fixed hyper-parameters
  - Dimension_PCA = 10
  - Regularization = L2-norm
  - C = 0.01

Let's seek better hyper-parameters!

Split data into develop / validate subsets

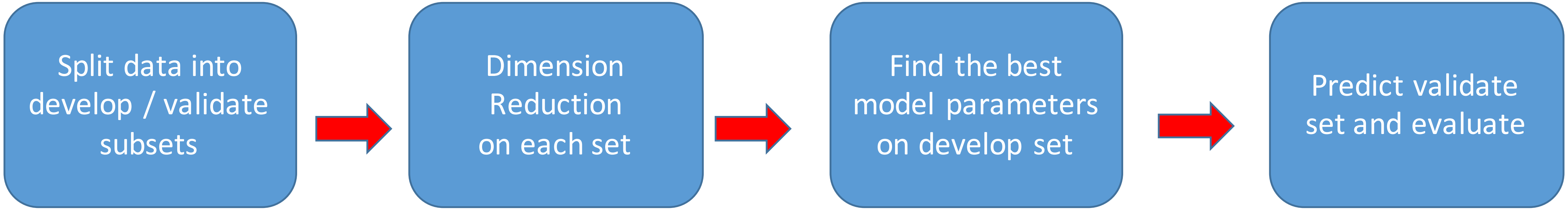
Dimension Reduction on each set

Find the best model parameters on develop set

Predict validate set and evaluate

Package everything

- Package the meta functional module as **Step**:



- Package the above as **Pipeline**

Search best hyper-parameters with 'pipeline'

- Grid Search

We search in:
Parameter:
C = {0.01, 0.1, 1.0, 10, 100}

In [35]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer

params = {'classifier__estimator__C': [0.01, 0.1, 1.0, 10., 100.]}
scorer = make_scorer(roc_auc_score, average='macro', needs_proba=True)

predictor = GridSearchCV(pipeline, params, cv=5, scoring=scorer)
```

- Result:

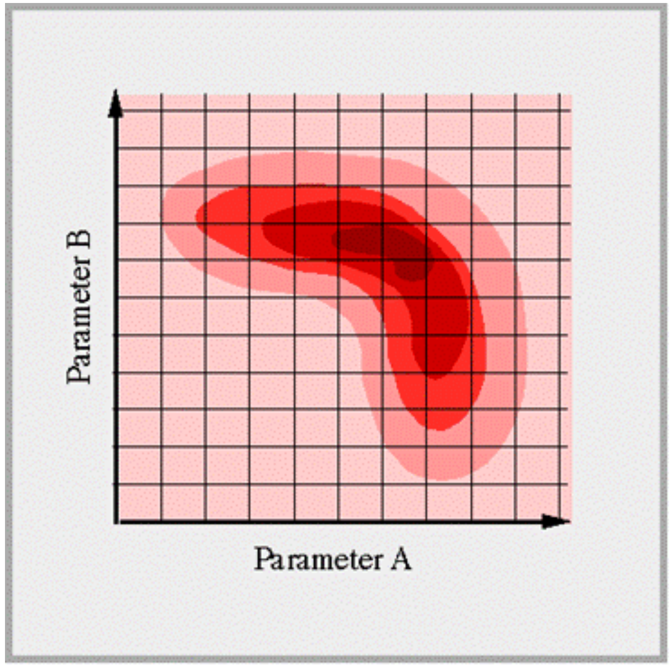
Score Improved !!!

In [38]:
```
Y_val_pred = predictor.predict_proba(X_val)
roc_auc_score(Y_val, Y_val_pred, average='macro')
```

Out[38]: 0.7924763556529914

Search best hyper-parameters with 'pipeline'

- Grid Search



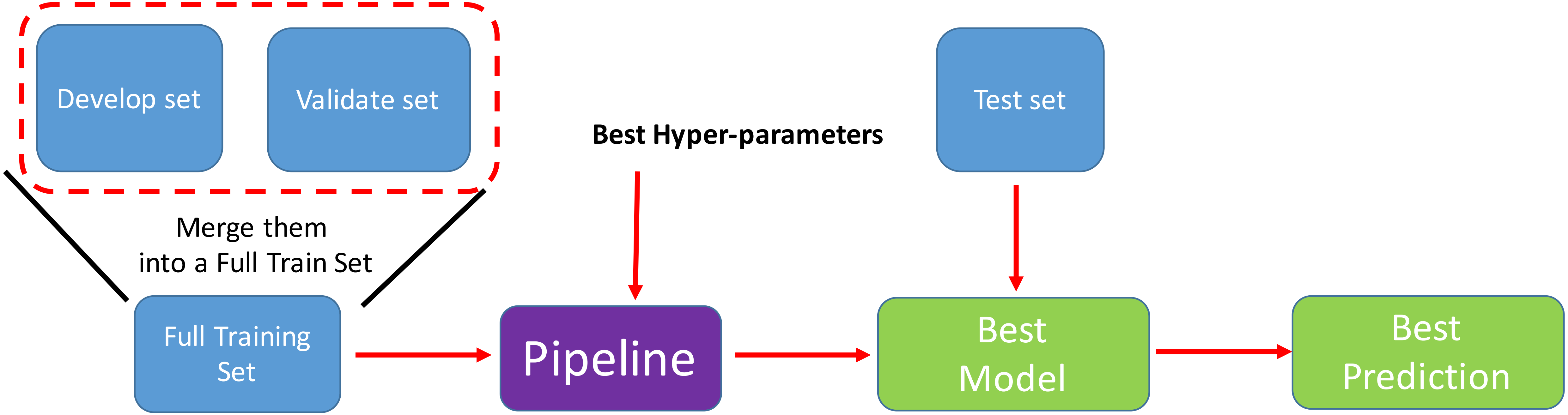
We search in:
Parameter A:
C = {0.01, 0.1, 1.0, 10, 100}
Parameter B:
Dimension_PCA = {10, 20, 50}

- Best Hyperparameters: C = 0.1, Dimension_PCA = 50
- Best Marco_Averaged_AUC score: 0.8546

Further improved !!!

Submission

- We found the best hyper-parameters
- But the training data was not fully exploited, so let's retrain.



Keep in mind

- Start with simple stuffs
  - i.e. Try a reliable tool first before moving on to advanced things
- Create a pipeline
  - See 'Cross-Validation' if you want to make the search for hyper-parameters more reliable
- Incrementally improve