AIML430/COMP309: ML Tools and Techniques
Lecture 9: Python EDA Tools, & Python Catchup

Ali Knott
School of Engineering and Computer Science, VUW

What we're doing this week

This week, we're looking at Exploratory Data Analysis (EDA).

- Last lecture we introduced EDA.
  - With a focus on visualisation methods.
- Today: a survey of Python tools for EDA.
  - With some Python background we skipped in Weeks 1-2...
  - With some very useful Python tricks...
- Thursday: EDA demos in Python and Orange.

EDA languages: An overview

You have lots of options...

- R has lots of tools for statistical computation, graphical data analysis.
  - `ggplot2` is particularly useful for data visualisation.
- Weka includes several EDA tools and algorithms.

We're focussing on Python and Orange in this course.

- Python in this lecture... Python and Orange in the tutorial.

Python EDA tools

Some key tools:

- Numerical Python (Numpy): matrix / numerical analysis layer.
  - This is a fundamental library. A key datastructure is ndarray.
- Scientific Python (Scipy): scientific computing utilities/functions.
  - For linear algebra, mathematical approximation...
- Scikit-learn (sklearn): a key machine learning toolbox.
- For plotting and visualisation: Matplotlib, Seaborn.
- For data manipulation and analysis, Pandas is very useful.
  - Provides powerful data structures, including data frame, series.

There are many other EDA-relevant things in Python!

Including...

- OpenCV: computer vision
- Statsmodels: Statistics in Python
- Bokeh, Plotly: Plotting and Visualisation
- NLTK, Gensim: NLP tools
- Theano, Caffe, Pytorch, Lasagne: Deep Learning
- Pybrain, Pylearn2: Machine learning
- DEAP, NEAT-python: Evolutionary Computation

Python aside 1: Anaconda

Anaconda is an *installation* of Python.

- It contains almost everything your need for basic machine learning and data analysis.
- You can install things yourself (with pip), but Anaconda is easier!

Why would you use Anaconda?

- There are often *compatibility issues* with Python & its libraries.
- Anaconda just means you don't have to worry about those.
- Anaconda carries over 7500 open source packages, many of which are't in the pip repository.
- It supports *high performance computing* (Anaconda Accelerate)

Anaconda is big! (Around 3GB).

- For many things, you can get away with Miniconda (400MB).

Python aside 2: IDEs

The environment you develop, run and test code in is called an Interactive Development Environment (IDEA).

A 'traditional' IDE for Python is PyCharm.

- A very good IDE!

But Python can also be used *interactively*: the iPython (or Jupyter) IDE makes innovative use of this.

- You can create notebooks, that combine text and code.
- In these notebooks, the user can run *selected blocks of code, rather than whole programs*.
- Very good for teaching, learning, experimenting!

Using Python modules

Python libraries are called modules. Each must be imported before use.

Three ways to import:

- Import the full module:
  - `import numpy`
- Import selected functions from the module:
  - `from numpy import sin`
- Import all functions from the module:
  - `from numpy import *`
  - Not recommended!

All modules support *shortcuts*:

- E.g. `import numpy as np`

```
In [1]: sin(pi)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-69d9a90af4a5> in <module>()
----> 1 sin(pi)

NameError: name 'sin' is not defined
```

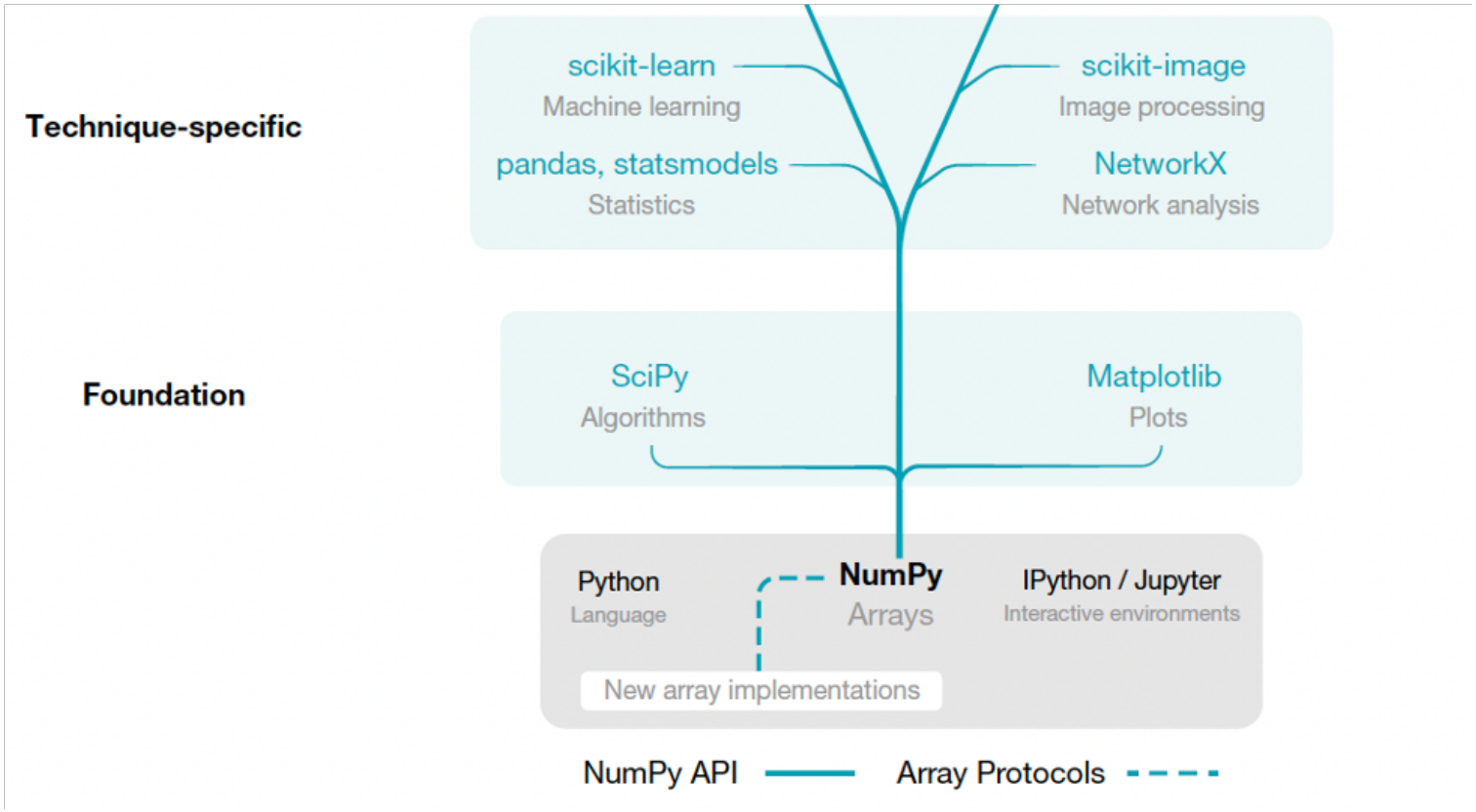
```
In [1]: import numpy as np
        np.sin(np.pi)
Out[1]: 1.2246467991473532e-16
```

```
In [2]: from numpy import sin, pi
        sin(pi)
Out[2]: 1.2246467991473532e-16
```

```
In [3]: from numpy import *
        sin(pi)
Out[3]: 1.2246467991473532e-16
```

The stack of Python tools for EDA / ML / Data Science

Python modules for EDA build on each other.



- NumPy is a core module—especially in implementing arrays.
- SciPy, Matplotlib build on that. . . Pandas, Scikit-learn build further.

NumPy

Numpy is the foundation for scientific computing in Python.

It has many features:

- Its array data structure is powerful for large, multi-dimensional arrays
- It provides basic linear algebra functions (e.g. matrix operations)
- It provides Fourier transforms (useful in signal processing, graphics)
- It has sophisticated random number capabilities
- It has tools for integrating C/C++ code, Fortran code.

What can NumPy do for EDA?

Data manipulation:

- NumPy *arrays* let you efficiently manipulate and process data.
- E.g. *filtering*, *slicing*, *indexing*, *reshaping data*.

Mathematical operations:

- NumPy supports many maths operations (e.g. logarithms, trig).
- You can do *element-wise* operations on whole arrays of data.

Descriptive statistics:

- NumPy offers statistical functions to compute descriptive statistics on your data, e.g., *mean*, *median*, *standard deviation*, *variance*, *min*, *max*, *percentiles*, etc.

Data visualization:

- NumPy routines are used in visualization libraries like Matplotlib and Seaborn.

SciPy

SciPy is widely used in various scientific and engineering disciplines, including physics, chemistry, biology, economics, and data science.

It provides many useful features for EDA. (And for science more widely.)

- Statistical tests: e.g., *t-tests*, *chi-square tests*, *ANOVA*, and *correlation* tests, to assess relationships between variables and identify significant differences in data, and for hypothesis testing to validate assumptions and draw conclusions about the data.
- Outlier detection: see `scipy.stats.zscore`
- Distance calculations: see `scipy.spatial.distance`.

Pandas

Pandas (Python Data Analysis Library) is a python library specifically designed for data analytics.

- It provides easy-to-use data structures and data analysis tools for data manipulation, data cleaning, data exploration, and data preparation tasks.
- It depends on Numpy and Scipy, and (partly) on Matplotlib.

It has many features:

- Powerful tools, that enable analytics *methods*: DataFrame, Series
- *Data cleaning* operations
- Tools for wrangling databases: *re-shape*, *merge* (for joining)
- Database-like operations: filtering, sorting, grouping, aggregating
- Some data visualisation methods too.

Some useful Pandas EDA functions

Data inspection and summary:

- info(), describe(), shape() give summary statistics (including missing information) of a DataFrame.
- head(), tail() let you inspect portions of a DataFrame.

Database-like operations:

- loc[] and iloc[] are filtering/selection tools, that let you extract portions of a DataFrame that match a Boolean query.

Grouping and aggregation functions:

- groupby() lets you group data based on one or more columns and apply aggregate functions like sum(), mean(), count().
- pivot(), melt(), stack() unstack() let you *reshape* datasets.

Pandas also gives basic visualisation for DataFrames, with plot().

Matplotlib

Matplotlib is a comprehensive graphics library for generating scientific figures.

- It generates high-quality figures in all major graphics formats. (PNG, SVG, PDF etc.)
- It works efficiently with data frames and arrays.
- It includes a GUI (Graphical User Interface), for interactively exploring figures.
  - But it also supports figure generation without the GUI.
- It integrates well with Jupyter notebooks. . .
- It supports 3D figures.

Seaborn

Seaborn is a visualisation library based on Matplotlib.

- Its default style is more polished—figures come out looking better
- It's more convenient to use on a Pandas DataFrame
- . . . but 3D is not supported.

For basic plots, use Matplotlib. . .
For more advanced statistical visualisations, use Seaborn!

Scikit-learn

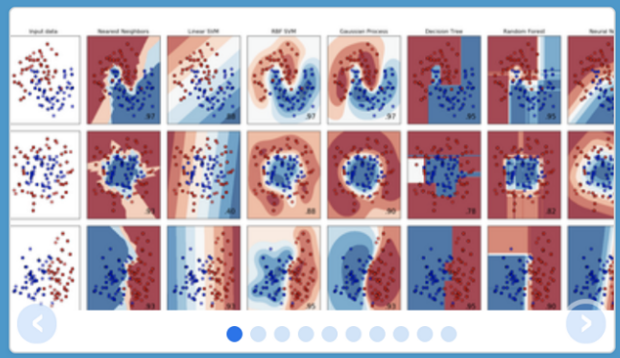
Scikit-learn (sklearn) stands for 'SciPy Toolkit'.

- 'It's an open source library that provides a consistent API for using traditional state-of-the-art ML algorithms/methods in Python.'
- It's mostly written in Python. But some internal routines are written in Cython ('C extensions for Python').
- Sklearn's major Python dependencies are scipy, numpy and matplotlib.

scikit learn

Home Installation Documentation Examples

Google Custom Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, … — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

— Examples

Some useful Scikit-learn functions

Scikit-learn is useful for a few stock ML tasks.

- *Class balancing* (if your classes are of unequal size):
  - resample() is useful for that.
- *Scaling a dataset*:
  - scale() is the key function for that.
- *Selecting useful features*:
  - variance_threshold() lets you identify low-variance features...
  - selectKBest() is a high-level feature-selection function.

All these operations are classed as transformers in sklearn.

- But transformers also include learning functions.

Pipelines in Scikit-learn

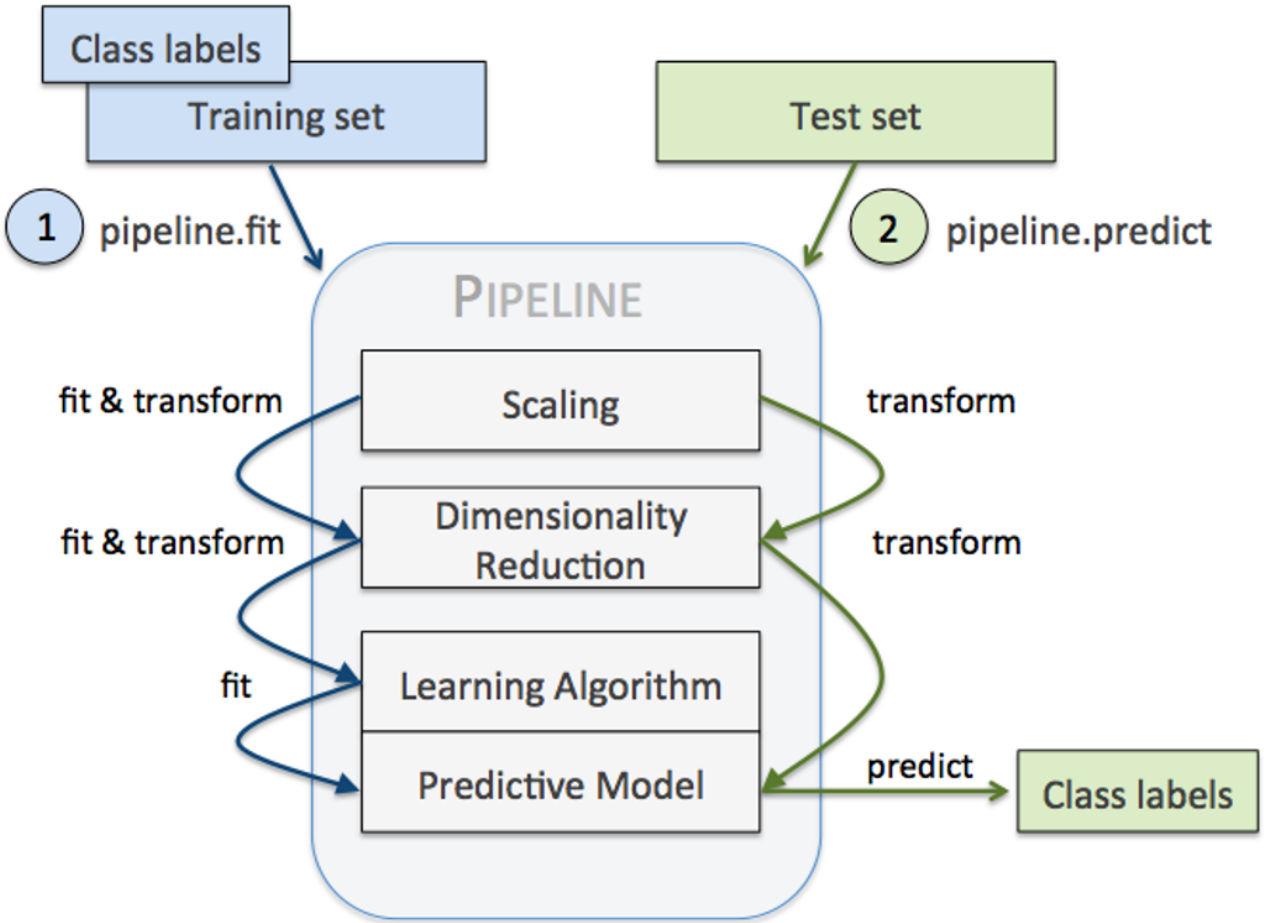
Pipelines are a good way to *structure* a data-processing/ML project.

The raw data must be processed before it's used to train on.

- That's done by pipeline.fit.

The same processing must be done on the test data.

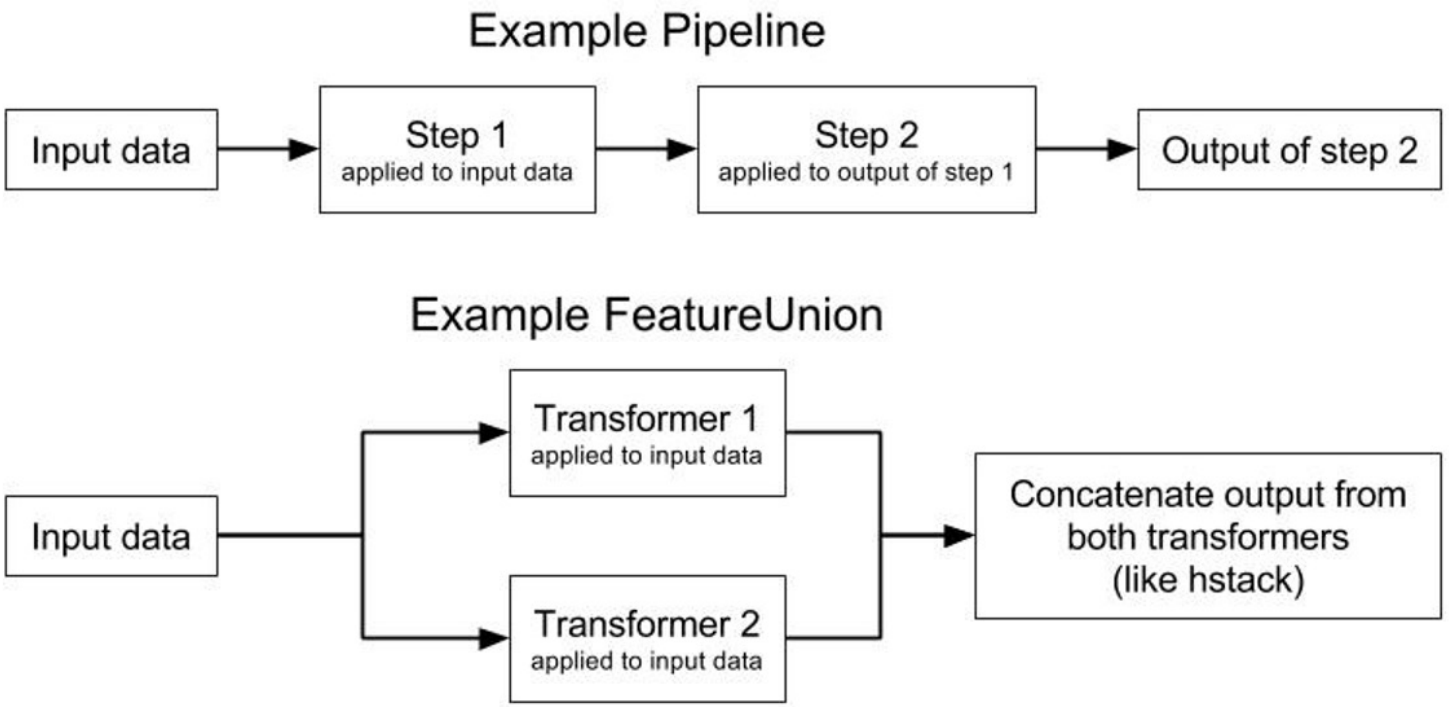
- That's done by pipeline.predict.



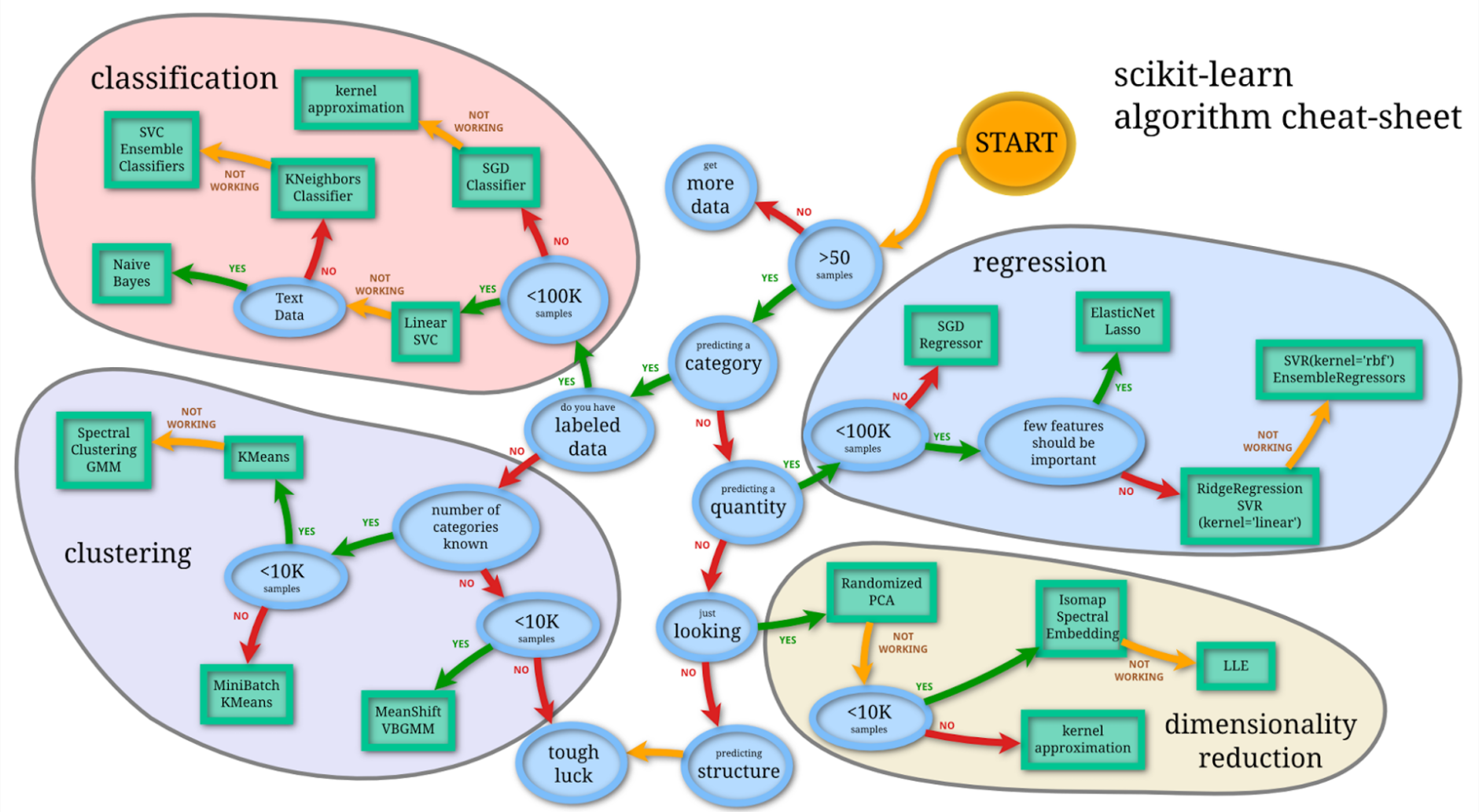
Parallelism in pipelines

Often, we want to transform different columns of our dataset in different ways.

- Scikit-learn supports that with the featureUnion() function.
- This takes a set of transformers, and applies them in parallel to the input, concatenating the results into a new table.



Scikit-learn cheat sheet



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```

Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

DataCamp
Learn Python for Data Science Interactively