From workflows to pipelines

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor

Revisiting our workflow

```
from sklearn.ensemble import RandomForestClassifier as rf
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

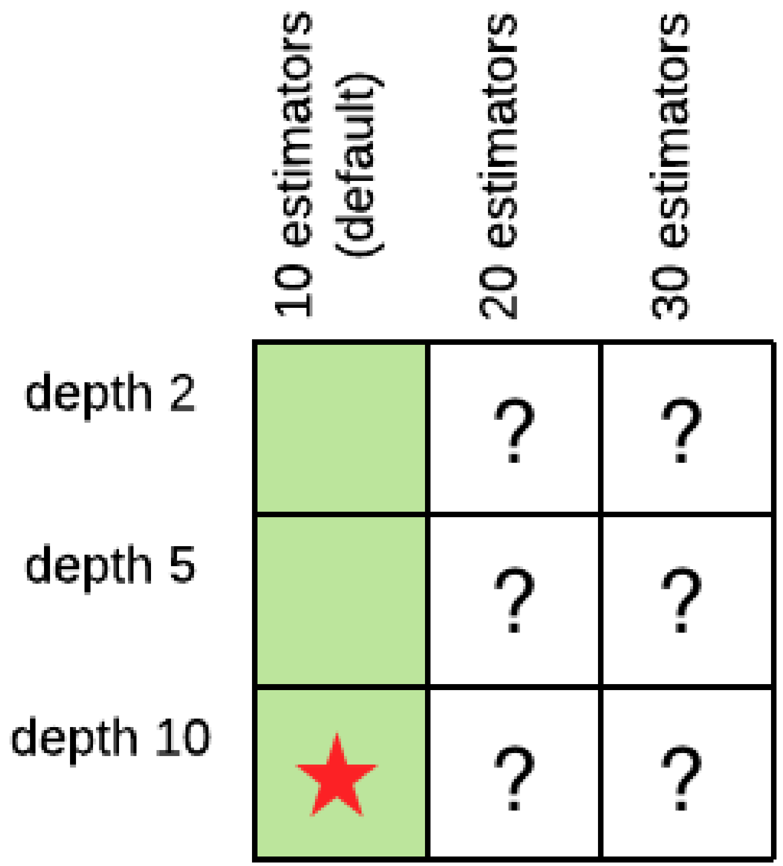
```
grid_search = GridSearchCV(rf(), param_grid={'max_depth': [2, 5, 10]})
grid_search.fit(X_train, y_train)
depth = grid_search.best_params_['max_depth']
```

```
vt = SelectKBest(f_classif, k=3).fit(X_train, y_train)
clf = rf(max_depth=best_value).fit(vt.transform(X_train), y_train)
accuracy_score(clf.predict(vt.transform(X_test), y_test))
```

The power of grid search

Optimize `max_depth` :

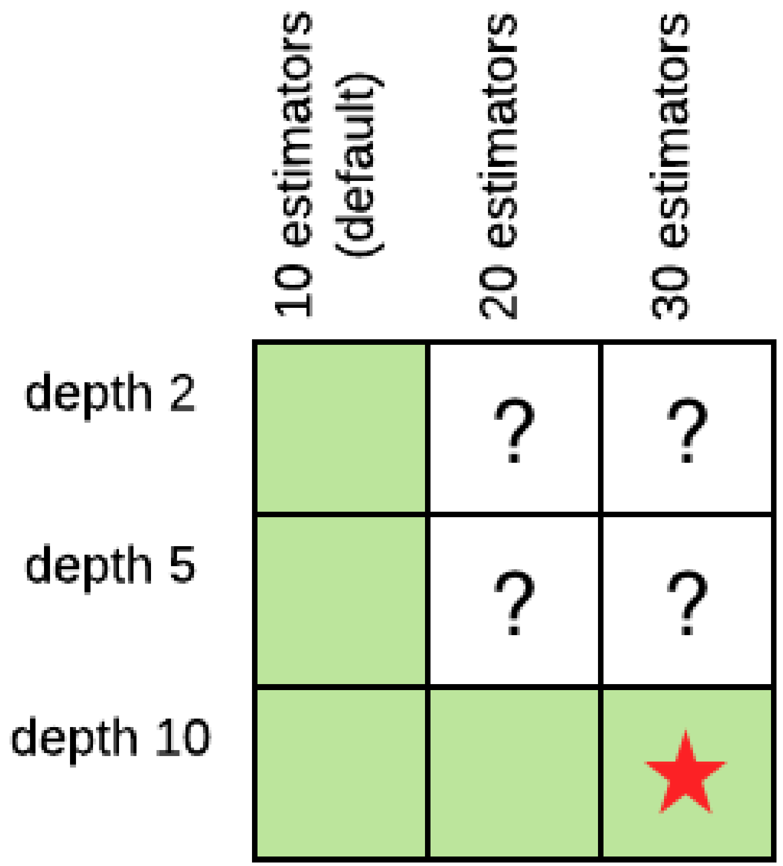
```
pg = {'max_depth': [2,5,10]}
gs = GridSearchCV(rf(),
   param_grid=pg)
gs.fit(X_train, y_train)
depth = gs.best_params_['max_depth']
```



The power of grid search

Then optimize `n_estimators` :

```
pg = {'n_estimators': [10,20,30]}
gs = GridSearchCV(
   rf(max_depth=depth),
   param_grid=pg)
gs.fit(X_train, y_train)
n_est = gs.best_params_[
    'n_estimators']
```

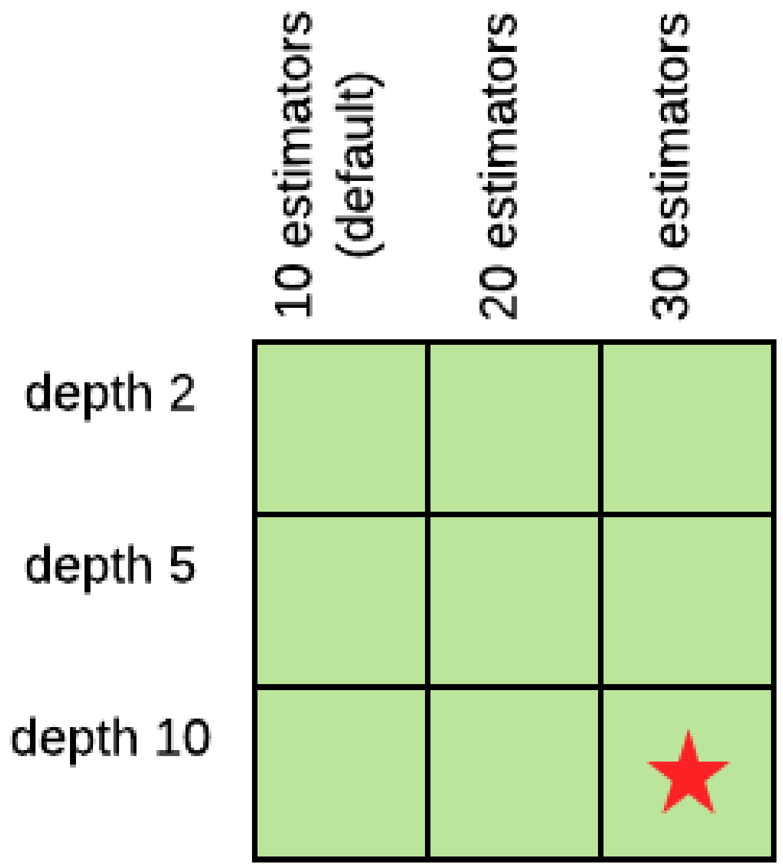


The power of grid search

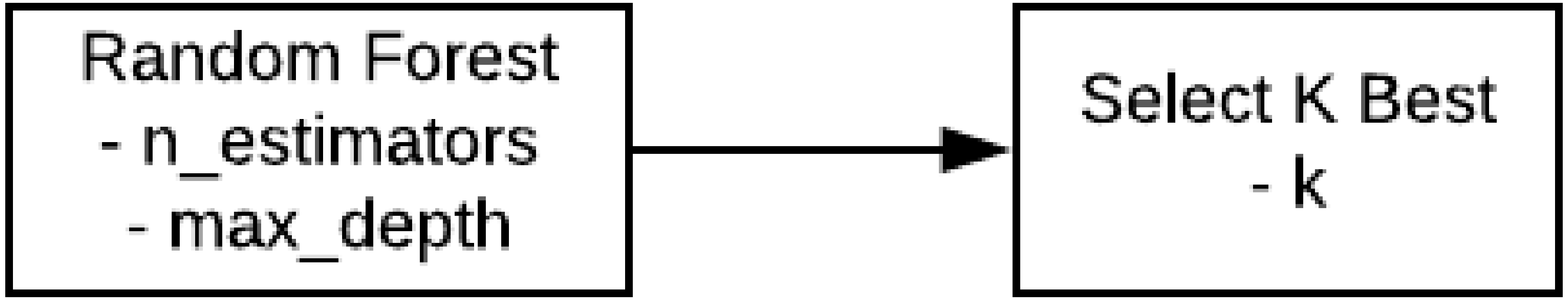
Jointly `max_depth` and `n_estimators` :

```
pg = {
    'max_depth': [2,5,10],
    'n_estimators': [10,20,30]
}
gs = GridSearchCV(rf(),
    param_grid=pg)
gs.fit(X_train, y_train)
print(gs.best_params_)
```

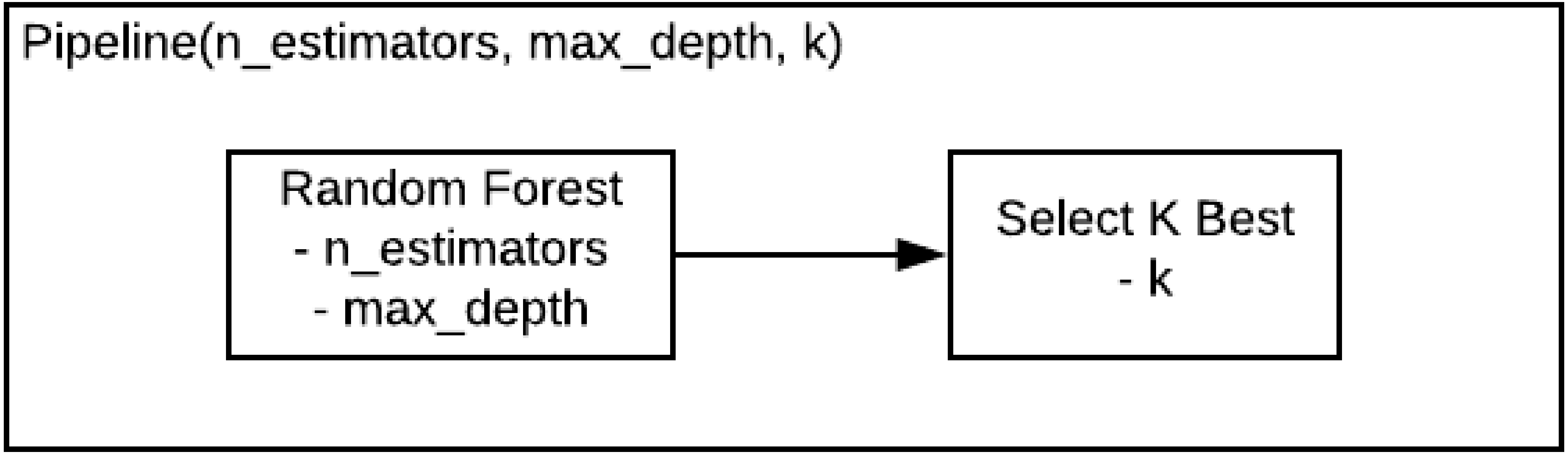
```
{'max_depth': 10, 'n_estimators': 20}
```



Pipelines



Pipelines



Pipeline(n_estimators, max_depth, k)
Random Forest
- n_estimators
- max_depth
Select K Best
- k

Pipelines

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([
    ('feature_selection', SelectKBest(f_classif)),
    ('classifier', RandomForestClassifier())
])
params = dict(
    feature_selection__k=[2, 3, 4],
    classifier__max_depth=[5, 10, 20]
)
grid_search = GridSearchCV(pipe, param_grid=params)
gs = grid_search.fit(X_train, y_train).best_params_
```

```
{'classifier__max_depth': 20, 'feature_selection__k': 4}
```

Customizing your pipeline

```
from sklearn.metrics import roc_auc_score, make_scorer
auc_scorer = make_scorer(roc_auc_score)

grid_search = GridSearchCV(pipe, param_grid=params, scoring=auc_scorer)
```

Don't overdo it

```
params = dict(
    feature_selection__k=[2, 3, 4],
    clf__max_depth=[5, 10, 20],
    clf__n_estimators=[10, 20, 30]
)
grid_search = GridSearchCV(pipe, params, cv=10)
```

3 x 3 x 3 x 10 = 270 classifier fits!

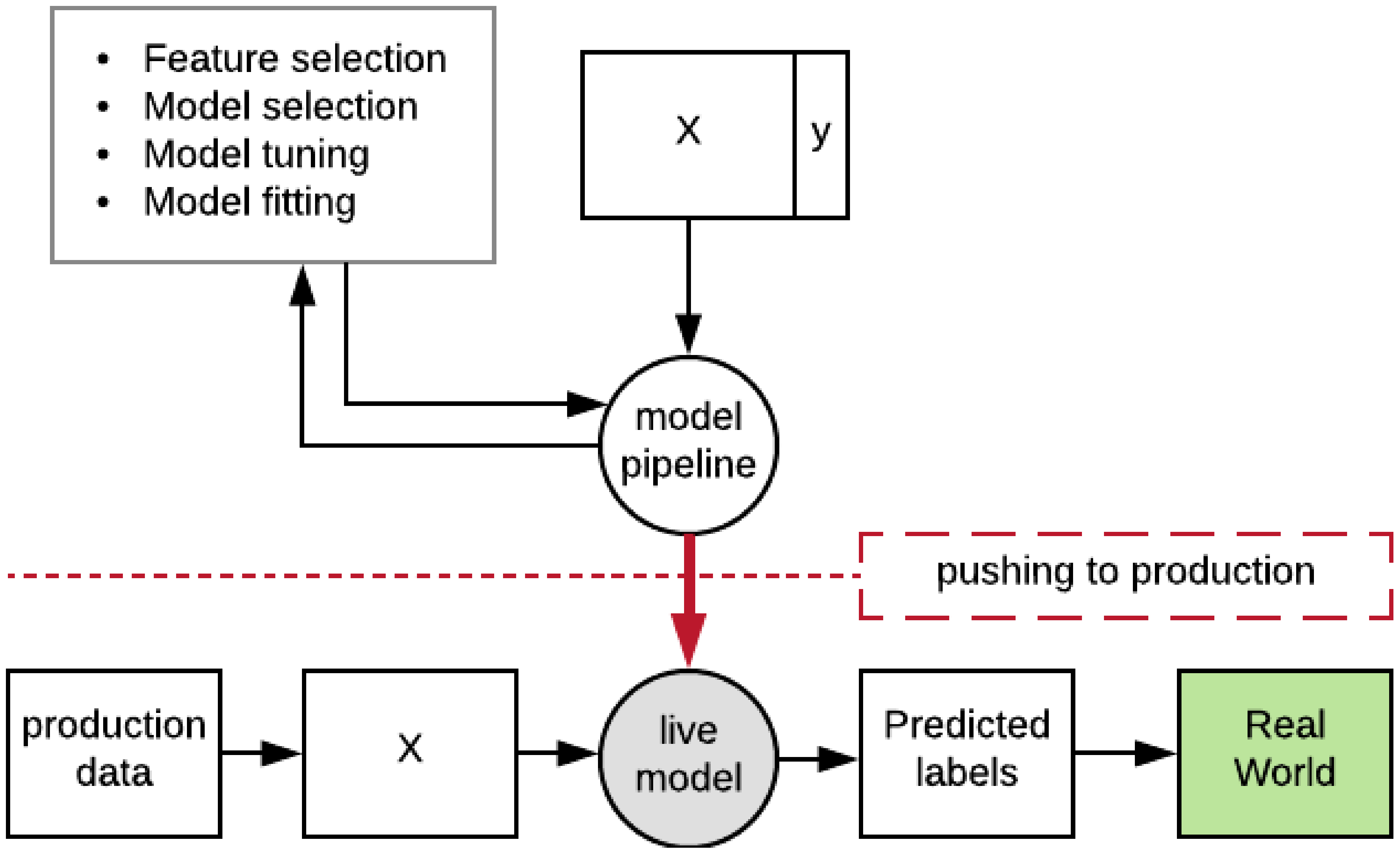
Supercharged workflows

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Model deployment

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor


• Feature selection
• Model selection
• Model tuning
• Model fitting
X
y
model pipeline
pushing to production
production data
X
live model
Predicted labels
Real World

Serializing your model

Store a classifier to file:

```
import pickle
clf = RandomForestClassifier().fit(X_train, y_train)
with open('model.pkl', 'wb') as file:
    pickle.dump(clf, file=file)
```

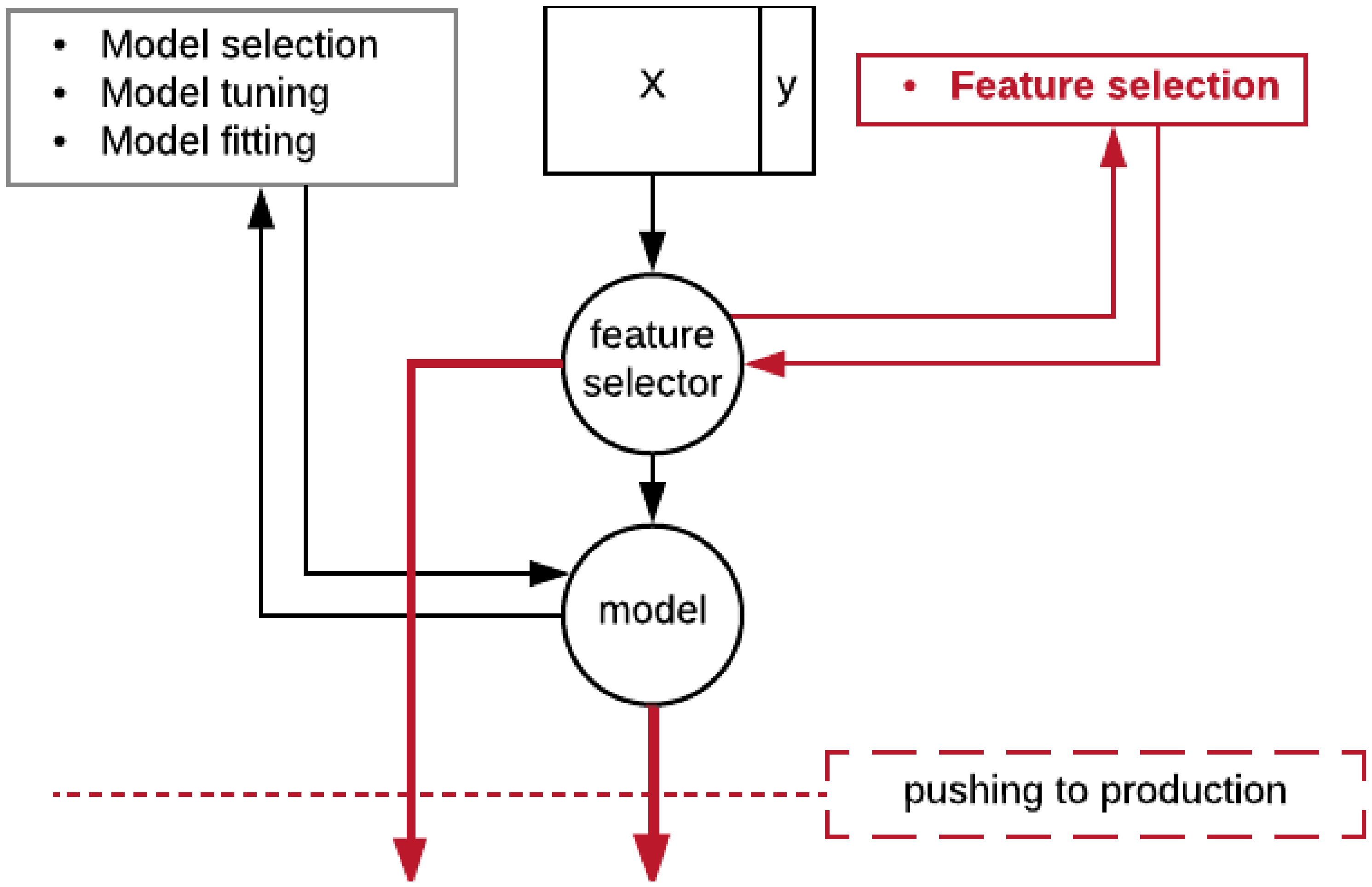
Load it again from file:

```
with open('model.pkl', 'rb') as file:
    clf2 = pickle.load(file)
```

Serializing your pipeline

Development environment:

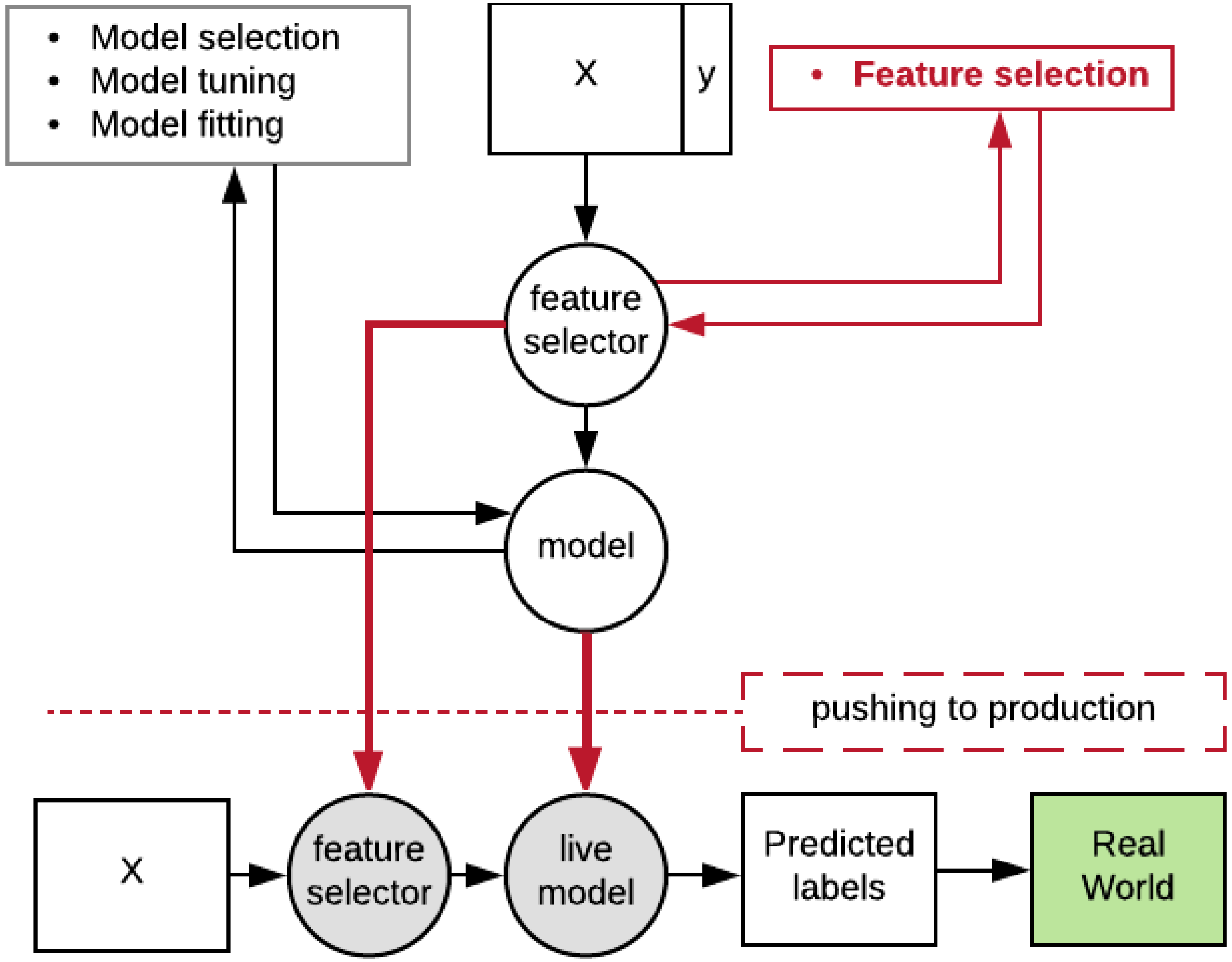
```
vt = SelectKBest(f_classif).fit(
    X_train, y_train)
clf = RandomForestClassifier().fit(
    vt.transform(X_train), y_train)
with open('vt.pkl', 'wb') as file:
     pickle.dump(vt)
with open('clf.pkl', 'wb') as file:
     pickle.dump(clf)
```



Serializing your pipeline

Production environment:

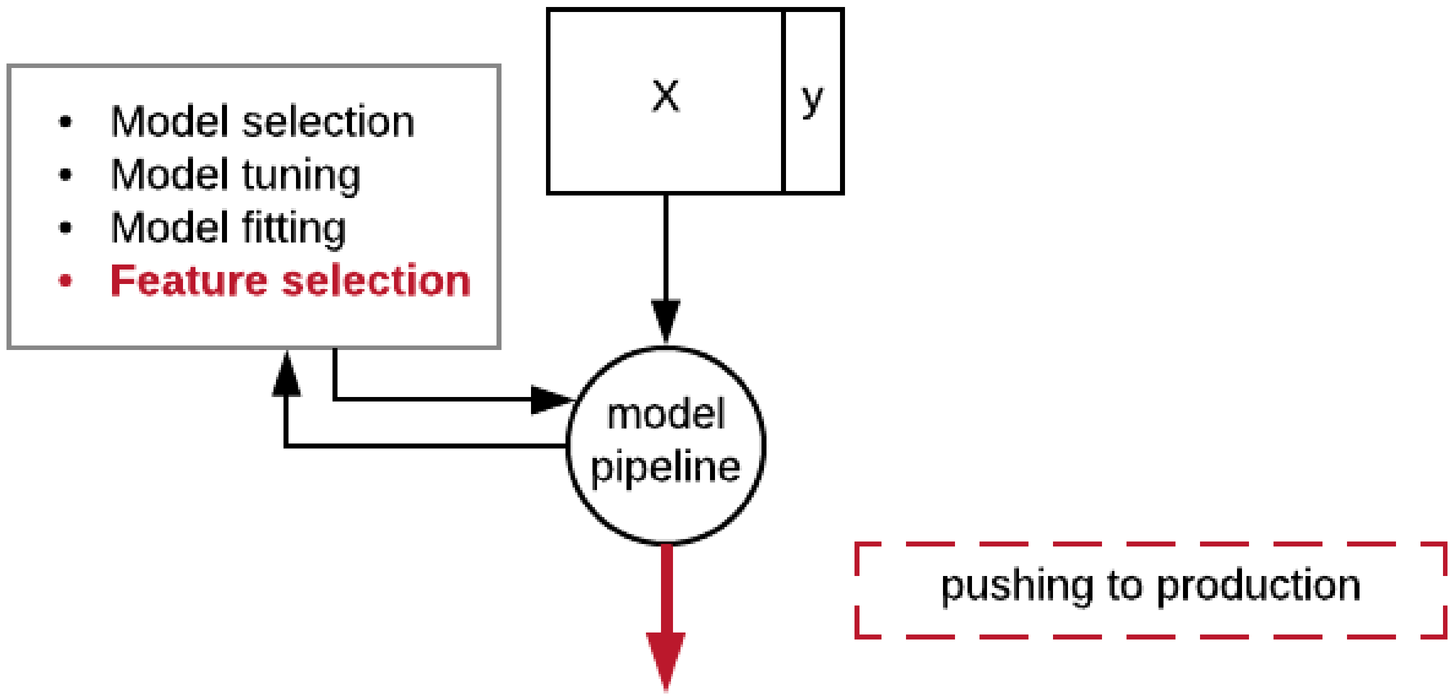
```
with open('vt.pkl', 'rb') as file:
    vt = pickle.load(vt)
with open('clf.pkl', 'rb') as file:
    clf = pickle.load(clf)
clf.predict(vt.transform(X_new))
```



Serializing your pipeline

Development environment:

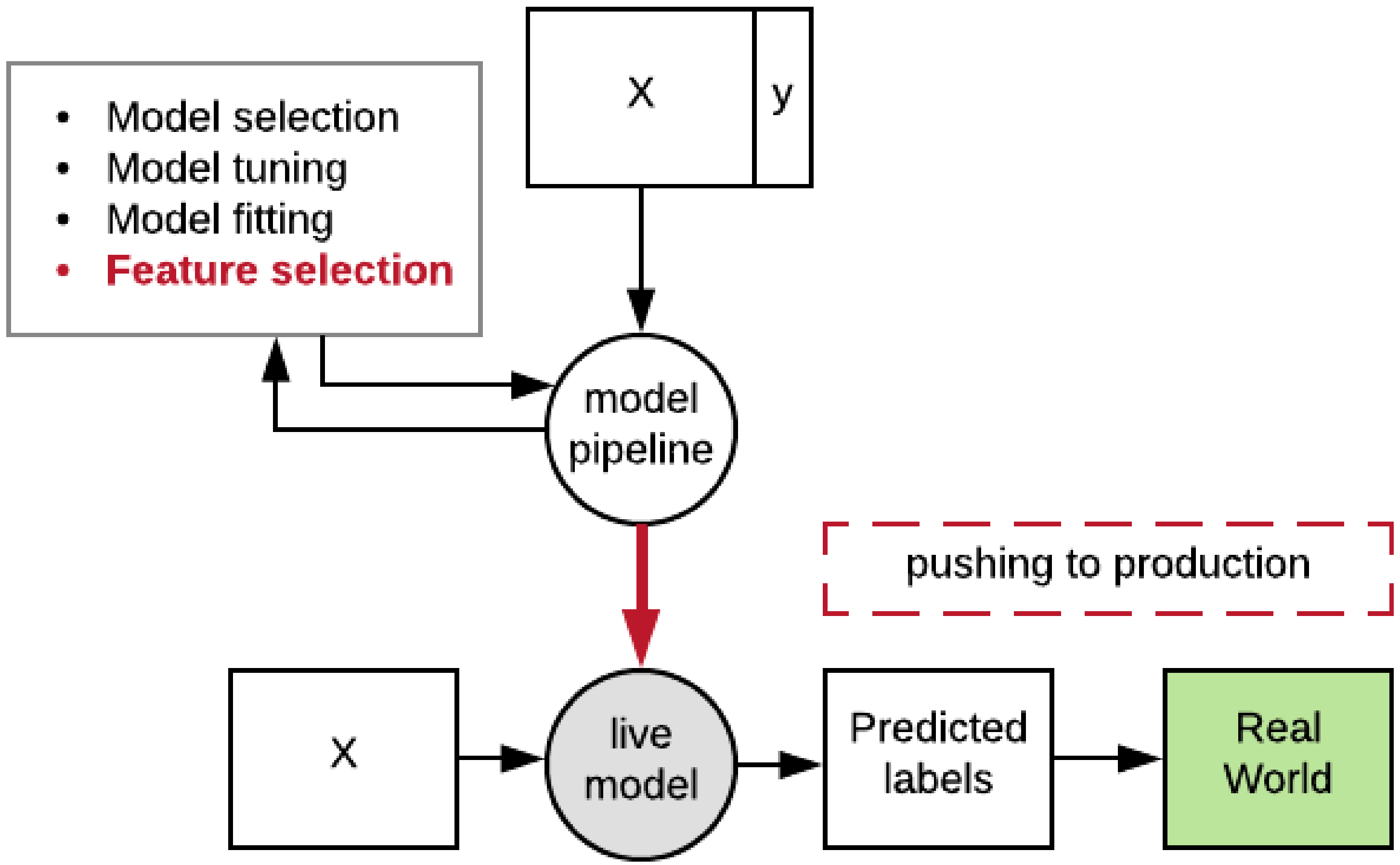
```
pipe = Pipeline([
    ('fs', SelectKBest(f_classif)),
    ('clf', RandomForestClassifier())
])
params = dict(fs__k=[2, 3, 4],
    clf__max_depth=[5, 10, 20])
gs = GridSearchCV(pipe, params)
gs = gs.fit(X_train, y_train)

with open('pipe.pkl', 'wb') as file:
    pickle.dump(gs, file)
```



Serializing your pipeline

Production environment:

```
with open('pipe.pkl', 'rb') as file:
    gs = pickle.dump(gs, file)
gs.predict(X_test)
```



Custom feature transformations

```
   checking_status  duration       ...       own_telephone  foreign_worker
0                1         6       ...                   1               1
1                0        48       ...                   0               1
```

```python
def negate_second_column(X):
    Z = X.copy()
    Z[:,1] = -Z[:,1]
    return Z
```

```python
pipe = Pipeline([('ft', FunctionTransformer(negate_second_column)),
    ('clf', RandomForestClassifier())])
```

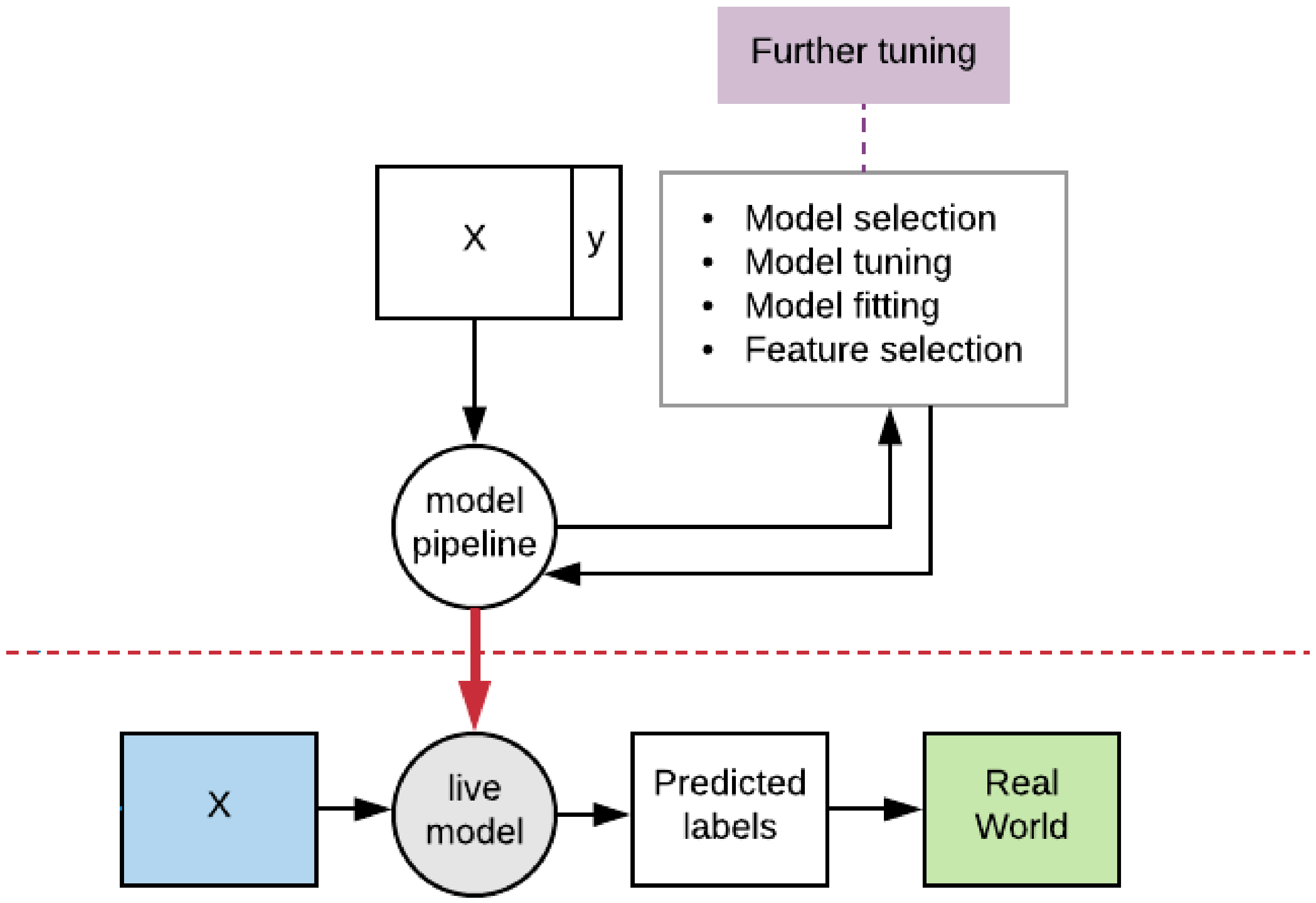
Production ready!

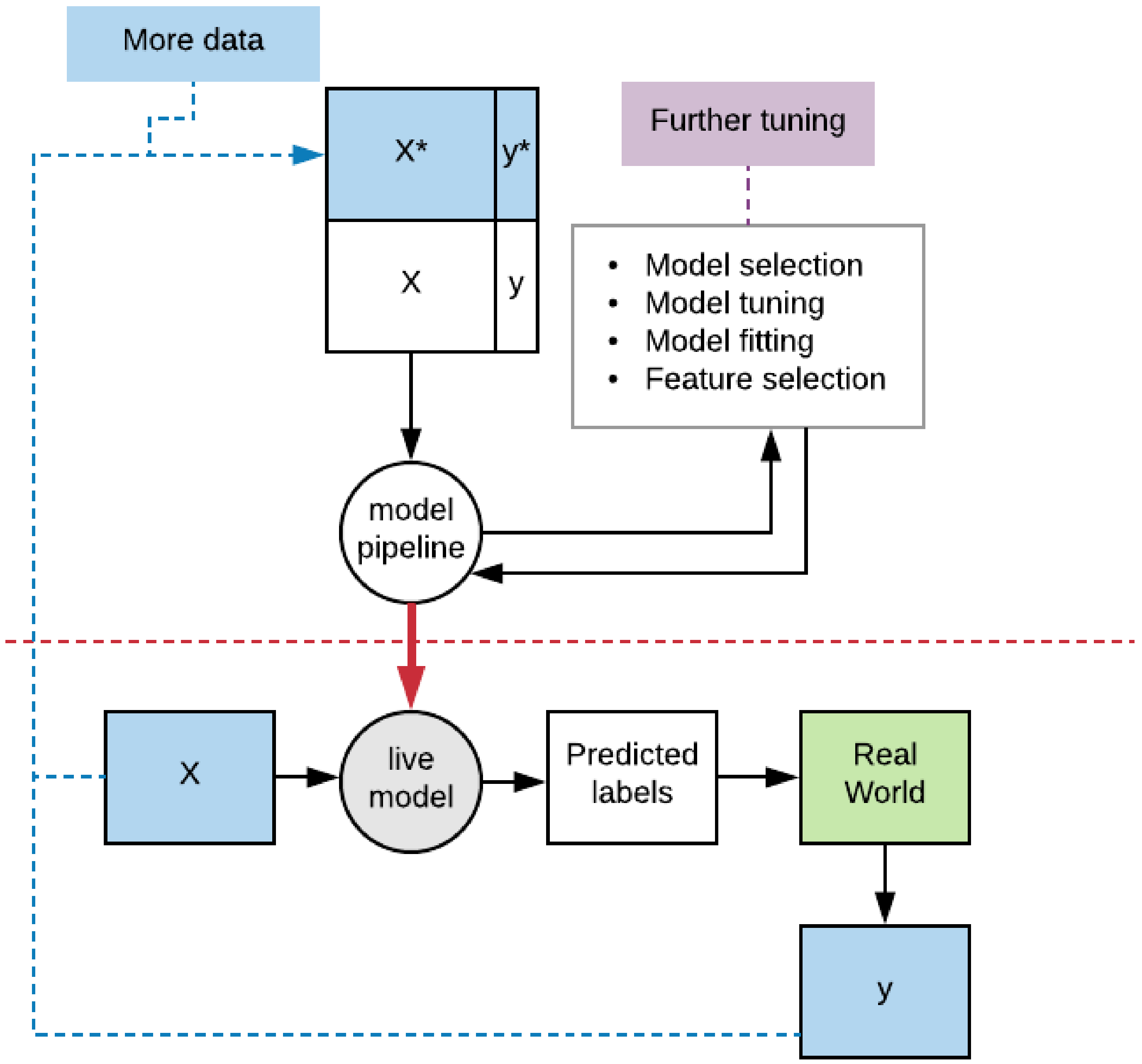
DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

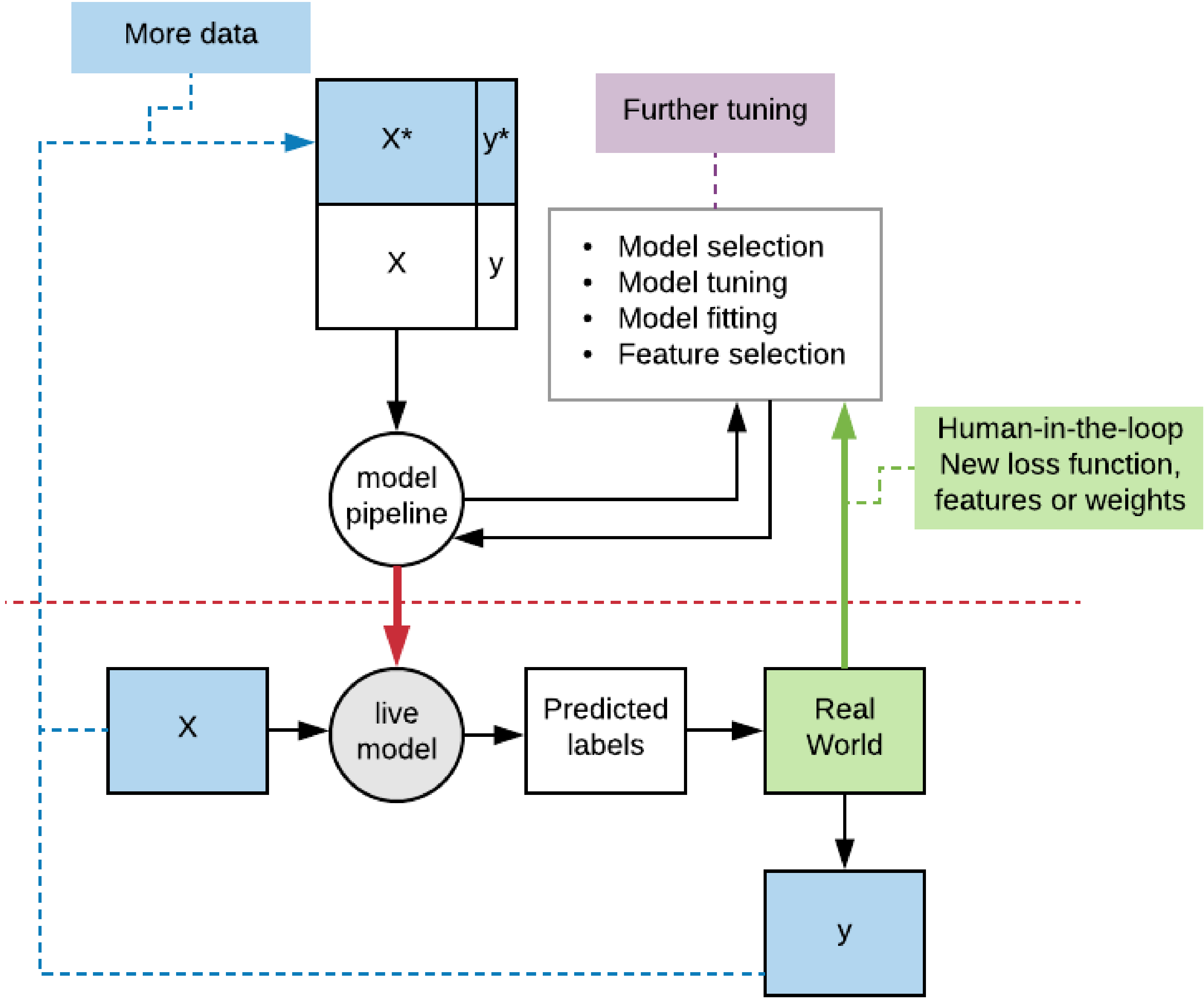
Iterating without overfitting

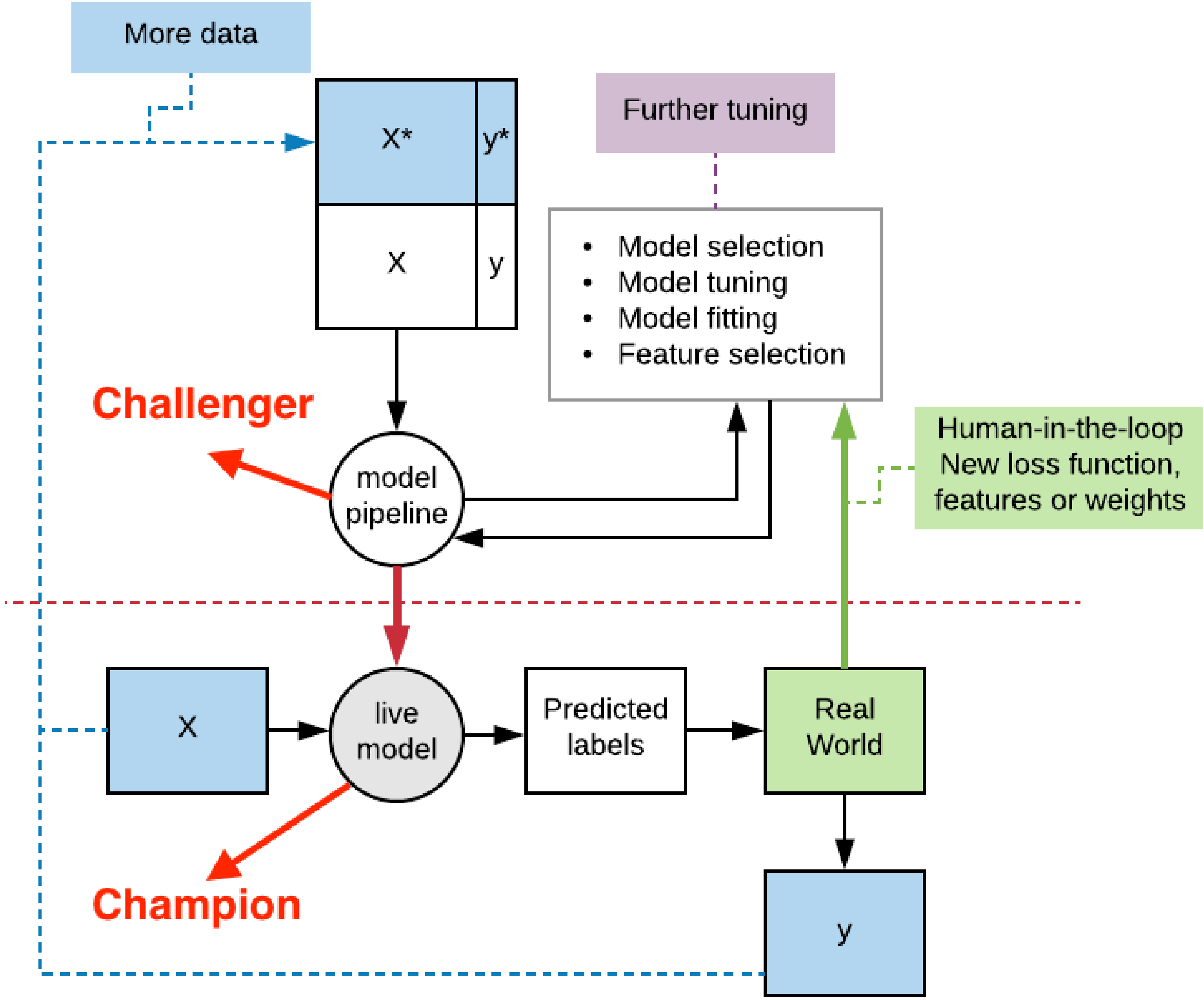
DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor


Further tuning
X
y
• Model selection
• Model tuning
• Model fitting
• Feature selection
model pipeline
X
live model
Predicted labels
Real World


More data
X*
y*
X
y
Further tuning
Model selection
Model tuning
Model fitting
Feature selection
model pipeline
X
live model
Predicted labels
Real World
y


More data
X*
y*
X
y
Further tuning
• Model selection
• Model tuning
• Model fitting
• Feature selection
Human-in-the-loop
New loss function,
features or weights
model
pipeline
X
live
model
Predicted
labels
Real
World
y


More data
X*
y*
X
y
Further tuning
Model selection
Model tuning
Model fitting
Feature selection
Challenger
model pipeline
Human-in-the-loop
New loss function,
features or weights
X
live model
Predicted labels
Real World
Champion
y

Cross-validation results

```
grid_search = GridSearchCV(pipe, params, cv=3, return_train_score=True)
gs = grid_search.fit(X_train, y_train)
results = pd.DataFrame(gs.cv_results_)
```

```
results[['mean_train_score', 'std_train_score',
   'mean_test_score', 'std_test_score']]
```

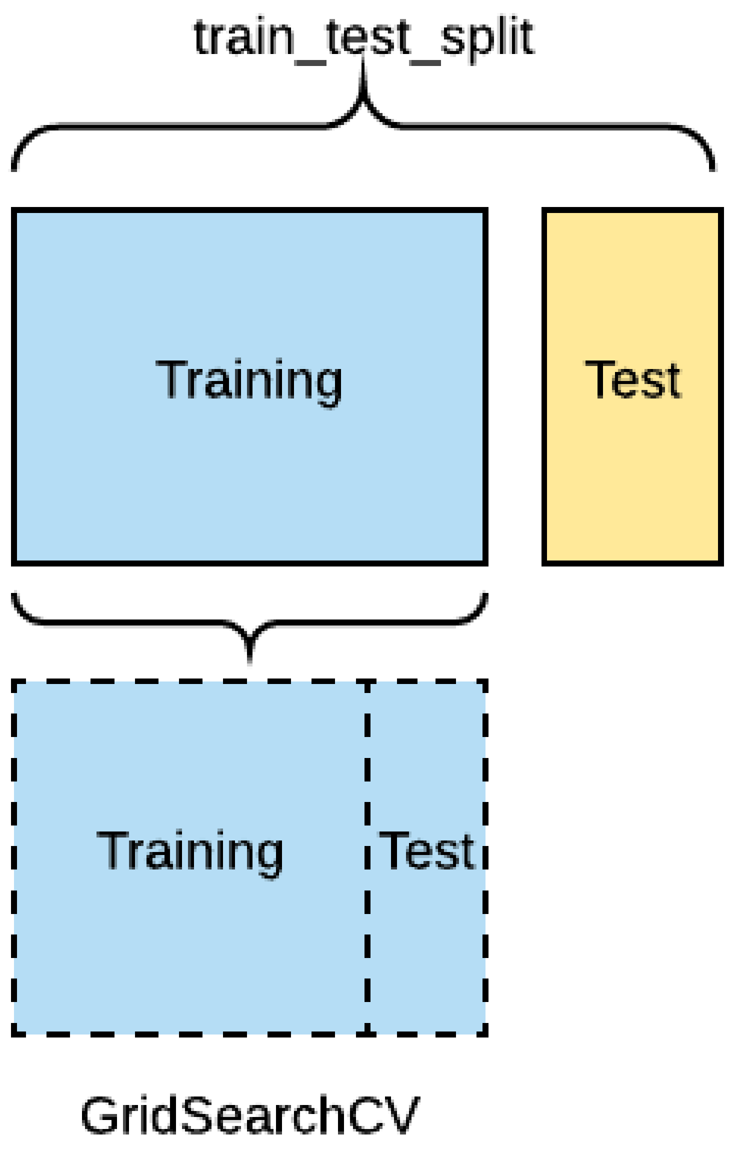
```
   mean_train_score  std_train_score  mean_test_score  std_test_score
0             0.829            0.006            0.735           0.009
1             0.829            0.006            0.725           0.009
2             0.961            0.008            0.716           0.019
3             0.981            0.005            0.749           0.024
...
```

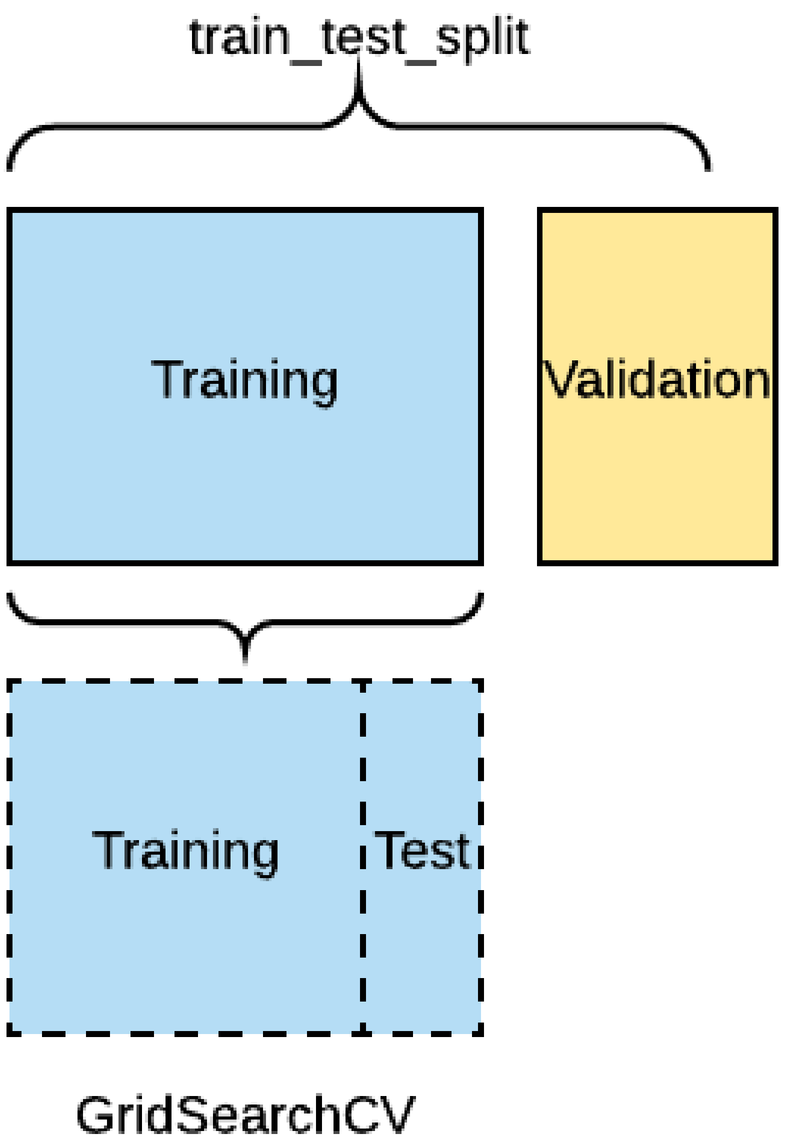
Cross-validation results

	mean_train_score	std_train_score	mean_test_score	std_test_score
0	0.829	0.006	0.735	0.009
1	0.829	0.006	0.725	0.009
2	0.961	0.008	0.716	0.019
3	0.981	0.005	0.749	0.024
4	0.986	0.003	0.728	0.009
5	0.995	0.002	0.751	0.008

Observations:

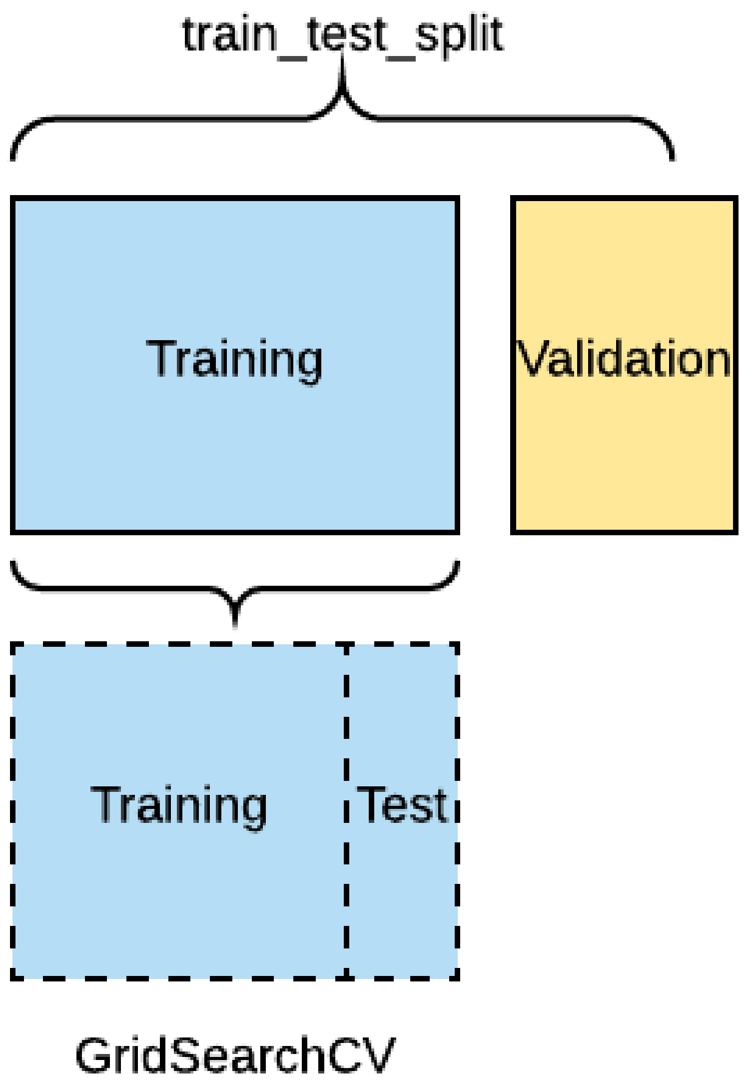
- Training score much higher than test score.
- The standard deviation of the test score is large.

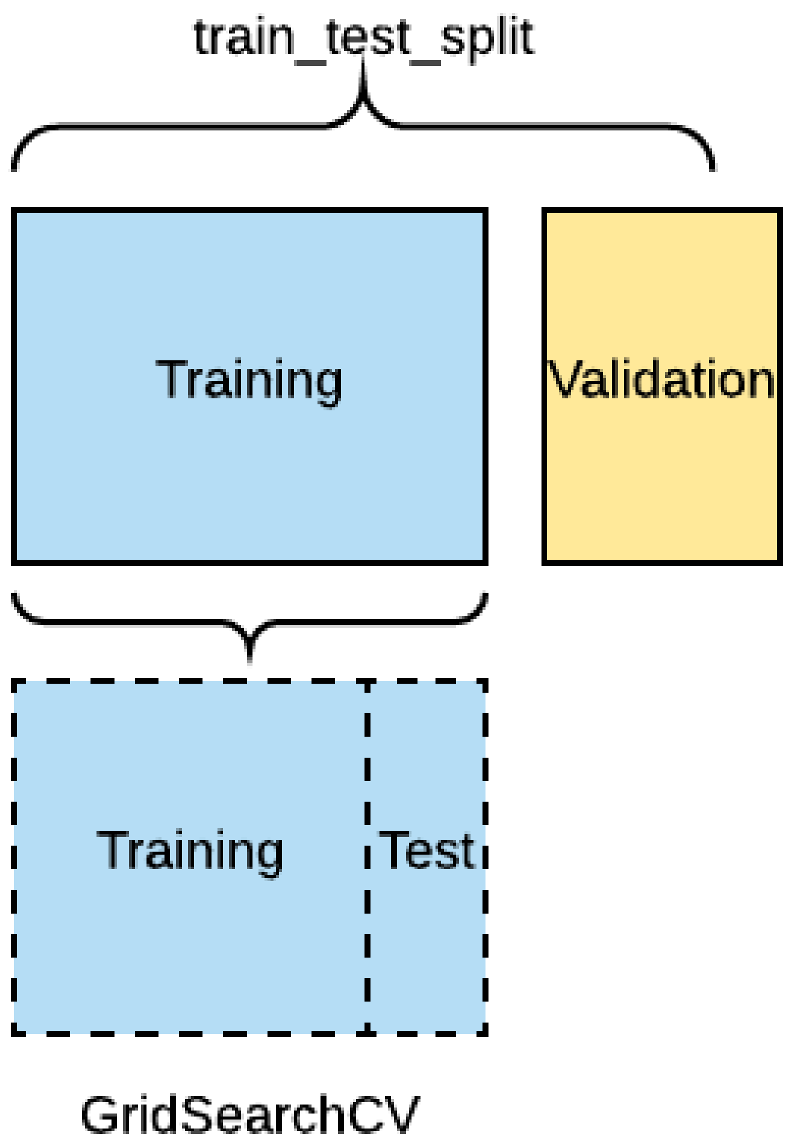

train_test_split
Training
Test
Training
Test
GridSearchCV

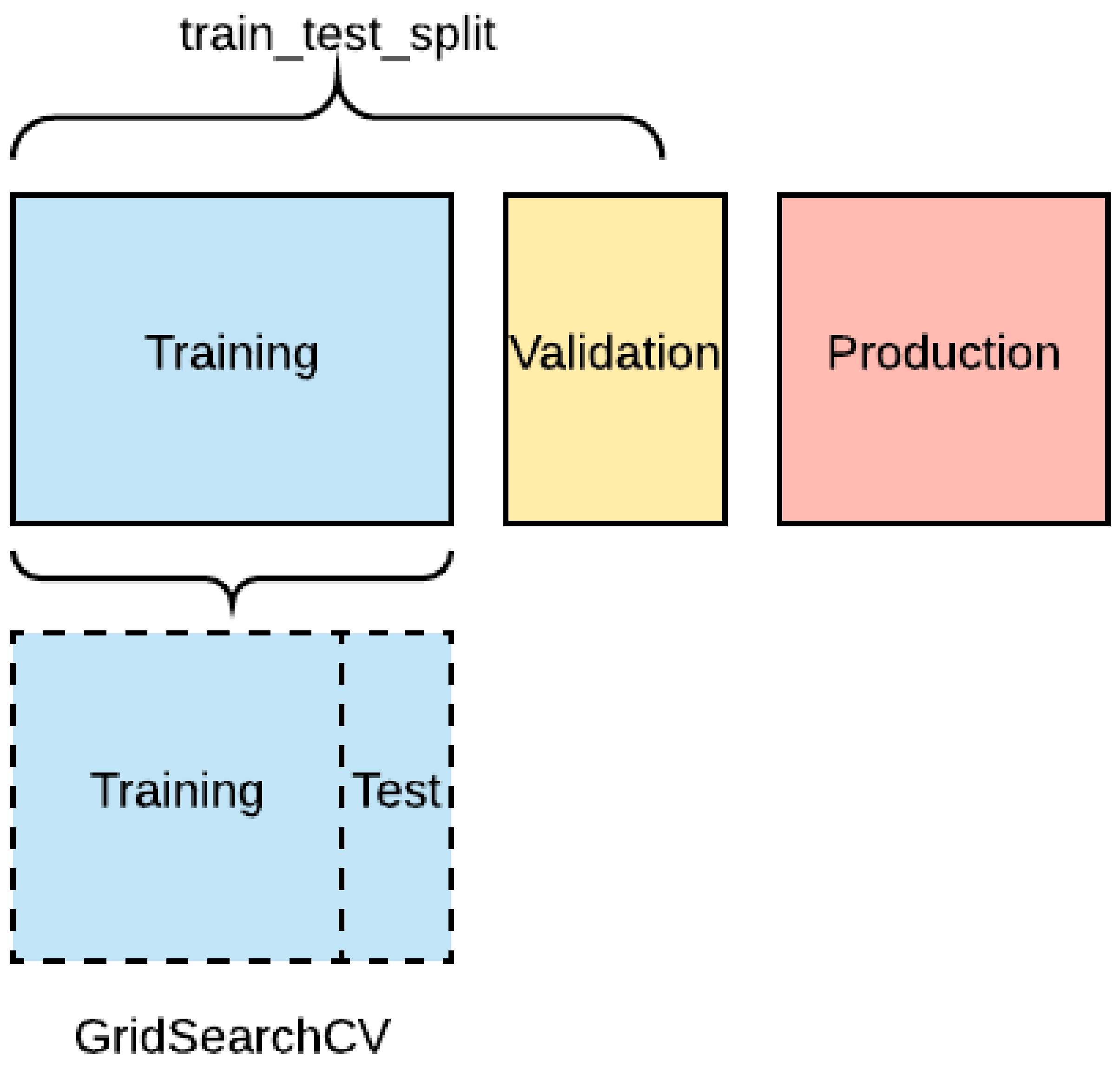

train_test_split
Training
Validation
Training
Test
GridSearchCV

Detecting overfitting

- CV Training Score >> CV Test Score
  - *overfitting in model fitting stage*
  - reduce complexity of classifier
  - get more training data
  - increase cv number
- CV Test Score >> Validation Score
  - *overfitting in model tuning stage*
  - decrease cv number
  - decrease size of parameter grid




train_test_split
Training
Validation
Training
Test
GridSearchCV


train_test_split
Training
Validation
Production
Training
Test
GridSearchCV

"Expert in CV" in your CV!

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dataset shift

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor

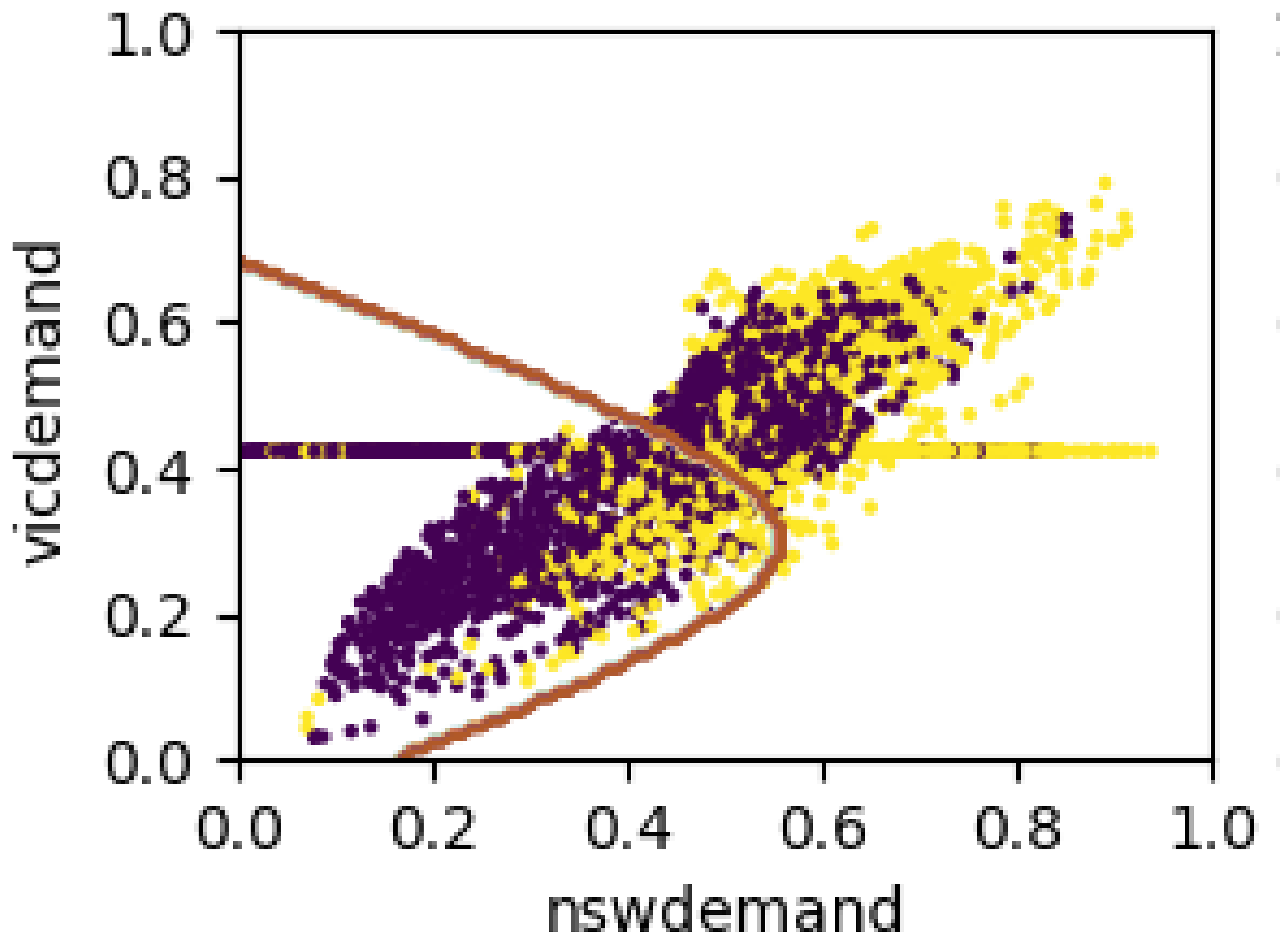
What is dataset shift?

`elec` dataset:

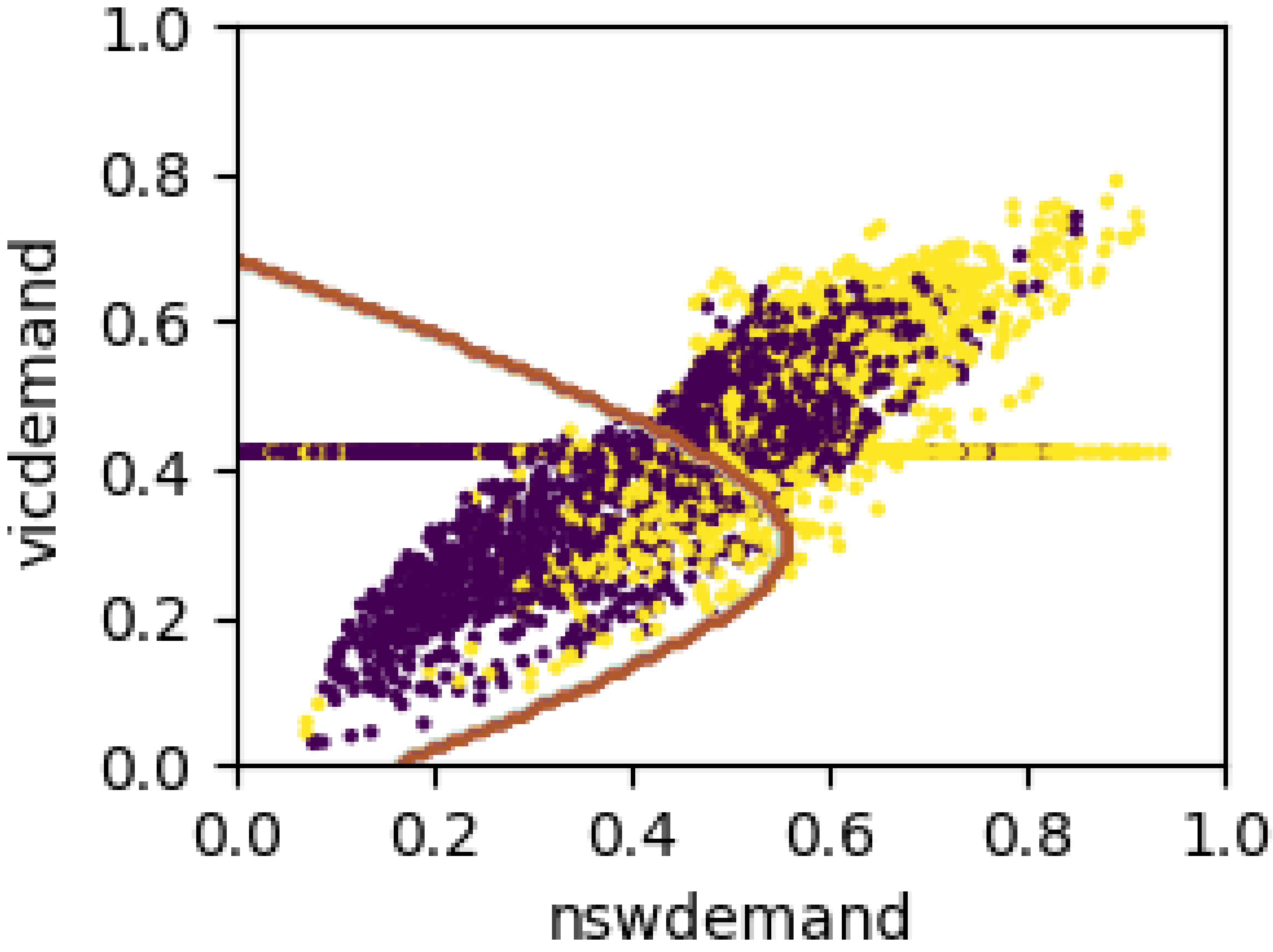
- 2 years worth of data.
- `class=1` represents price went *up* relative to last 24 hours, and `0` means *down*.

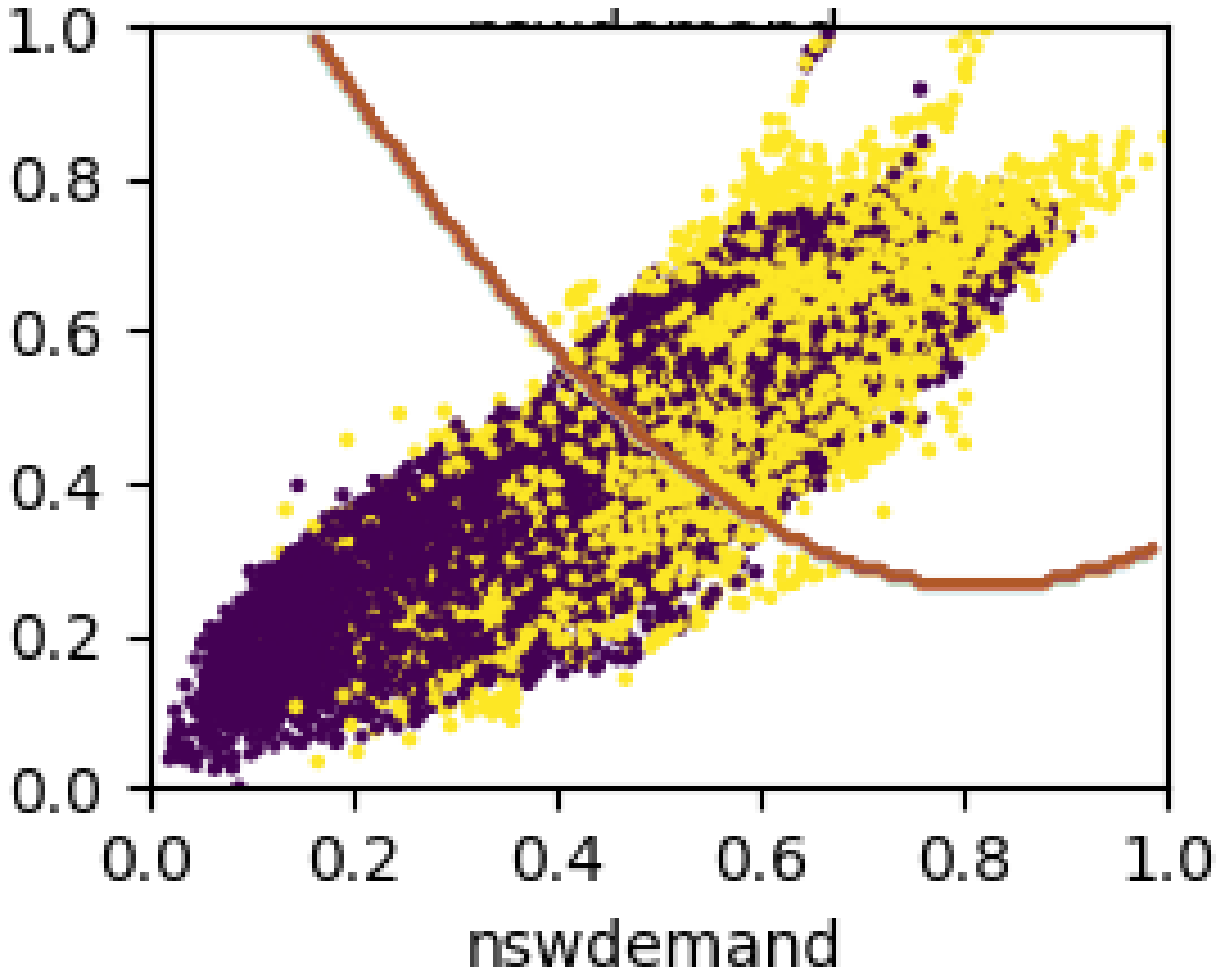
```
   day    period  nswprice  ...    vicdemand  transfer  class
0    2  0.000000  0.056443  ...     0.422915  0.414912      1
1    2  0.553191  0.042482  ...     0.422915  0.414912      0
2    2  0.574468  0.044374  ...     0.422915  0.414912      1

[3 rows x 8 columns]
```

What is shifting exactly?



What is shifting exactly?





Windows

Sliding window

```
window = (t_now-window_size+1):t_now
sliding_window = elec.loc[window]
```

	day	period	nswprice	...	vicdemand	transfer	class
0	2	0.000000	0.056443	...	0.422915	0.414912	1
1	2	0.553191	0.042482	...	0.422915	0.414912	0
2	2	0.574468	0.044374	...	0.422915	0.414912	1
3	2	0.595745	0.044374	...	0.422915	0.414912	1
4	2	0.617021	0.042482	...	0.422915	0.414912	0
5	2	0.638298	0.040861	...	0.422915	0.414912	0
6	2	0.659574	0.041161	...	0.422915	0.414912	0
7	2	0.680851	0.041161	...	0.422915	0.414912	0
8	2	0.702128	0.041161	...	0.422915	0.414912	0
9	2	0.723404	0.042482	...	0.422915	0.414912	0
10	2	0.765957	0.041041	...	0.422915	0.414912	0
11	2	0.787234	0.040711	...	0.422915	0.414912	0
12	2	0.808511	0.040711	...	0.422915	0.414912	0
13	2	0.829787	0.040861	...	0.422915	0.414912	0
14	2	0.851064	0.041041	...	0.422915	0.414912	0
15	2	0.872340	0.042482	...	0.422915	0.414912	0
16	2	0.893617	0.041161	...	0.422915	0.414912	0
17	2	0.914894	0.051489	...	0.422915	0.414912	1
18	2	0.936170	0.056443	...	0.422915	0.414912	1
19	2	0.957447	0.054642	...	0.422915	0.414912	1

Expanding window

```
window = 0:t_now
expanding_window = elec.loc[window]
```

	day	period	nswprice	...	vicdemand	transfer	class
0	2	0.000000	0.056443	...	0.422915	0.414912	1
1	2	0.553191	0.042482	...	0.422915	0.414912	0
2	2	0.574468	0.044374	...	0.422915	0.414912	1
3	2	0.595745	0.044374	...	0.422915	0.414912	1
4	2	0.617021	0.042482	...	0.422915	0.414912	0
5	2	0.638298	0.040861	...	0.422915	0.414912	0
6	2	0.659574	0.041161	...	0.422915	0.414912	0
7	2	0.680851	0.041161	...	0.422915	0.414912	0
8	2	0.702128	0.041161	...	0.422915	0.414912	0
9	2	0.723404	0.042482	...	0.422915	0.414912	0
10	2	0.765957	0.041041	...	0.422915	0.414912	0
11	2	0.787234	0.040711	...	0.422915	0.414912	0
12	2	0.808511	0.040711	...	0.422915	0.414912	0
13	2	0.829787	0.040861	...	0.422915	0.414912	0
14	2	0.851064	0.041041	...	0.422915	0.414912	0
15	2	0.872340	0.042482	...	0.422915	0.414912	0
16	2	0.893617	0.041161	...	0.422915	0.414912	0
17	2	0.914894	0.051489	...	0.422915	0.414912	1
18	2	0.936170	0.056443	...	0.422915	0.414912	1
19	2	0.957447	0.054642	...	0.422915	0.414912	1

Dataset shift detection

```
# t_now = 40000, window_size = 20000
clf_full = RandomForestClassifier().fit(X, y)
clf_sliding = RandomForestClassifier().fit(sliding_X, sliding_y)
```

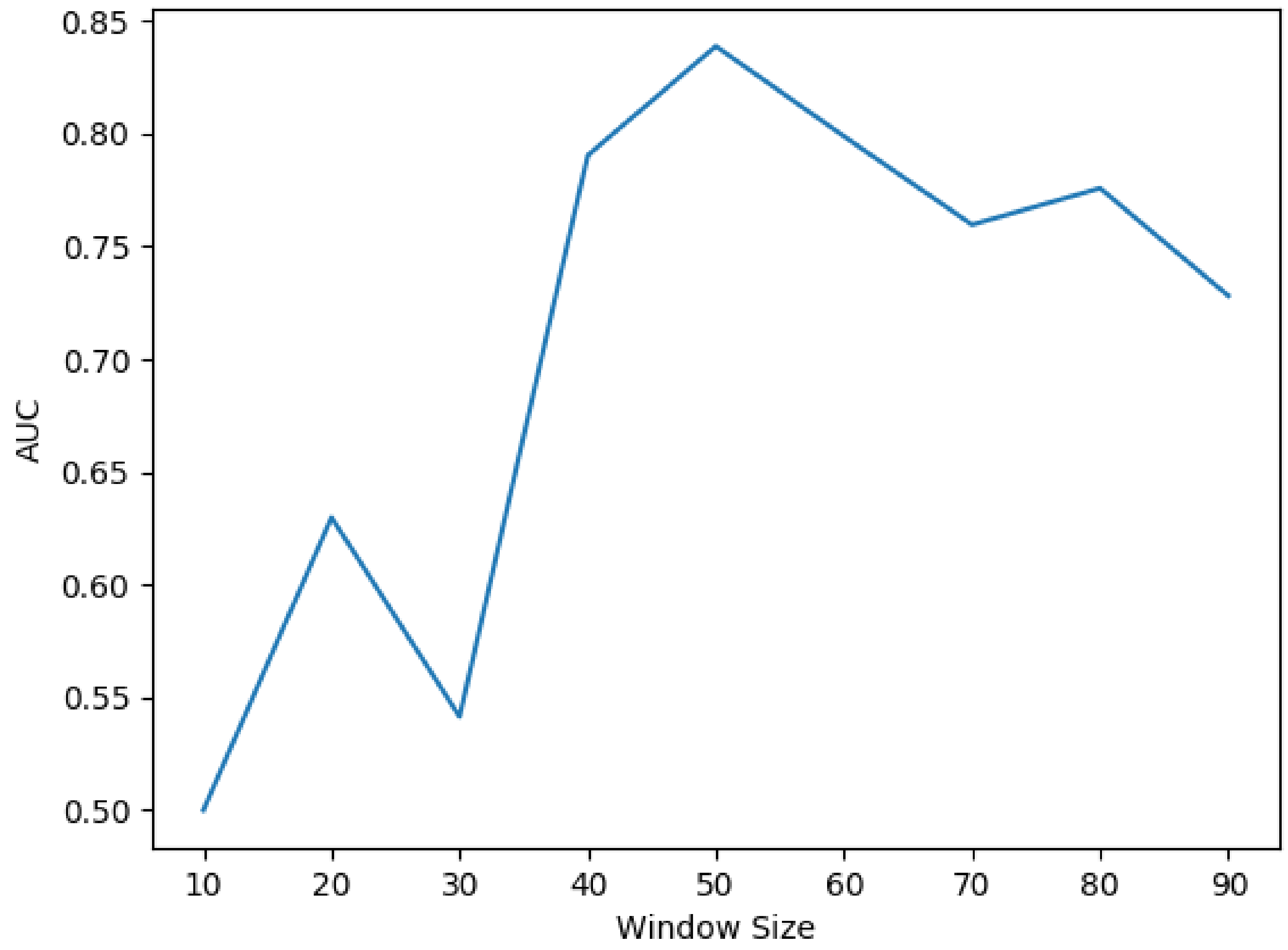
```
# Use future data as test
test = elec.loc[t_now:elec.shape[0]]
test_X = test.drop('class', 1); test_y = test['class']
```

```
roc_auc_score(test_y, clf_full.predict(test_X))
roc_auc_score(test_y, clf_sliding.predict(test_X))
```

```
0.775
0.780
```

Window size

```python
for w_size in range(10, 100, 10):
    sliding = arrh.loc[
      (t_now - w_size + 1):t_now
    ]
    X = sliding.drop('class', 1)
    y = sliding['class']
    clf = GaussianNB()
    clf.fit(X, y)
    preds = clf.predict(test_X)
    roc_auc_score(test_y, preds)
```



Domain shift

`arrhythmia` dataset:

```
   age  sex  height  ...    chV6_TwaveAmp  chV6_QRSA  chV6_QRSTA  class
0   75    0     190  ...              2.9       23.3        49.4      0
1   56    1     165  ...              2.1       20.4        38.8      0
2   54    0     172  ...              3.4       12.3        49.0      0
3   55    0     175  ...              2.6       34.6        61.6      1
4   75    0     190  ...              3.9       25.4        62.8      0

[5 rows x 280 columns]
```

More data is not always better!

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON