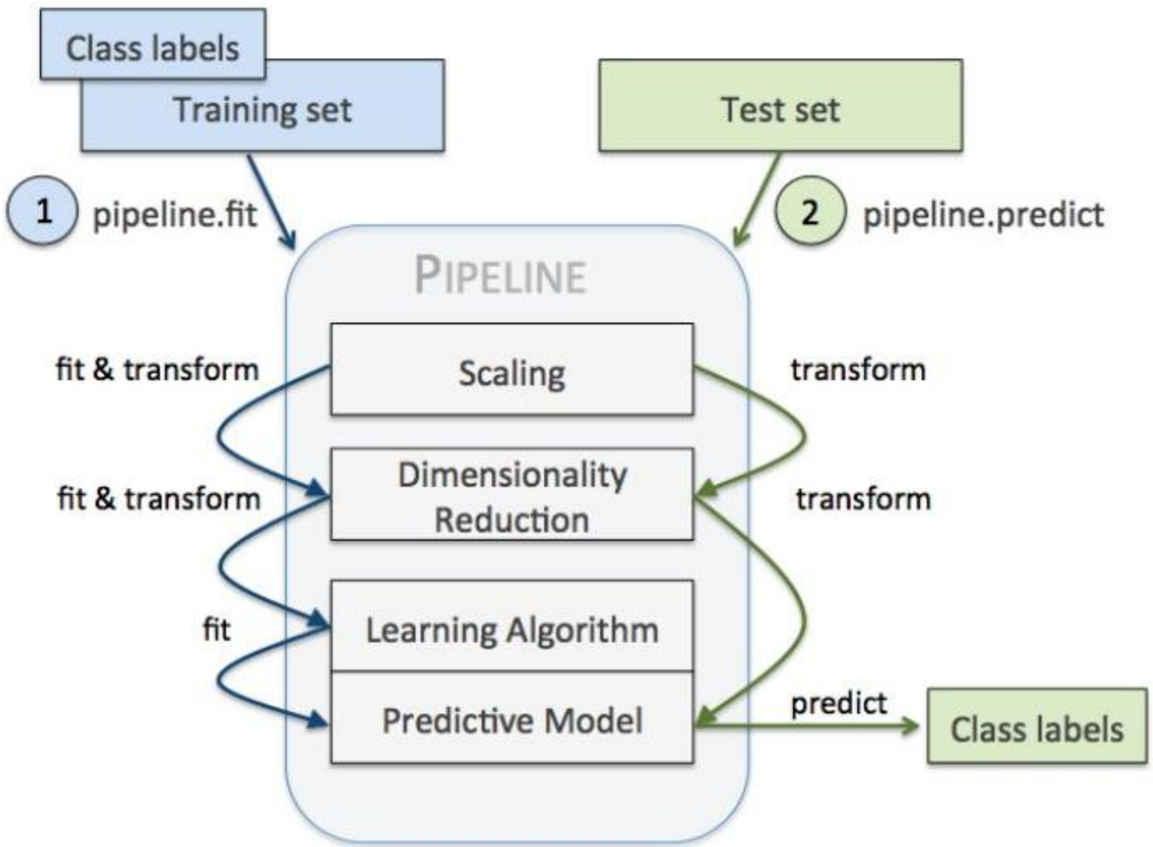
Pipeline:
Workflow Optimization

Pipeline: chaining estimators

- Pipeline can be used to chain multiple estimators into one.
- Pipeline serves two purposes:
  - Convenience and encapsulation
  - Joint parameter selection
- All estimators in a pipeline, except the last one, must be transformers.
  - The last estimator may be any type (transformer, classifier, etc.)

Pipeline: chaining estimators

- Training and prediction procedure of the pipeline



Building Pipelines

```
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
# load and split the data
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC())])
```

The **Pipeline** is built using a list of **(key, value)** pairs, where the **key** is a string containing the name you want to give this step and **value** is an estimator object:

```
pipe.fit(X_train, y_train).score(X_test, y_test)
```

```
0.951048951048951
```

You only have to call **fit** and **predict** once on your data to fit a whole sequence of estimators

Using Pipelines in Grid-searches

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'svm__C': [0.001, 0.01, 0.1, 1, 10, 100],
              'svm__gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

Parameters of the estimators in the pipeline shoud be defined using the **estimator__parameter** syntax

```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation accuracy: {:.2f}".format(
    grid.best_score_))
print("Test set score: {:.2f}".format(grid.score(X_test, y_test)))
print("Best parameters: {}".format(grid.best_params_))
```

```
Best cross-validation accuracy: 0.98
Test set score: 0.97
Best parameters: {'svm__C': 1, 'svm__gamma': 1}
```

Convenient Pipeline creation with *make_pipeline*

```
from sklearn.pipeline import make_pipeline
# standard syntax
pipe_long = Pipeline([("scaler", MinMaxScaler()),
                      ("svm", SVC(C=100))])
# abbreviated syntax
pipe_short = make_pipeline(MinMaxScaler(), SVC(C=100))
```

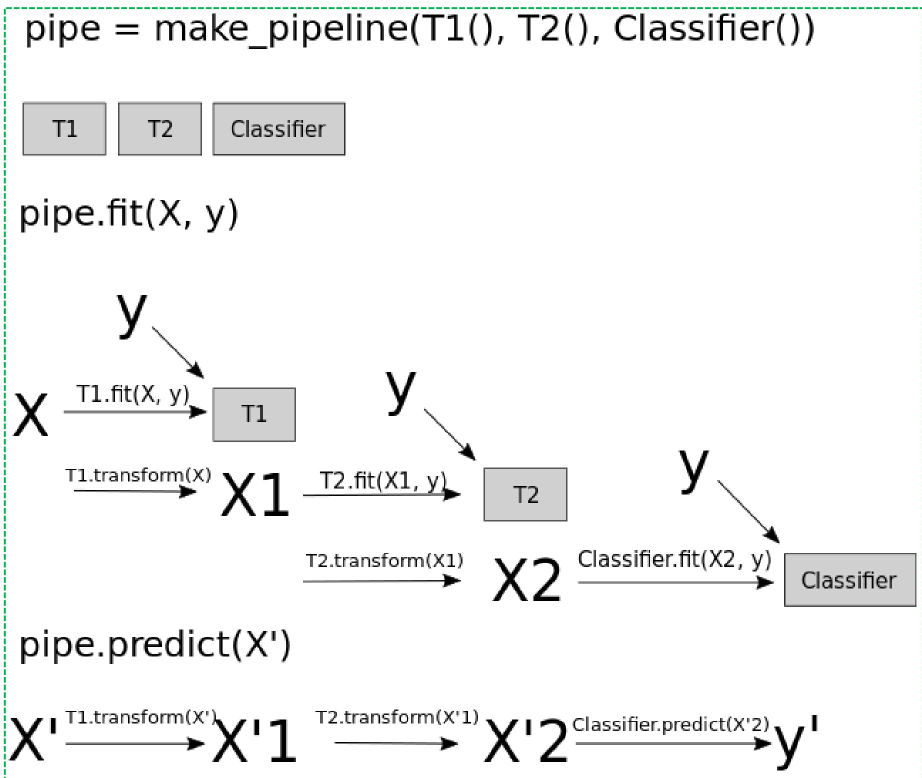
```
print("Pipeline steps:\n{}".format(pipe_short.steps))
```

```
Pipeline steps:
[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('svc',
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))]
```

Make_pipeline does not require, and does not permit, naming the estimators. Instead, their names will be set to the **lowercase of their types** automatically.

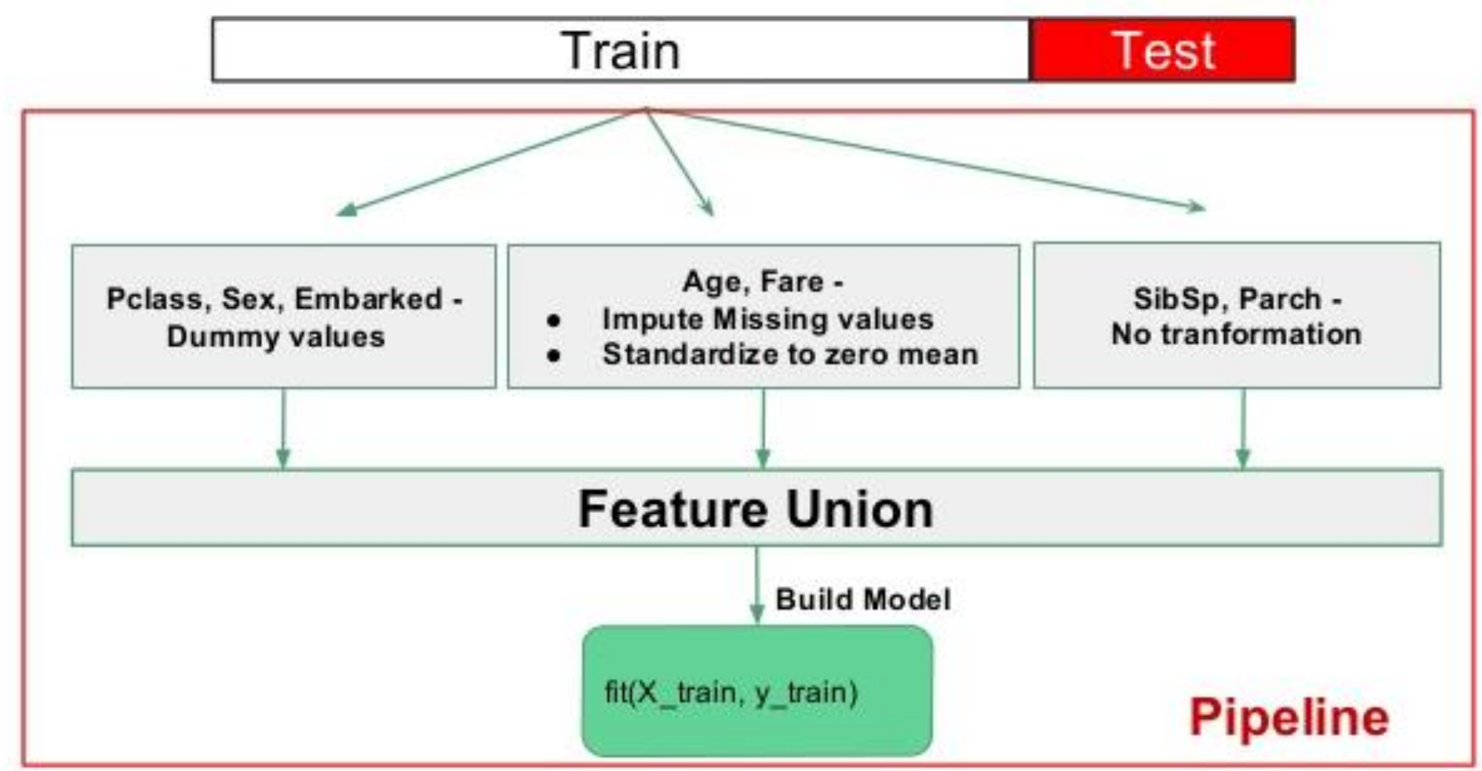
Pipeline Interface

- All estimators in a pipeline, except the last one, must be transformers. The last estimator may be any type (transformer, classifier, etc.)
- Training and prediction procedure of the pipeline



Combining Features with *FeatureUnion*

- FeatureUnion applies a list of transformer objects in parallel to the input data, then concatenates the results.
- This is useful to combine several feature extraction mechanisms into a single transformer.



Learning from Imbalanced Data

Real-life imbalanced problems

Application area	Problem description
Activity recognition	Detection of rare or less-frequent activities (multi-class problem)
Behavior analysis	Recognition of dangerous behavior (binary problem)
Cancer malignancy grading	Analyzing the cancer severity (binary and multi-class problem)
Hyperspectral data analysis	Classification of varying areas in multi-dimensional images (multi-class problem)
Industrial systems monitoring	Fault detection in industrial machinery (binary problem)
Sentiment analysis	Emotion and temper recognition in text (binary and multi-class problem)
Software defect prediction	Recognition of errors in code blocks (binary problem)
Target detection	Classification of specified targets appearing with varied frequency (multi-class problem)
Text mining	Detecting relations in literature (binary problem)
Video mining	Recognizing objects and actions in video sequences (binary and multi-class problem)

Source: "Learning from imbalanced data: open challenges and future directions", Bartosz Krawczyk

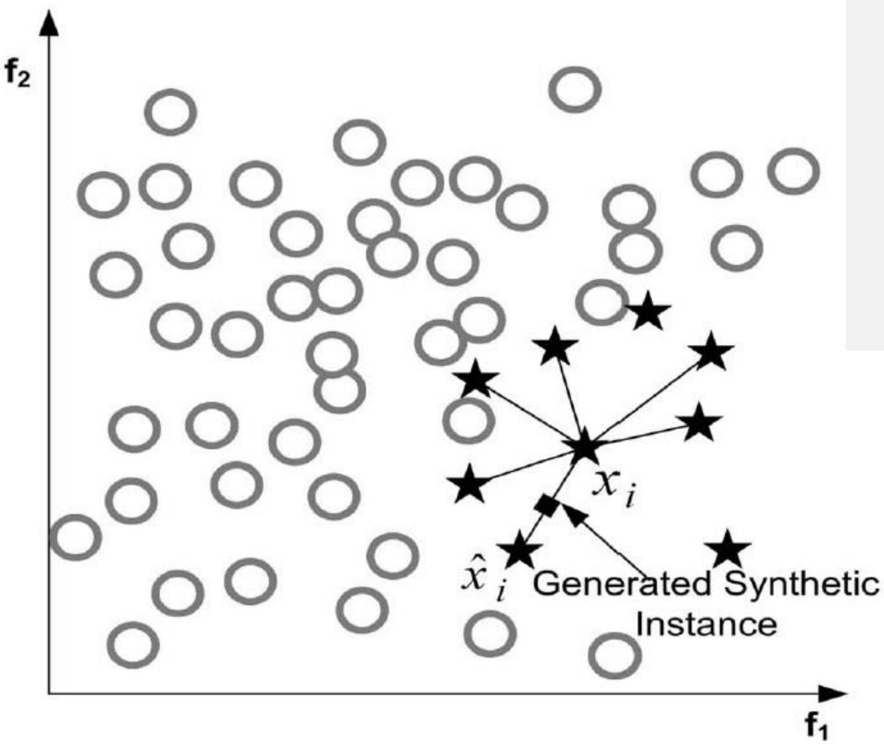
Approaches

- Sampling
  - Random Undersampling
  - Random Oversampling
  - SMOTE
  - Tomek Links
  - SMOTE + Tomek Links
  - Using GAN(Generative Adversarial Networks)
- Algorithms
  - Cost-sensitive learning methods
  - Kernel-based methods

Sampling heuristics

- Consider testing under-sampling when you have an a lot data (tens- or hundreds of thousands of instances or more)
- Consider testing over-sampling when you don't have a lot of data (tens of thousands of records or less)
- Consider testing random and non-random (e.g. stratified) sampling schemes.
- Consider testing different resampled ratios (e.g. you don't have to target a 1:1 ratio in a binary classification problem, try other ratios)

SMOTE

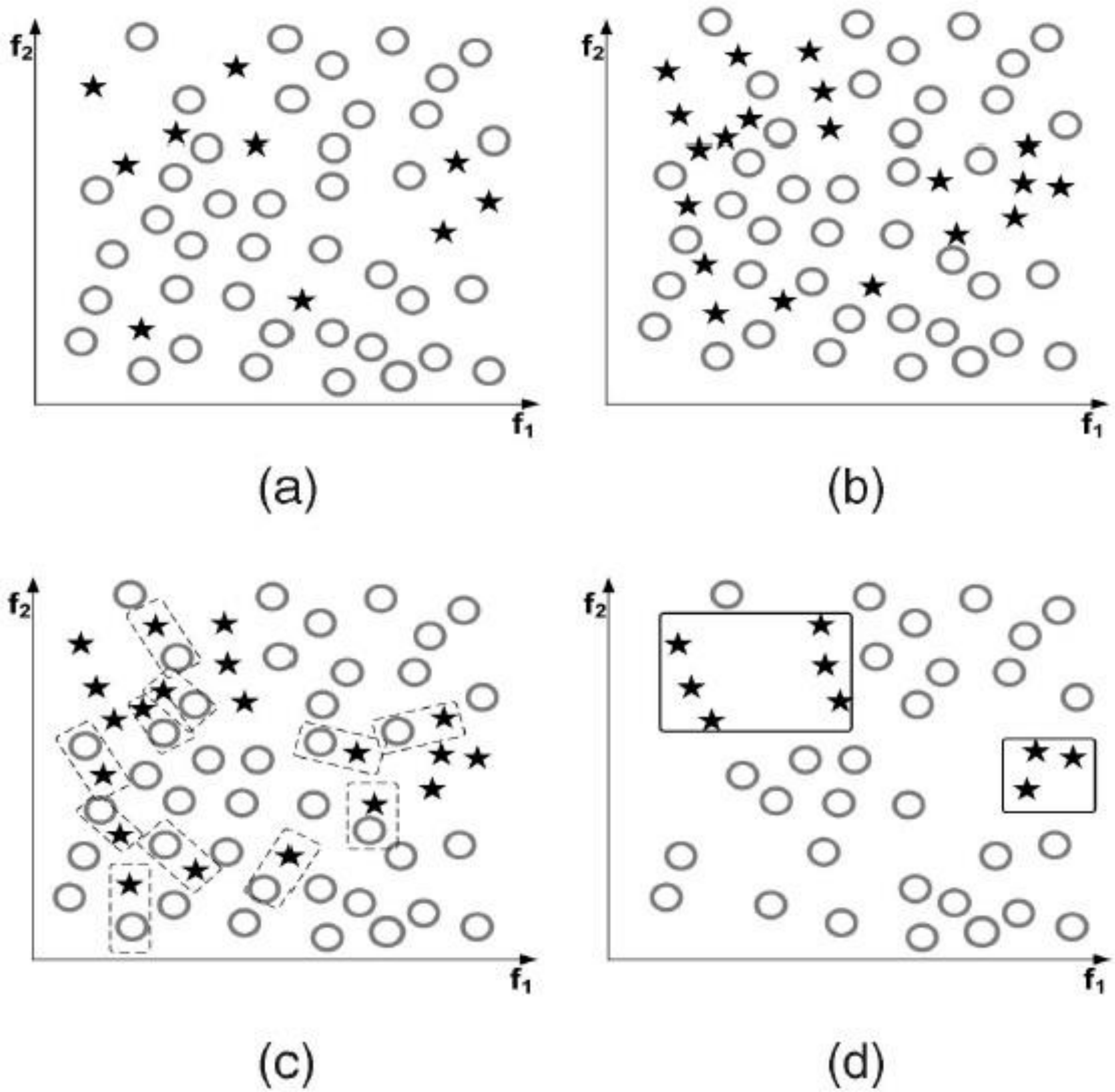


For each point ***p*** in ***S*** (minor class):

1. Compute its ***k*** nearest neighbors in ***S***.
2. Randomly choose ***r*** ≤ ***k*** of the neighbors (with replacement).
3. Choose a random point along the lines joining ***p*** and
   each of the ***r*** selected neighbors.
4. Add these synthetic points to the dataset with class ***S***.

Source: "Survey of resampling techniques for improving classification performance in unbalanced datasets", Ajinkya More

Tomek links



- A pair of examples is called a ***Tomek link*** if they belong to different classes and are each other's nearest neighbors.
- Undersampling can be done by removing all tomek links from the dataset.