

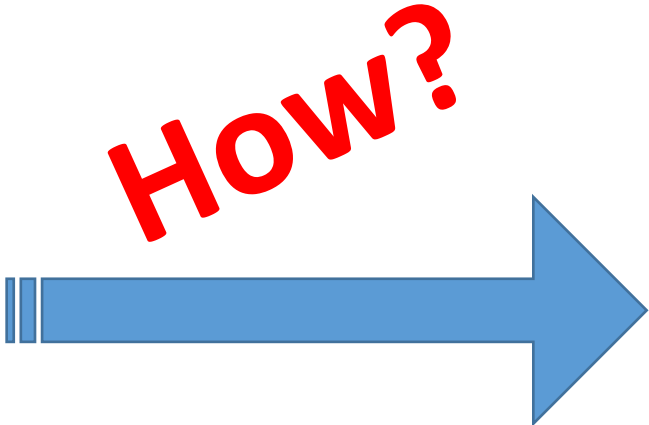
A Tutorial on Predictive Modeling with Python

Predictive Modeling Challenge
@Statistical Learning Theory, 2017

Jiuding Duan (TA)
dj@ml.ist.i.kyoto-u.ac.jp

This Tutorial

data.zip



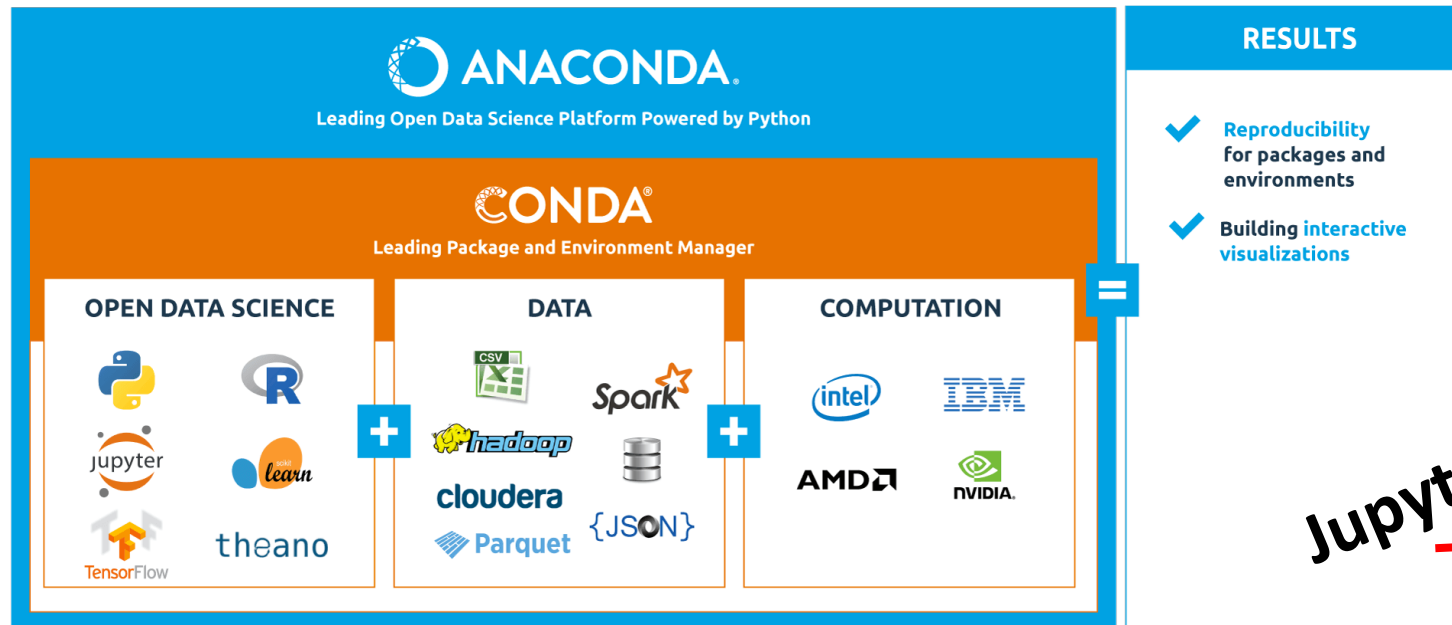
submission.dat

Python

- Install:
 - Windows / Mac OS X/ Linux
 - management tools: pip, homebrew
- Libraries:
 - Numpy, Scipy: for numerical computation
 - Pandas: for data manipulation
 - Matplotlib: 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



```
JiudingdeMacBook-Pro:~ dj$ conda list
# packages in environment at /Users/dj/miniconda2:
#
appdirs                1.4.3                <pip>
appnope                0.1.0                <pip>
backports-abc          0.5                  <pip>
backports.shutil-get-terminal-size 1.0.0                <pip>
bash_kernel            0.6                  <pip>
bleach                 2.0.0                <pip>
certifi                2017.4.17            <pip>
cffi                   1.9.1                py27_0
conda                  4.3.16               py27_0
conda-env              2.6.0                _0
configparser           3.5.0                <pip>
cryptography           1.7.1                py27_0
cvxopt                 1.1.8                py27_4
decorator              4.0.11               <pip>
entrypoints            0.2.2                <pip>
enum34                 1.1.6                py27_0
functools32           3.2.3.post2         <pip>
html5lib               0.999999999         <pip>
idna                   2.2                  py27_0
ipaddress              1.0.18               py27_0
ipykernel              4.6.1                <pip>
ipython                5.3.0                <pip>
ipython-genutils      0.2.0                <pip>
ipywidgets             6.0.0                <pip>
Jinja2                 2.9.6                <pip>
jsonschema             2.6.0                <pip>
jupyter                1.0.0                <pip>
jupyter-client         5.0.1                <pip>
jupyter-console        5.1.0                <pip>
jupyter-core           4.3.0                <pip>
MarkupSafe             1.0                  <pip>
mistune                0.7.4                <pip>
mkl                    2017.0.1             _0
```

Jupyter lab (Optional)

- An extensible open-source web application for Jupyter notebook
- Installation guide:
 - [Jupyter lab] : <https://github.com/jupyterlab/jupyterlab>

File Notebook Editor Terminal Console Help

Files

Commands

> drivendata > deep-water

Name	Last Modified
data_clean	5 months ago
data_raw	5 months ago
img	2 months ago
plots	5 months ago
rconnect	2 months ago
report.ipynb	2 months ago
cleaning.R	5 months ago
data_load.R	5 months ago
data_water.Rproj	5 months ago
eda.R	5 months ago
helpers.R	5 months ago
machine_learning.R	5 months ago
maps.R	5 months ago
notes.Rmd	5 months ago
report.html	2 months ago
report.Rmd	2 months ago

File Explorer

Launcher x report.ipynb Python 2

```

In [2]: import pandas as pd
import missingno

%matplotlib inline

In [4]: # collect data urls
train_features_url = "http://s3.amazonaws.com/drivendata/data/7/pub
train_labels_url = "http://s3.amazonaws.com/drivendata/data/7/publi
test_features_url = "http://s3.amazonaws.com/drivendata/data/7/publ

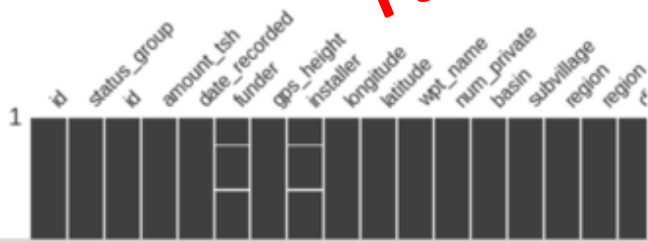
# read in data
train_features = pd.read_csv(train_features_url)
train_labels = pd.read_csv(train_labels_url)
test_features = pd.read_csv(test_features_url)

In [5]: # merge dataframes
train = pd.concat([train_labels, train_features], axis=1)

In [6]: # missing data visualise
missingno.matrix(train)

```

Interactive Panel



```

./a/deep-water x
-rw-r--r-- 1 boyanangelov staff 256 Feb 16 20:16 maps.R
-rw-r--r-- 1 boyanangelov staff 706 Feb 16 20:16 notes.Rmd
drwxr-xr-x 3 boyanangelov staff 102 Feb 16 20:16 plots
-rw-r--r-- 1 boyanangelov staff 7316 Jun 5 21:07 report.Rmd
-rw-r--r-- 1 boyanangelov staff 3136447 Jun 5 21:09 report.html
-rw-r--r-- 1 boyanangelov staff 395119 Jun 5 21:00 report.ipynb
drwxr-xr-x 3 boyanangelov staff 102 Jun 5 21:00 rconnect

# boyanangelov @ mac-home in ~/ds/drivendata/deep-water on git:master x [11:54:12]
$

```

OS Terminal

lending_club.ipynb Python 2

```

In [6]: sns.boxplot(x = data_raw.grade, y = data_raw.int_rate)

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd42ebc6290>

```

```

In [7]: data_raw.shape

Out[7]: (887379, 74)

In [15]: sns.distplot(data_raw['loan_amnt']);

```

Interactive Panel

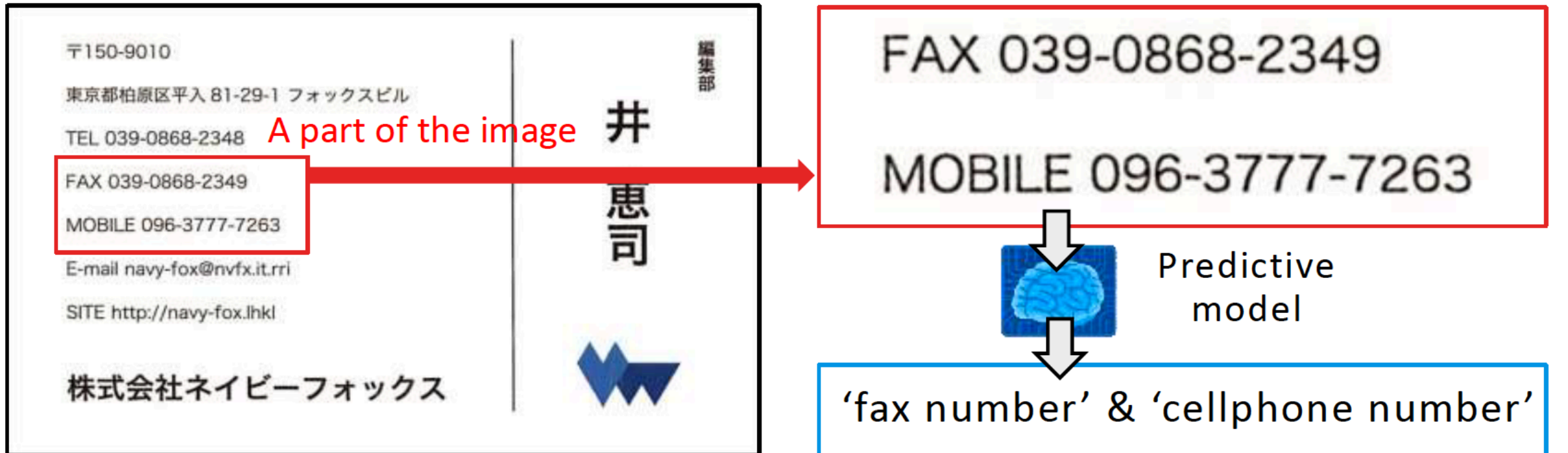
Running

This Tutorial

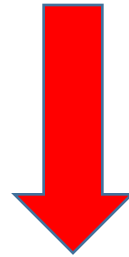
- The 'Sansan Data Analysis Challenge'
 - Business card field labeling
- A hand-on Python workflow for
 - basics of predictive modeling
 - construct a basic predictive model pipeline
 - select the best predictive model
 - A hand-on workflow for the above
- See python notebook:
 - [日本語] : <http://universityofbigdata.net/competition/tutorial/5723788444434432>
 - [English]:
<http://universityofbigdata.net/competition/tutorial/5723788444434432?lang=en>

Multi-label classification

Name card image



Preparation



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

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
```

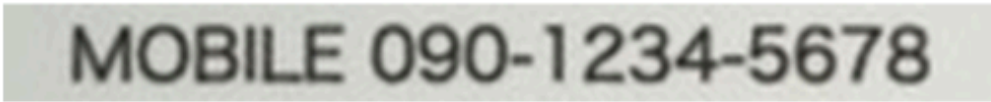
filename	2842.png	}	X
left	491		
top	455		
right	796		
bottom	485		
company_name	0	}	y
full_name	0		
position_name	0		
address	0		
phone_number	0		
fax	0		
mobile	1		
email	0		
url	0		
Name: 0, dtype: object			

Clean the data - 3

- Extract the useful part

```
In [6]: 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]:



MOBILE 090-1234-5678

} 30

└──────────────────┘
305

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

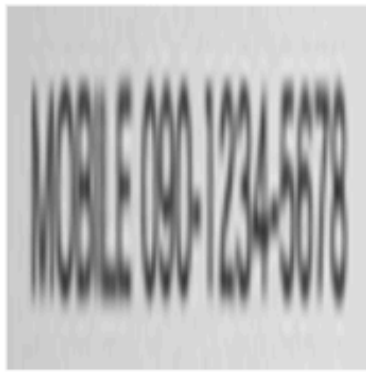
Generate feature vectors - 1

- Resize the image into 100 * 100

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

```
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

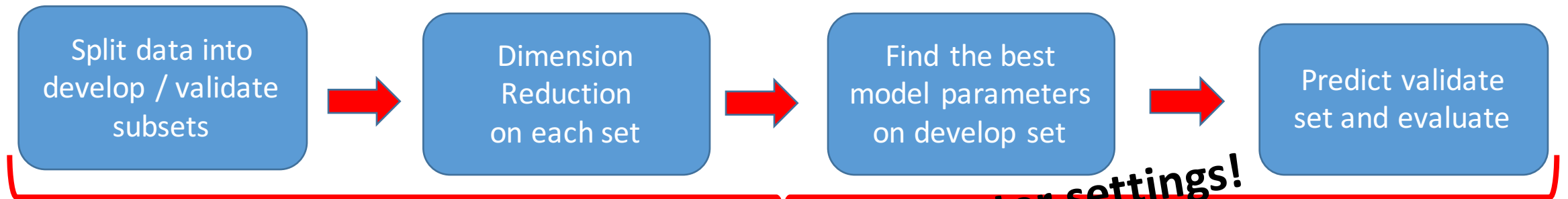
- After normalization, all entries are within [0, 1]

```
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:
 - $(X_{\text{develop}}, Y_{\text{develop}}), (X_{\text{validate}}, Y_{\text{validate}})$
 - 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 this for different hyper-parameter settings!

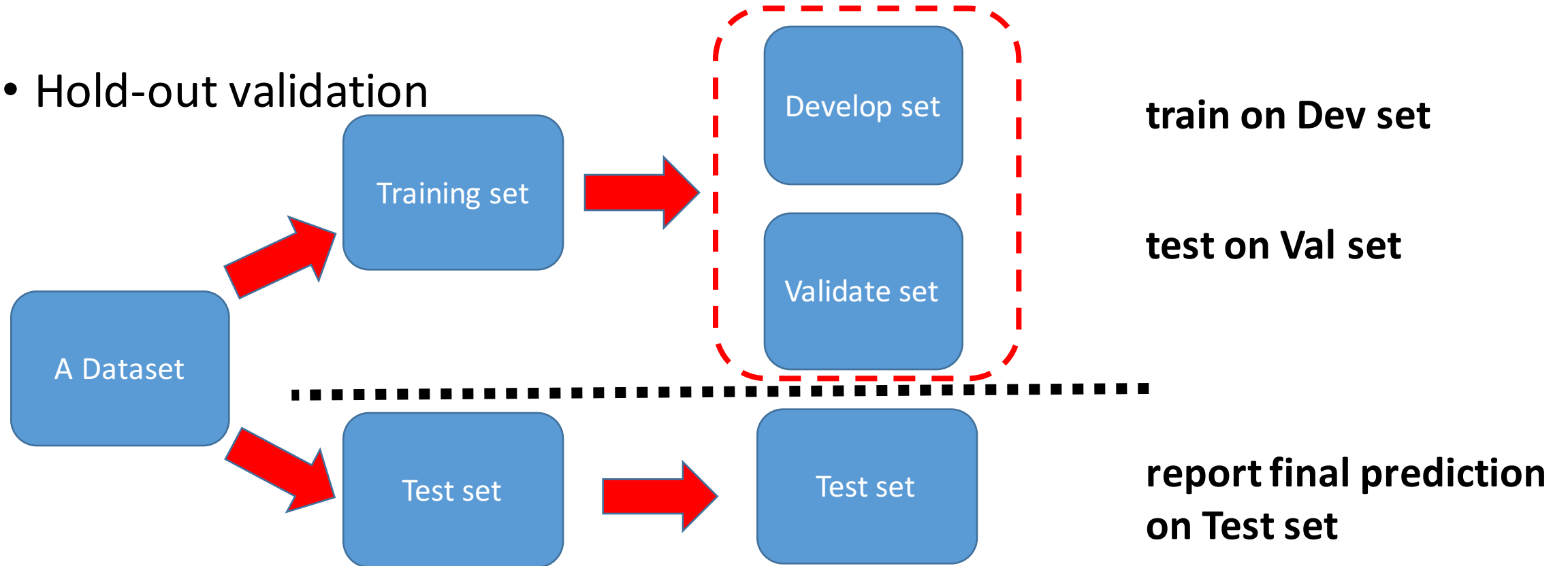
Do not let your training process see what it will test.

That is cheating.

The predictive model will be meaningless.

Splitting Data

- Hold-out validation



- 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 : $[1\ 1\ 1\ 0\ 0] = 1 * \underline{[1\ 1\ 1\ 0\ 0]}$
 - Jim : $[3\ 3\ 3\ 0\ 0] = 3 * \underline{[1\ 1\ 1\ 0\ 0]}$
 - John : $[4\ 4\ 4\ 0\ 0] = 4 * \underline{[1\ 1\ 1\ 0\ 0]}$
 - Jill : $[0\ 0\ 0\ 4\ 4] = 4 * \underline{[0\ 0\ 0\ 1\ 1]}$
 - Jenny : $[0\ 0\ 0\ 5\ 5] = 5 * \underline{[0\ 0\ 0\ 1\ 1]}$

	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

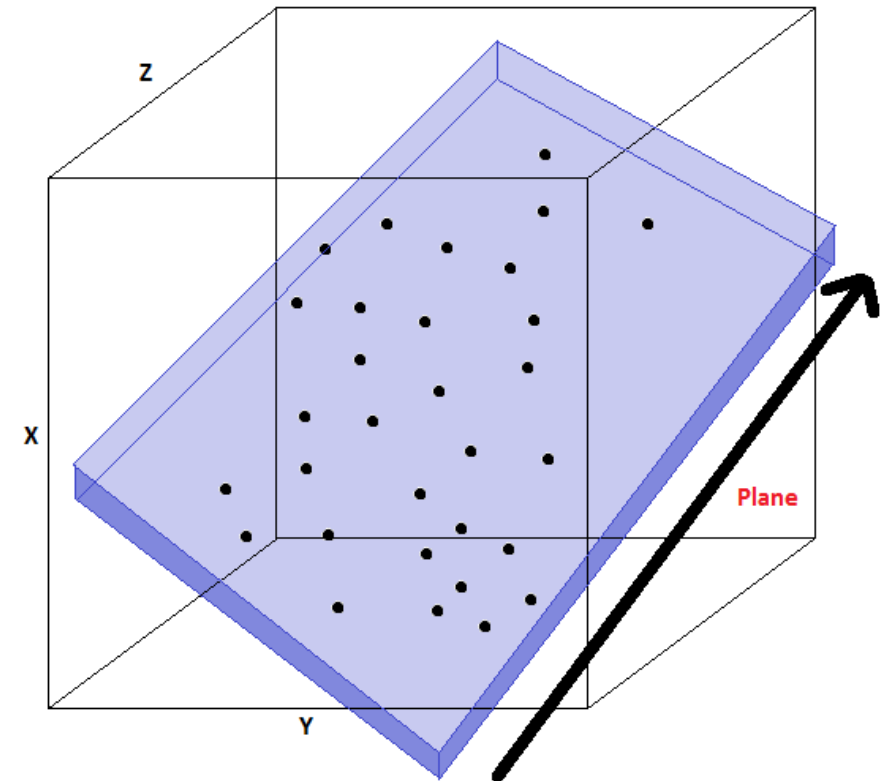
- The genuine dimension is 2!
 - A method to find an efficient projection: Principal Component Analysis (PCA)

Principal component analysis (PCA)

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

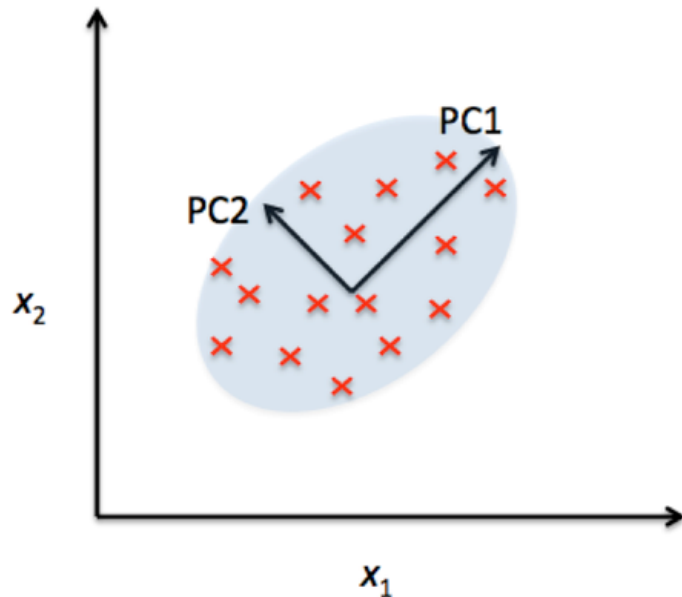
PCA tells you a good W

- 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_{val} 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)
```

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)
```

Project, don't do PCA again!

Classification

- Logistic Regression

(1) define x and y → `y = Y_dev[:, j]`
(2) define a classifier → `classifier = LogisticRegression(penalty='l2', C=0.01)`
(3) fit it → `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 Macro-Averaged-AUC

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

```
Out[31]: 0.79005493561401241
```

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]: 0.79005493561401241
```

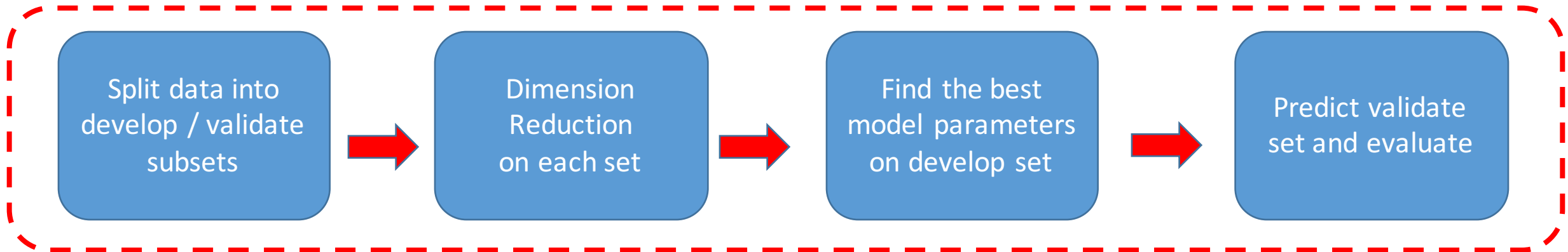


the same score as we've achieved!

Tuning Hyper-parameters

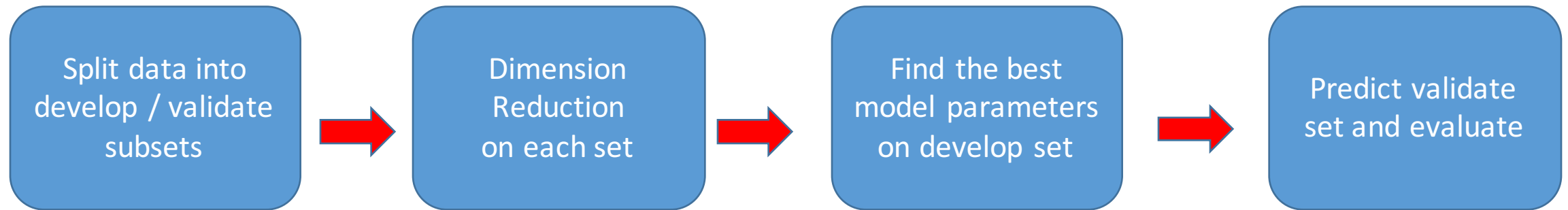
- So far we've finished a **pipeline** with fixed hyper-parameters
 - Dimension_PCA = 10
 - Regularization = L2-norm
 - C = 0.01

Let's seek better hyper-parameters!



Package everything

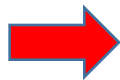
- Package the meta functional module as **Step**:



-

Package the above as **Pipeline**

Input:
Hyper-parameters



Output:
- best score
- best model
- best hyper-parameters

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:

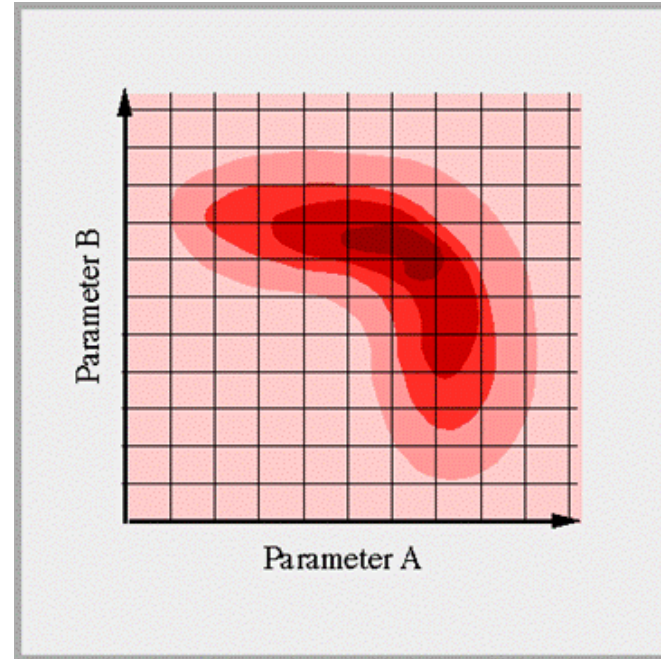
Score Improved !!!

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

→ Out[38]: 0.79247635565299146

Search best hyper-parameters with 'pipeline'

- Grid Search



We search in:

Parameter A:

$C = \{0.01, 0.1, 1.0, 10, 100\}$

Parameter B:

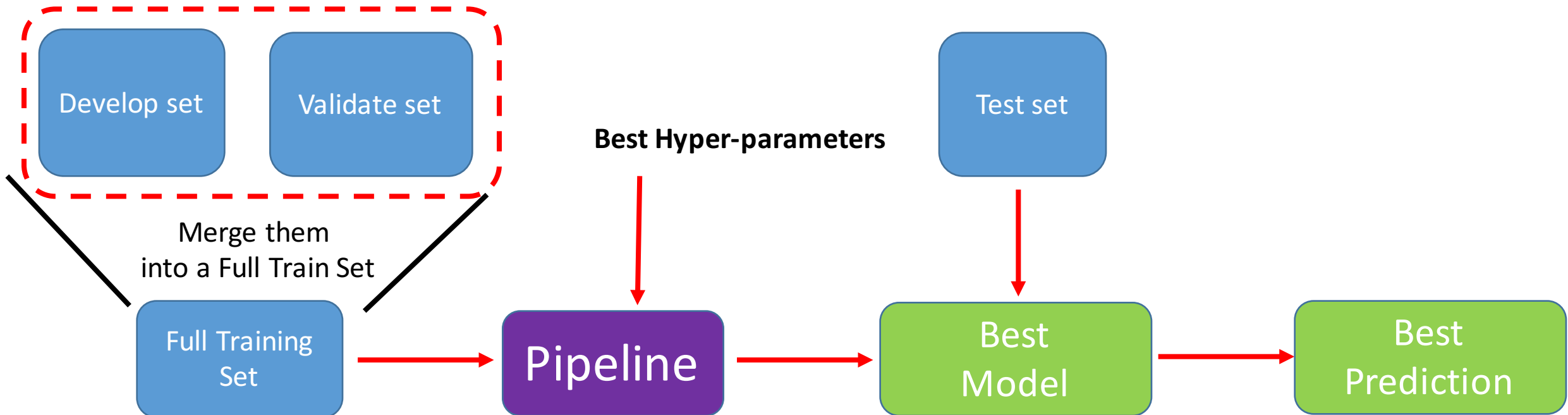
$\text{Dimension_PCA} = \{10, 20, 50\}$

- Best Hyperparameters: $C = 0.1$, $\text{Dimension_PCA} = 50$
- Best Macro_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