

# **COMP47750/COMP47990**

# **Machine Learning with Python**

## **Model Selection**

**Part I**  
**Pipelines**

**Pádraig Cunningham**

**School of Computer Science**



# What is Model Selection?

---

- Easy:
  - What algorithm is best, e.g. k-NN, Decision Trees, Naive Bayes, etc
- Not so easy:
  - What preprocessing steps?
    - Data scaling
    - Missing value imputation
    - Preprocessing text data
  - Setting model hyperparameters

## What is Model Selection?

### Pipeline

- A set of ‘canned’ steps can be grouped together into a pipeline
  - e.g. StandardScalar + Classifier

### Grid Search

- [Hyper]parameter tuning
- Grid is the space of all parameter combinations

- e.g. 5 x 2 grid:
  - $k = [1, 3, 5, 7, 10]$ ,
  - $distance = [\text{weighted}, \text{unweighted}]$

Test data should not be used in parameter tuning  
So pipelines and grid search used together

# BTW: What is a Hyperparameter?

---

## ■ Model parameters

- Estimated by the learning algorithm, e.g.
  - Coefficients in linear models
  - Weights in neural net
  - Conditional probabilities in Naive Bayes

## ■ Hyperparameters

- ***Set by hand***, e.g.
  - *k* in *k*-Nearest Neighbour
  - *max\_depth* in a Decision Tree
  - *[split] criterion*: ('gini' or 'entropy') in a Decision Tree.

In practice: hyper-parameter tuning  
might be automated

Does that not make them regular parameters?



# First: Missing Value Imputation

## ■ Why?

- A preprocessing step where access to test data can have an impact

Replace with mean for column

```
imp = SimpleImputer(missing_values=np.nan,
                     strategy='mean')
imp.fit(X)
xi = imp.transform(X)
```

Impute from similar examples

```
imp_kNN = KNNImputer(missing_values = np.nan)
imp_kNN.fit(X)
xi = imp_kNN.transform(X)
```

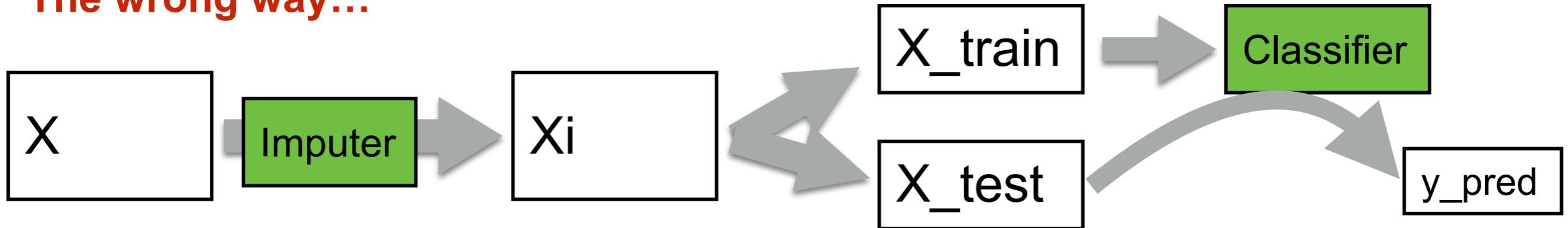
Imputer should not have access to test data

**UCI Mammographic Mass Data**

Age	Shape	Margin	Density	Severity
67.0	3.0	5.0	3.0	1
43.0	1.0	1.0	NaN	1
58.0	4.0	5.0	3.0	1
28.0	1.0	1.0	3.0	0
74.0	1.0	5.0	NaN	1
65.0	1.0	NaN	3.0	0
70.0	NaN	NaN	3.0	0
42.0	1.0	NaN	3.0	0
57.0	1.0	5.0	3.0	1
60.0	NaN	5.0	1.0	1

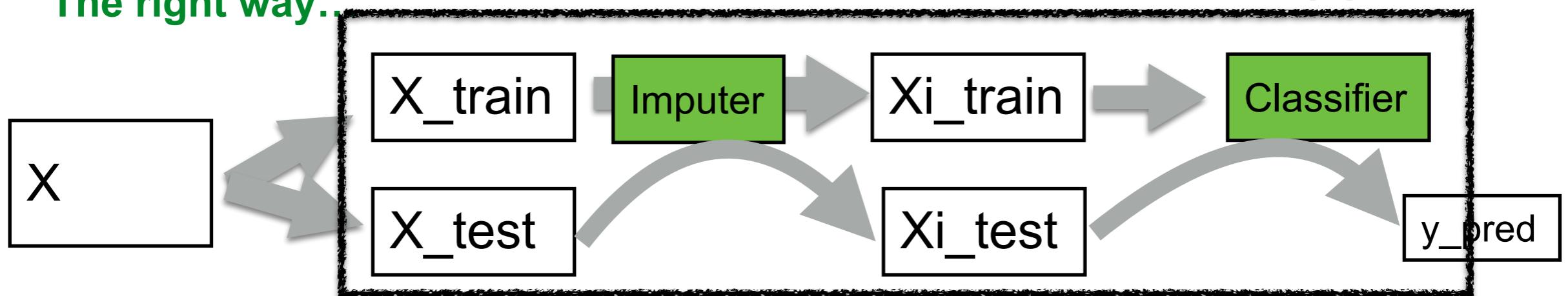
# Pipelines

The wrong way...



The right way...

This can be a pipeline



**X\_test not used to 'fit' the Imputer**

```

imp_kNN = KNNImputer(missing_values = np.nan)
imp_kNN.fit(x_train)
xi_train = imp_kNN.transform(x_train)
xi_test = imp_kNN.transform(x_test)
  
```

# Pipeline: Hold-out testing

## ■ Pipeline:

- Two transforms:
  - **KNNImputer**
  - **StandardScaler**
- One Estimator
  - **KNeighborsClassifier**

```
kNNpipe = Pipeline(steps=[  
    ('imputer', KNNImputer(missing_values = np.nan)),  
    ('scaler', StandardScaler()),  
    ('classifier', KNeighborsClassifier())])  
In [150]:  
kNNpipe.fit(X_train, y_train)  
y_pred = kNNpipe.predict(X_test)  
print("Accuracy: {:.4f}".format(accuracy_score(y_test,y_pred)))  
confusion_matrix(y_test, y_pred)
```

# Pipeline: Cross-Validation

- Pipeline object passed to `cross_val_score`
- All fitting and transforming done automatically
  - New imputer and scaler for each fold

```
kNNpipe = Pipeline(steps=[  
    ('imputer', KNNImputer(missing_values = np.nan)),  
    ('scaler', StandardScaler()),  
    ('classifier', KNeighborsClassifier())])  
  
acc_arr = cross_val_score(kNNpipe, X, y, cv=5)  
print("Accuracy: {:.4f}".format(sum(acc_arr)/len(acc_arr)))
```

- Hold-out accuracy: 0.82
- X-val accuracy: 0.78

*Why the difference, which is more reliable?*

# Data Leak

- When model selection / training has access to the test data
- Data leakage is particularly an issue with feature selection

The wrong way

```
rkf = RepeatedKFold(n_splits=10, n_repeats=10, random_state=42)

FS_trans = SelectKBest(mutual_info_classif, k=7).fit(X, y)
X_FS = FS_trans.transform(X)

knn = KNeighborsClassifier()
acc_arr = cross_val_score(knn, X_FS, y, cv=rkf, n_jobs = -1)
```

Feature selection  
uses all the data

Acc: 0.66

The right way

```
FSpipe = Pipeline(steps=[
    ('fs', SelectKBest(mutual_info_classif, k=7)),
    ('classifier', KNeighborsClassifier())])

acc_arr = cross_val_score(FSpipe, X, y, cv=rkf, n_jobs = -1)
```

Feature selection  
uses all the data

Acc: 0.63

# **COMP47750/COMP47990**

# **Machine Learning with Python**

## **Model Selection**

**Part II**  
**Grid Search**

**Pádraig Cunningham**

**School of Computer Science**



# Grid Search

- The *grid* is the space of all hyperparameter combinations
- KNeighborsClassifier
  - n\_neighbors: {1,3,5,10}
  - weights: {'uniform', 'distance'}
  - metric: {'euclidean', 'manhattan'}

4 x 2 x 2 = 16 combinations

```
knn = KNeighborsClassifier()

param_grid = {'n_neighbors': [1,3,5,10],
              'metric': ['manhattan', 'euclidean'],
              'weights': ['uniform', 'distance']}

knn_gs = GridSearchCV(knn, param_grid, cv=10,
                      verbose = 1, n_jobs = -1)
```

# Running Grid Search

- Parameter sets are ‘scored’ based on the default score for the classifier.
  - For `KNeighborsClassifier()` this is accuracy

```
knn = KNeighborsClassifier()

param_grid = {'n_neighbors': [1,3,5,10],
              'metric': ['manhattan', 'euclidean'],
              'weights': ['uniform', 'distance']}

In [16]:
knn_gs = GridSearchCV(knn, param_grid, cv=10,
                      verbose = 1, n_jobs = -1)
knn_gs = knn_gs.fit(X_trainS,y_train)
```

Fitting 10 folds for each of 16 candidates, totalling 160 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed:    2.1s finished
```

# Grid Search: using the results - 3 options

- The GridSearchCV object IS a classifier

```
y_pred_gs = knn_gs.predict(x_tests)
```

- Explicitly build a classifier with the best parameters

- **best\_params\_** dictionary

```
knn_gs.best_params_
Out[25]:
{'metric': 'manhattan', 'n_neighbors': 1, 'weights': 'uniform'}
In [19]:
knn2 = KNeighborsClassifier(metric= 'manhattan',
                            n_neighbors = 1, weights = 'uniform')
```

- Unpack the best parameters directly

```
knn3 = KNeighborsClassifier(**knn_gs.best_params_)
```



# Pipelines & Grid Search

## ■ Syntax is a bit cumbersome

```

kNNpipe = Pipeline(steps=[
    ('imputer', KNNImputer(missing_values = np.nan)),
    ('scaler', StandardScaler()),
    ('classifier', KNeighborsClassifier())])

param_grid = {
    'scaler':[StandardScaler(), MinMaxScaler(), 'passthrough'],
    'classifier_n_neighbors':[1,3,5,10],
    'classifier_metric':[ 'manhattan', 'euclidean'],
    'classifier_weights':[ 'uniform', 'distance']}
  
```

3 scaler options

classifier options

```

pipe_gs = GridSearchCV(kNNpipe, param_grid, cv=10,
                       verbose = 1, n_jobs = -1)
  
```

### □ How many candidates?

Fitting 10 folds for each of 48 candidates, totalling 480 fits

pipe\_gs.best\_params\_

```

{'classifier_metric': 'manhattan',
 'classifier_n_neighbors': 10,
 'classifier_weights': 'uniform',
 'scaler': StandardScaler()}
  
```

# Grid Search

## ■ Access all results:

```
scores_df = pd.DataFrame(pipe_gs.cv_results_)
scores_df =
    scores_df.sort_values(by=['rank_test_score']).reset_index(drop='index')
scores_df[['rank_test_score', 'mean_test_score', 'param_scaler',
           'param_classifier__n_neighbors']].head()
```

	rank_test_score	mean_test_score	param_scaler	param_classifier__n_neighb
0	1	0.79	StandardScaler()	10
1	2	0.78	MinMaxScaler()	10
2	3	0.78	MinMaxScaler()	10
3	4	0.78	passthrough	10
4	5	0.77	StandardScaler()	10

# RandomizedSearchCV

---

- A randomised rather than an exhaustive search
- Suitable when the parameter space is huge
- A parameter search budget can be set
  - Specify the number of states to be checked
- Insensitive to parameters that don't matter

# Summary

---

- What is Model Selection?
- Model Selection support in scikit-learn
  - Pipelines
  - Grid Search
- Work through the two notebooks
- Tackle the Tutorial exercises