Machine Learning

Model Selection

Sarah Jane Delany

What is Model Selection?

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

Overview

Code Notebooks:
07 Pipelines, 07 GridSearch

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* = [weighted, 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
    - Support vectors in SVM
- 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.
    - α learning rate in Gradient Descent

In practice: hyper-parameter tuning might be automated

Does that not make them regular parameters? 🤷

Preprocessing example: 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)
```

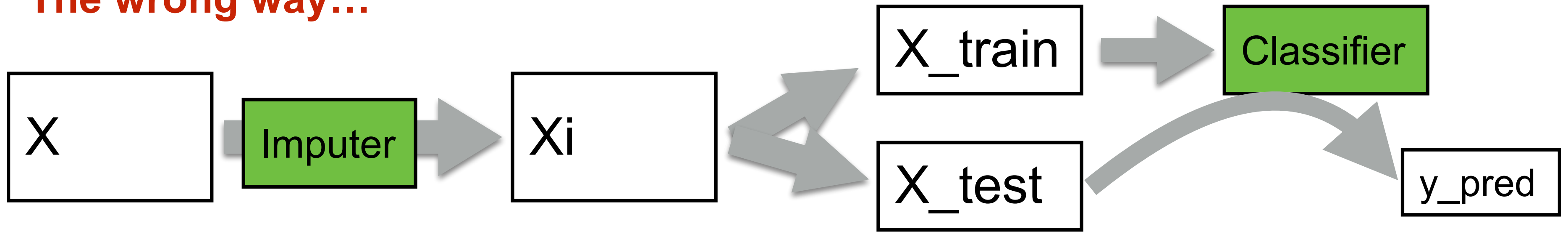
Imputer should not have access to test data

UCI Mammographic Mass Data

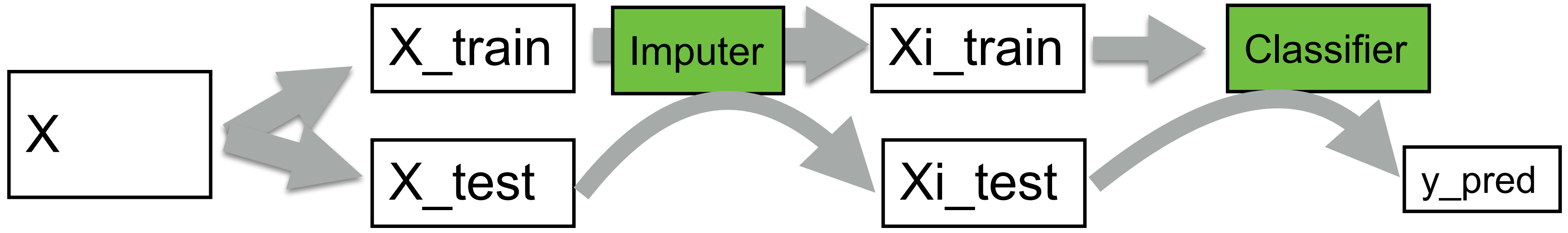
Age	Shape	Margin	Density	Severity
67.0	3.0	5.0	3.0	1
43.0	1.0	1.0	NaN	1
58.0	4.0	5.0	3.0	1
28.0	1.0	1.0	3.0	0
74.0	1.0	5.0	NaN	1
65.0	1.0	NaN	3.0	0
70.0	NaN	NaN	3.0	0
42.0	1.0	NaN	3.0	0
57.0	1.0	5.0	3.0	1
60.0	NaN	5.0	1.0	1

Preprocessing & Data Splitting…

The wrong way…



The right way…



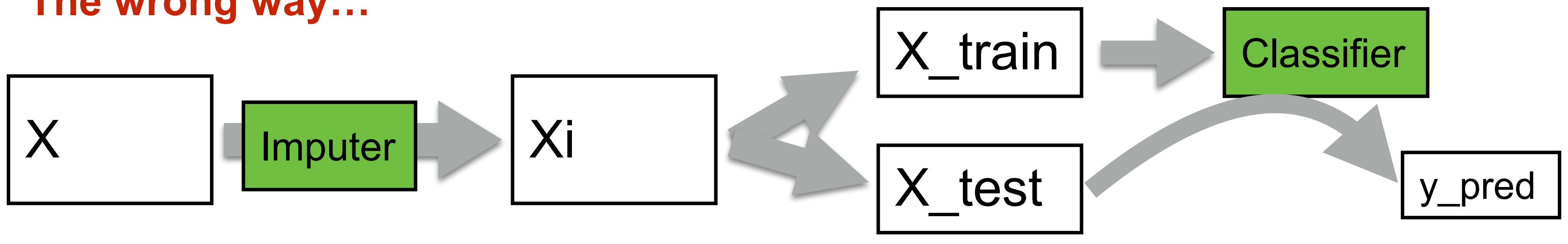
X_test not used to 'fit' the Imputer

```
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)
```

See example in 07 Pipelines

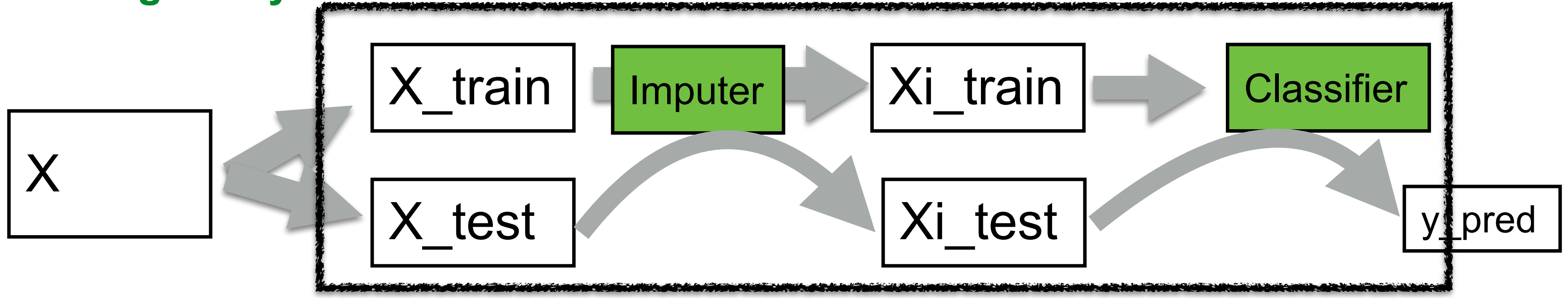
Preprocessing & Data Splitting…

The wrong way…



The right way…

This can be a pipeline



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: {0:4.2f}".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: {0:4.2f}".format(sum(acc_arr)/len(acc_arr)))
```

➡ Hold-out accuracy: 0.82

➡ X-val accuracy: 0.78

➡ *Why the difference, which is more reliable?*

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

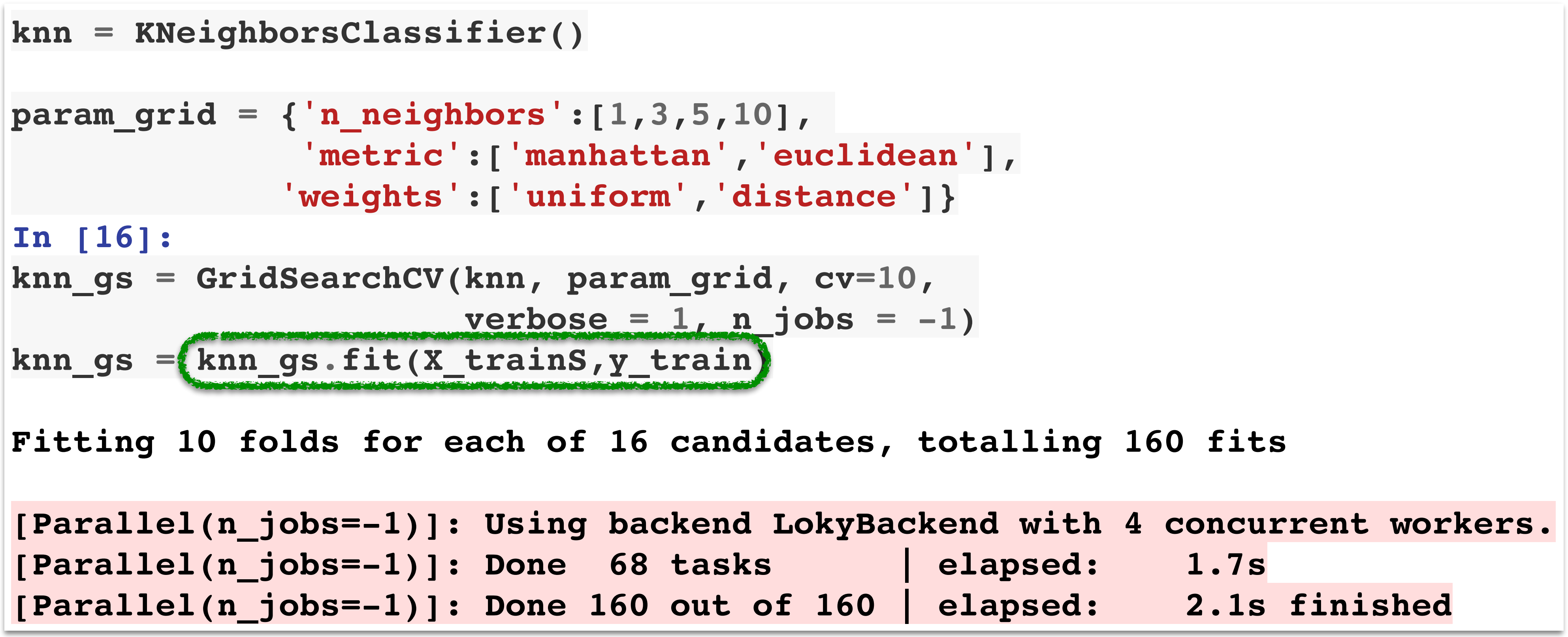
```
knn = KNeighborsClassifier()

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_)
```

🤯

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 Lab exercise