Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Machine Learning

Lecture 5: Scikit Learn

Introduction to SciKit Learn

- Scikit-learn provides a range of supervised and unsupervised learning algorithms in Python.

  - The library is built upon the following:
  - **NumPy**: Base n-dimensional array package
  - **SciPy**: Fundamental library for scientific computing
  - **Matplotlib**: Comprehensive 2D/3D plotting
  - **Pandas**: Data structures and analysis

- The library is focused on modelling data. It is not focused on loading, manipulating and summarizing data.
- For these features, you need to use the techniques from NumPy and Pandas.

Introduction to Scikit Learn

- **Tutorials and API pages for Scikit-Learn are available online.**
- **The following are the main components are Scikit-learn.**
- **Classification**: a large collection of learning algorithms such as naive bayes, lazy methods, neural networks, support vector machines and decision trees.
- **Clustering**: for grouping unlabelled data such as KMeans.
- **Regression**: libraries for predicting real-valued attributes such as multiple linear regression, ridge regression, etc.
- **Pre-processing**: Outlier detection, normalization, encoding categorical features
- **Dimensionality Reduction**: Reduces the number of features that you need to consider in your dataset.
- **Model Selection**: Comparing, validating and choosing parameters and models.

Introduction to Scikit Learn

- The following are some important requirements that you should keep in mind when working with Scikit learn.
  - Features and classes are **separate** objects (data structures)
  - Features and classes should be **continuous valued**
  - Features and classes should be **NumPy** arrays
  - Features and classes should have a **specific shape**
    - Features should be 2D (Columns correspond to numbers of features and rows is number of data instances)
    - Class array should be one dimensional with same number of instances as there are data instances in the features array

Using Datasets

- Scikit-learn comes with a number of standard example datasets, including the iris dataset and digits datasets for classification and the Boston house prices dataset for regression.

- These datasets are **dictionary-like** objects holding at least two items:
  - A NumPy array of shape *n_samples* * *n_features* with the key ***data***
  - A NumPy array of length *n_samples*, containing the class values, with key ***target***.

- The datasets also contain a description in DESCR and some contain feature_names and target_names.

Using datasets

```
from sklearn import datasets

iris = datasets.load_iris()

print (iris.data)
print (iris.target)
print ( iris.data[:, [2,3]] )
print ( iris.data.shape )
print ( iris. DESCR )
```

Load iris dataset into a dataset object

Accesses the data stored in the dataset object (2D numpy array)

Accesses the class associated with each data item

Accesses all rows of the dataset but just columns with index 2 and 3

Outputs the dimensions of the data in this case (150, 4)

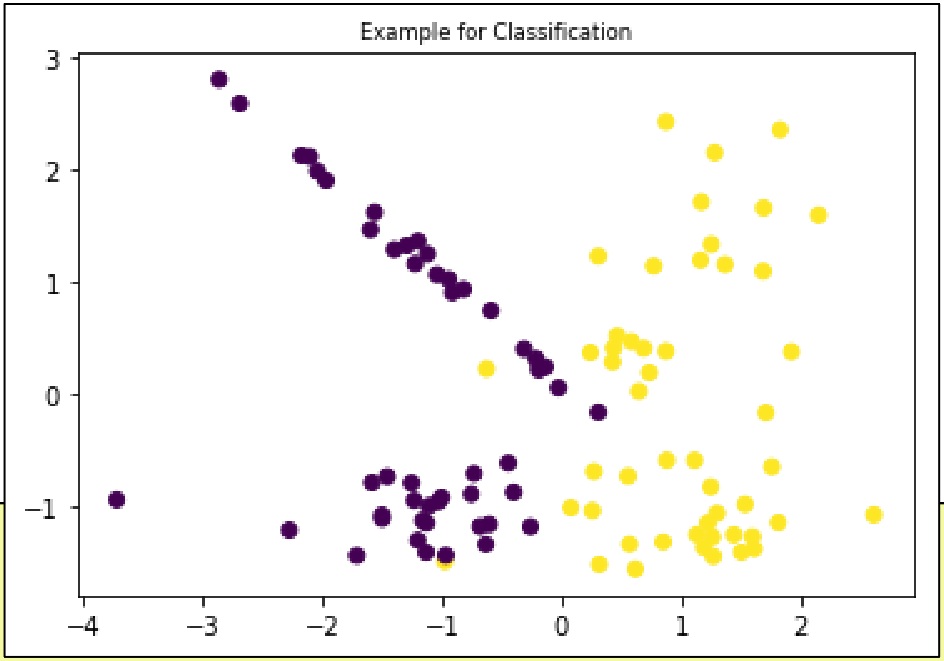
Sample Generators

- In addition, scikit-learn includes various random sample generators that can be used to build artificial datasets of controlled size and complexity.

- For example, **datasets.make_classification**

  - **n_samples : int**, optional (default=100) The number of samples.
  - **n_features : int**, optional (default=20) The total number of features.
  - **n_informative : int**, optional (default=2) The number of informative features.
  - **n_redundant : int**, optional (default=2) The number of redundant features.
  - **n_classes : int**, optional (default=2) The number of classes (or labels) of the classification problem.

Sample Generators




```
from sklearn import datasets
import matplotlib.pyplot as plt

plt.title("Example for Classification", fontsize='small')

X1, Y1 = datasets.make_classification(n_features=2, n_redundant=0, n_informative=2)

plt.scatter(X1[:, 0], X1[:, 1], marker='o', c=Y1)
plt.plot()
```

Incorporating External Data

```
titanic = pd.read_csv("titanic.csv")

data = titanic[["Sex","Fare","Age"]]
target = titanic[["Survived"]]

target = target[~data["Age"].isnull()]
data = data[~data["Age"].isnull()]

data["Sex"] = data["Sex"].map({"male": 0, "female": 1})
```

Read Dataframe using Pandas

Select and extract features for classification

Select and extract classification target

Make sure all features and targets are valid

Transform categories into numerical values where necessary

Basic classification

- Scikit Learn provides a wide range of classification algorithms, that can be used as black box (to a degree)

- All classification algorithms implement at least the following interface

  - **fit(training_features, training_labels)**

    This function is called first to learn the statistics of the feature set based on the line-by-line correspondence with the associated labels

  - **predict(new_features)**

    After the classifier has been trained with the fit function, we call this function to predict labels for new features

Basic classification

```
digits = datasets.load_digits()

clf = svm.SVC()

training_features = digits.data[:-10]
training_labels = digits.target[:-10]
clf.fit(training_features, training_labels)

test_features = digits.data[-10:]
test_labels = digits.target[-10:]
predictions = clf.predict(test_features)

print(test_labels)
print(predictions)
```

Create the classifier object (e.g. Support Vector Machine)

Train the classifier on the training data

Predict labels for the test data

Compare test data with the predictions

```
[5 4 8 8 4 9 0 8 9 8]
[5 4 5 5 4 9 5 5 5 5]
```

Serialisation

- A classifier can be serialised after being trained, enabling it to be stored or transferred to other machines

```
import pickle

clf = svm.SVC()
clf.fit(training_features, training_labels)

s = pickle.dumps(clf)

# Transfer the serialised and trained classifier

clf2 = pickle.loads(s)
predictions = clf2.predict(test_features)
```

Import pickle, the library for serialisation of Python objects

Create a string encoding the complete state of the classifier

De-serialise the string back into a classifier and use it for prediction

Serialisation

- Less flexible, but more efficient (in particular for larger instances) is the use of *joblib*, which allows to write a trained classifier to a file
- *It training the classifier takes very long, this can save a lot of time*

```
from joblib import dump, load

clf = svm.SVC()
clf.fit(training_features, training_labels)

dump(clf, 'filename.joblib’)

# Do something else, potentially terminating the program

clf2 = load('filename.joblib’)
predictions = clf2.predict(test_features)
```

Import dump and load from joblib

Save the state of the trained classifier to a file

Load the state again, ready for making predictions

Basics of using Scikit Learn

- In the following slides we look at three separate scenarios for evaluation:

1. There is a separate **training** and **test** dataset that can be used.

2. A **single** dataset into training and test data (holdout method).

3. We use **cross validation** in order to evaluate the accuracy of an algorithm.

Basics of using Scikit Learn

1. There is a separate **training** and **test** dataset that can be used.

f1	f2	f3	…	fn	class

Training Set

f1	f2	f3	…	fn	class

Testing Set

Assessing Accuracy

- Assuming I have separate training and test data I might have the following arrays
  - training_features, training_labels
  - test_features, test_labels

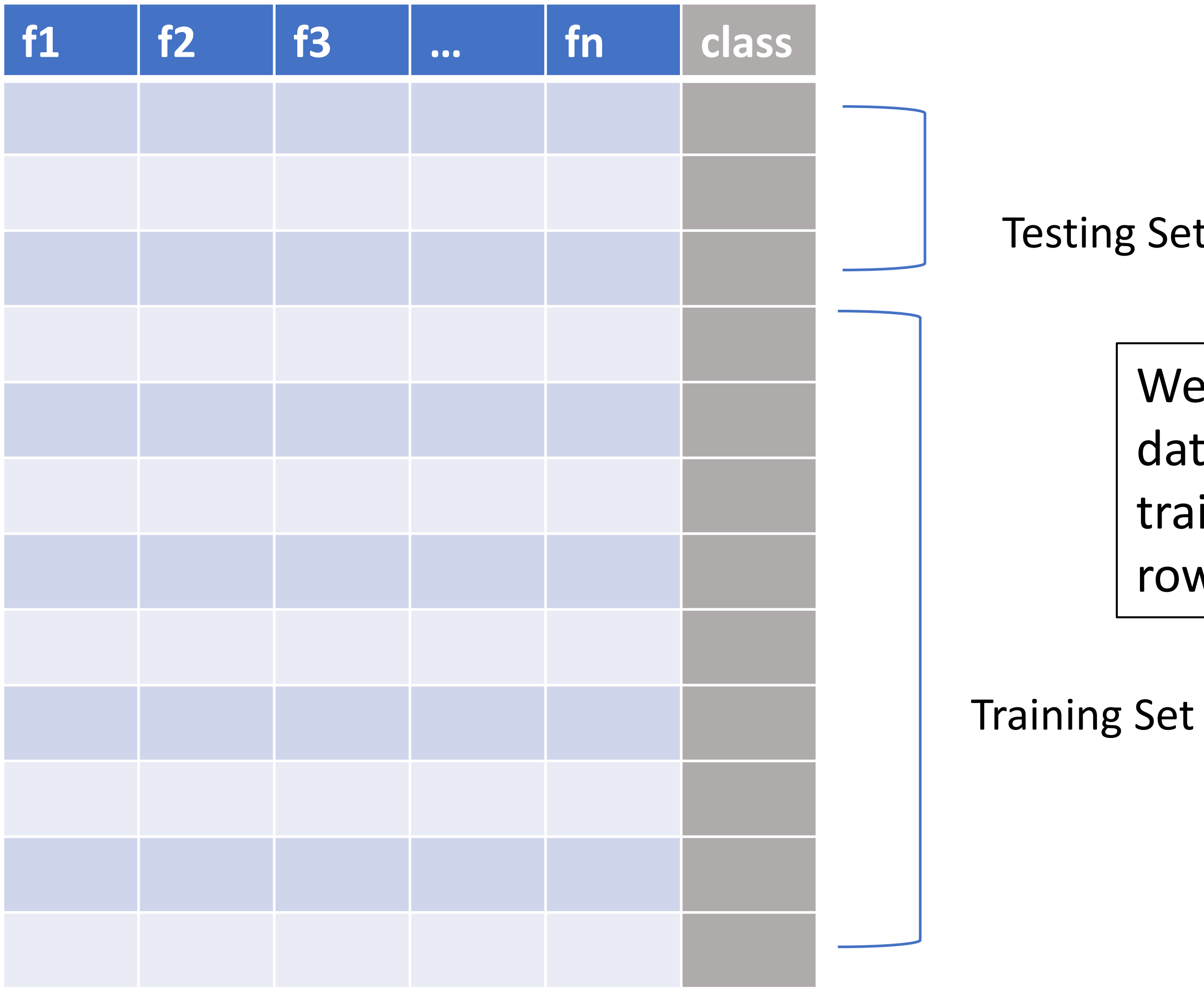
```
from sklearn import metrics
from sklearn import svm

clf = svm.SVC()
clf.fit(training_features, training_labels)

predictions = clf2.predict(test_features)

print(test_labels)
print(predictions)
print (metrics.accuracy_score(predictions, test_labels))
```

The accuracy_score function will count the number of classes we correctly predicated and express that as a **percentage** of the total number of test data instances.

```
[5 4 8 8 4 9 0 8 9 8]
[5 4 5 5 4 9 5 5 5 5]
0.4
```

Basics of using Scikit Learn

2. We split a **single** dataset into training and test data.



We split a single dataset into test and train data we select the rows **randomly**.

Assessing Accuracy (Splitting Training Data)

- In scikit-learn a random split into training and test sets can be quickly computed with the *train_test_split* helper function.
- As arguments we pass it the **original data** and **target** as well as the **percentage of the original data** we want for the training data. We also pass it a random seed.

```
from sklearn import model_selection
from sklearn import datasets

iris = datasets.load_iris()
print (iris.data.shape, iris.target.shape)

test_features, train_features, test_labels, train_labels = model_selection.train_test_split(
        iris.data, iris.target, test_size=0.8, random_state=0)

print (train_features.shape, train_labels.shape)
print (test_features.shape, test_labels.shape)
```

```
(150, 4) (150,)

(120, 4) (120,)
(30, 4) (30,)
```

80/20 split

Random seed

The drawback of the holdout method

- In typical machine learning applications, we are also interested in tuning and comparing different parameter settings to further improve the performance for making predictions on unseen data (hyper-parameter optimization).
- This process is called model selection, where the term model selection refers to a given classification problem for which we want to select the optimal values of tuning parameters (also called hyper-parameters).
- However, if we reuse the same test dataset over and over again during model selection, the model will just overfit on the training data using the hyper-parameters.
- Unfortunately, despite this very significant issue, many people still use the test set for model selection, which is not a good machine learning practice.

The drawback of the holdout method

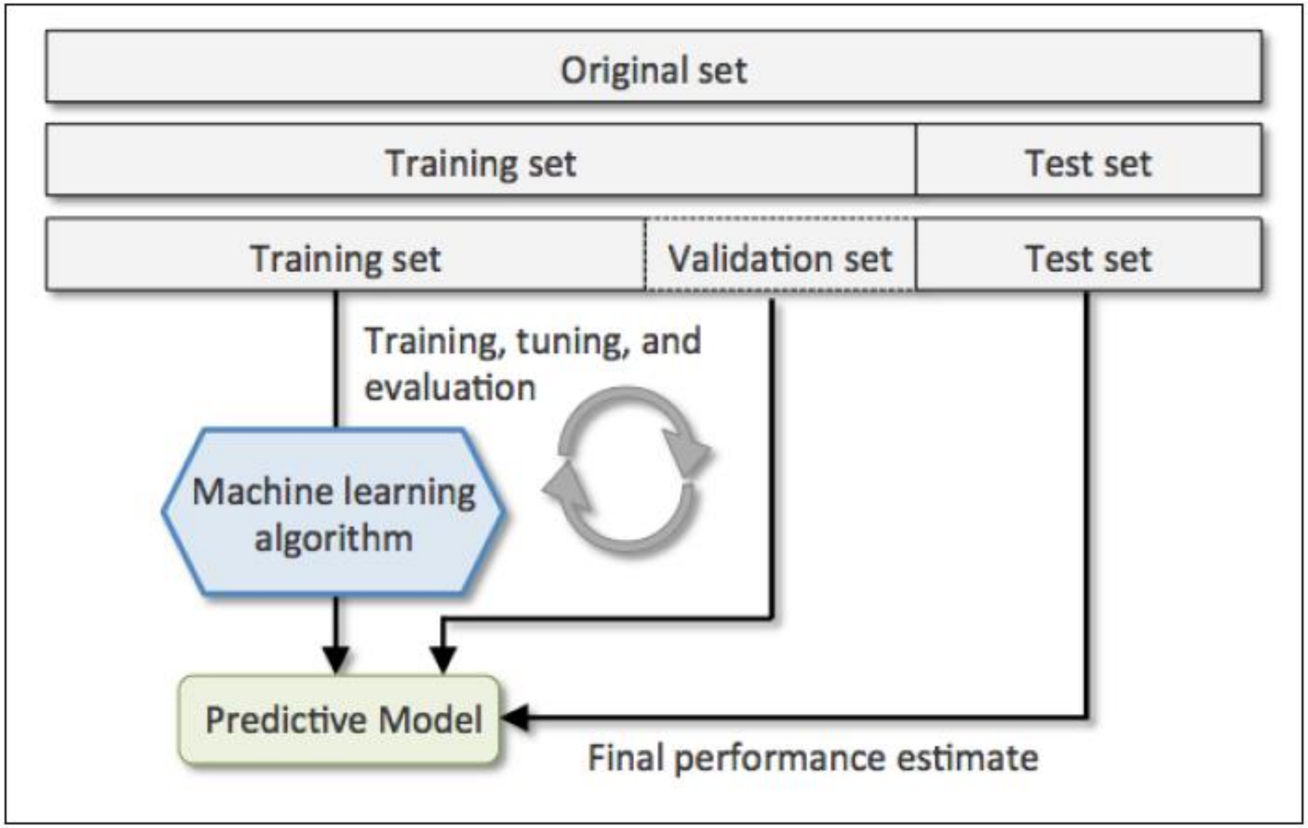
- A better way of using the holdout method for model selection is to separate the data into three parts: a **training** set, a **validation** set, and a **test** set.

- The training set is used to fit the different models, and the performance on the validation set is then used for the model selection (see next slide).

- The advantage of having a test set that the model hasn't seen before during the training and model selection steps is that we can obtain a less biased estimate of its ability to generalize to new data.

- This method is referred to as holdout cross validation.

Holdout cross-validation



Use a validation set to repeatedly evaluate the performance of the model after training using different parameter values. Once we are satisfied with the tuning of parameter values, we estimate the models' generalization error on the test dataset. Any drawbacks??

The drawback of holdout cross-validation

- A disadvantage of the holdout method is that the performance estimate is sensitive to how we partition the training set into the training and validation subsets.

- In other words the final accuracy estimate will vary for different samples of the data.
- Solution: Randomise this process and aggregate over many different samples of the data (k-fold cross validation)

Thank you for your attention