COMP7630 – Web Intelligence and its Applications

Introduction to SciKit-Learn

Outline

- Introduction to Scikit-learn
- Data Preprocessing and Dimensionality Reduction
- Classification
- Clustering

Introduction to Scikit-learn

- Scikit-learn
  - Machine learning library built on Numpy and Matplotlib

- What Scikit-learn can do
  - Unsupervised learning
    - Clustering
  - Supervised learning
    - Regression, classification
  - Data preprocessing
    - Feature extraction, feature selection, dimensionality reduction

Introduction to Scikit-learn

- What Scikit-learn cannot do
  - Distributed computation on multiple computers
    - Only multi-core optimization
  - Deep learning
    - Use Keras and Tensorflow instead

Introduction to Scikit-learn

- Scikit learn models work with structured data
  - Data must be in the form of 2D Numpy arrays
    - Rows represent the samples or instances (in our scenarios they usually are vectorized texts)
    - Columns represent the attributes or features
- ▪ This table is called *features matrix* or *data matrix*

shape = (3, 3)

	Price	Quantity	Liters
Sample 1	1.0	5	1.5
Sample 2	1.4	10	0.3
Sample 3	5.0	8	1

Introduction to Scikit-learn

- Features can be
  - Real values
  - Integer values to represent categorical data
- If you have texts or strings in your data:
  - if long texts, vectorize them using word or sentence embeddings (as seen)
  - if they are just categorical strings, convert them to integers (preprocessing)

Input data

1.0	**January**	1.5
1.4	**February**	0.3
5.0	**March**	1

→

Features matrix

1.0	**0**	1.5
1.4	**1**	0.3
5.0	**2**	1

Introduction to Scikit-learn

- Also missing values must be solved before applying any model
  - With imputation or by removing rows

Input data

1.0	0.5	1.5
1.4	**NaN**	0.3
5.0	0.5	1

→

Features matrix

1.0	0.5	1.5
1.4	**0.5**	0.3
5.0	0.5	1

Input data

1.0	0.5	1.5
1.4	**NaN**	0.3
5.0	0.5	1

→

Features matrix

1.0	0.5	1.5
5.0	0.5	1

  - ... but this rarely happens with vectorized texts, because we "artificially" build the features

Introduction to Scikit-learn

- For unsupervised learning you only need the features matrix
- For supervised learning you also need a target array to train the model
  - It is typically one-dimensional, with length n_samples

Features matrix
shape = (n_samples, n_features)

1.0	5	1.5
1.4	10	0.3
5.0	8	1

Target array
shape = (n_samples,)

A
A
B

Introduction to Scikit-learn

- The target array can contain
  - Integer values, each corresponding to a class label

Target labels
Dog
Dog
Cat

→

Target array
0
0
1

  - Real values for regression

Target array
0.4
1.8
-6.9

Introduction to Scikit-learn

- Scikit-learn estimator API
  - All models are represented with Python classes
  - Their classes include
    - The values of the hyperparameters used to configure the model
    - The values of the parameters learned after training
      - By convention these attributes end with an underscore
    - The methods to train the model and make inference
  - Scikit-learn models are provided with sensible defaults for the hyperparameters

Introduction to Scikit-learn

- Scikit learn models follow a simple, shared pattern
  1. **Import** the model that you need to use
  2. **Build** the model, setting its hyperparameters
  3. **Train** model parameters on your data
     - Using the **fit method**
  4. **Use** the model to make predictions
     - Using the **predict/transform methods**

- Sometimes fit and predict/transform are implemented within the same class method

Predictors vs transformers

- In scikit-learn, we separate estimators into:
  - Predictors
    - An estimator supporting predict() and/or fit_predict()
    - Used to predict values, generally after a training (fitting) step
    - Includes classifiers, regressors, outlier detectors, clusterers
  - Transformers
    - An estimator that supports transform() and/or fit_transform()
    - Used to compute a new representation of the same data
    - E.g., Min-Max scaling, PCA, Feature discretizers, Tf-idf

Methods for predictors

- fit(): learn model parameters from input data
  - E.g. train a classifier on a training set
- predict(): apply model parameters to make predictions on data
  - E.g. predict class labels for new samples
- fit_predict(): fit model and make predictions
  - E.g. apply clustering to data

Methods for transformers

- fit(): learn model parameters from input data
  - E.g. learn minimum value and maximum value for each feature, in Min-Max scaler
- transform(): transform data into a different representation
  - E.g. rescale input data to the [0, 1] range
- fit_transform(): fit model and transform data
  - E.g. apply PCA to transform data

Outline

- Introduction to Scikit-learn
- <u>Data Preprocessing and Dimensionality Reduction</u>
- Classification
- Clustering

Data Pre-processing

- Examples:
  - min-max scaling: MinMaxScaler
  - standardization to z-score: StandardScaler

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler


minmax_s = MinMaxScaler()
zscore_s = StandardScaler()
```

Data Pre-Processing

- Applying pre-processing correctly to train and test sets

In [1]:
```
X_train = [[0, 10], [0, 20], [2, 10], [2, 20]]
X_test = [[1, 15]]

minmax_s.fit(X_train) # NOTE: “learning” on training data only!
X_train_norm = minmax_s.transform(X_train)
X_test_norm = minmax_s.transform(X_test) # correct
X_test_wrong = minmax_s.fit_transform(X_test) # do not fit on test
print(X_test_norm)
print(X_test_wrong)
```

Out[1]:
```
[[0.5 0.5]]
[[0, 0]]
```

Dimensionality Reduction

- Useful when you want to reduce the number of features for high-dimensional data
  - For graphical representations
  - Before applying classification and clustering to give the features matrix a more compact representation

Dimensionality Reduction

- PCA with Scikit-learn

```
from sklearn.decomposition import PCA


pca = PCA(n_components=5)
X_projection = pca.fit_transform(X)
```

- *n_components* specify the number of components that you want to keep after applying PCA
  - Should be <= the number of initial features
  - … but if it is a real number in (0,1) it is interpreted as the percentage of variance to be explained
- The result is a features matrix with the specified number of features

Dimensionality Reduction

- Applying PCA and then a classifier

```
pca = PCA(n_components=6)
X_projection = pca.fit_transform(X_train)
my_classifier.train(X_projection, y_train)

# PCA is already fit on training data: do not fit it on test set!
X_test_proj = pca.transform(X_test)
y_test_pred = my_classifier.predict(X_test_proj)
```

- … supposing we already split our dataset in training and test sets

Outline

- Introduction to Scikit-learn
- Data Preprocessing and Dimensionality Reduction
- <u>Classification</u>
- Clustering

Classification

- Classification:
  - Given a 2D features matrix X
    - X.shape = (n_samples, n_features)
  - The task consists of assigning a class label y_pred to each data sample
    - y_pred.shape = (n_samples)

1.0	5	1.5
1.4	10	0.3
...	...	...

X

A
B
B

y_pred

Classification

- By following the estimator API pattern:
  - Import a model

```
from sklearn.tree import DecisionTreeClassifier
```

  - Build model object

```
clf = DecisionTreeClassifier()
```

Classification

- Important decision tree hyperparameters:

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth = 10,
                             min_impurity_decrease=0.01)
```

- Hyperparameters:
    - max_depth: maximum tree height
        - Default = None
    - min_impurity_decrease: split nodes only if impurity decrease above threshold
        - Default = 0.0

Classification

- Train model with ground-truth labels

```
clf.fit(X_train, y_train)
```

  - This operation builds the decision tree structure
    - X_train is the 2D Numpy array with input features (features matrix)
    - y_train is a 1D array with ground-truth labels

6.1	3.1	2
1.8	12	0.15
...	...	...

X_train

0
2
1

y_train

Classification

- Predict class labels for new data

```
In [1]: y_pred = clf.predict(X_test)

Out[1]: [3, 1, 1, 1, 2, 2, 0]
```

- This operation shows the capability of classifiers to make predictions for unseen data

1.0	5	1.5
1.4	10	0.3
...	...	...

X_test

→

1
3
3

y_pred

Classification

- To choose the most appropriate machine learning model for your data you have to evaluate its performance

- Evaluation can be performed according to a metric (scoring function)
  - E.g. accuracy, precision, recall

- To avoid overfitting evaluation must be performed on data that is not used for training the model
  - Divide your dataset into training and test set to simulate two different samples in the data distribution

Classification

Also "stratify" and "shuffle" parameters are very useful.
Stratification maintains the proportion of classes distribution in the splitting.
Shuffling allows to randomly shuffle the dataset before the splitting

You can any number of numpy matrices (X, y, z, W, etc.) and all of them will be splitted consistently to each other. **Useful if you have a numpy vector containing textual strings corresponding to X data matrix.**

- Simple train-test split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

- Evaluation = compare the following two vectors
  - y_test: the expected result (ground truth)
  - y_test_pred: the prediction made by your model

```
from sklearn.metrics import accuracy_score,
                            precision_recall_fscore_support

acc = accuracy_score(y_test, y_test_pred)
p, r, f1, s = precision_recall_fscore_support(y_test, y_test_pred)
```

Classification

- A correct crossvalidation with preprocessing steps

```
In [56]: from sklearn.pipeline import Pipeline

In [57]: #'X' is a data matrix already loaded, while 'y' is its corresponding labels vector.

In [58]: clf_plus_preprocessing = Pipeline( steps = [ ( 'standardization', StandardScaler()      ),
    ...:                                              ( 'classification',  LogisticRegression() )  ] )

In [59]: cv = RepeatedStratifiedKFold(n_repeats=10, n_splits=5)

In [60]: accuracies = cross_val_score(clf_plus_preprocessing, X, y, cv=cv, scoring='accuracy')

In [61]: accuracies.mean()
Out[61]: 0.8725
```

A Sklearn pipeline is a sequence of zero or more preprocessors + (optionally) one classifier as very last element

Run cross validation using the defined pipeline.
The implementation of the `Pipeline` object ensures that no data of the validation set is used for preprocessing!!!

Outline

- Introduction to Scikit-learn
- Data Preprocessing and Dimensionality Reduction
- Classification
- <u>Clustering</u>

Clustering

- Import a model

```
from sklearn.cluster import KMeans
```

- Build model object

```
km = KMeans(n_clusters = 5)
```

  - The hyperparameter n_clusters specifies the number of centroids (= number of clusters)
  - Default is 8 (buy may change across different library versions)

Clustering

- Apply clustering to input data

```
In [1]:  y_pred = km.fit_predict(X)

Out[1]:  [3, 1, 1, 1, 2, 2, 0]
```

- This operation assigns data to their respective cluster
  - X is the 2D NumPy array with input features (features matrix)
  - y_pred is a 1D array with cluster labels

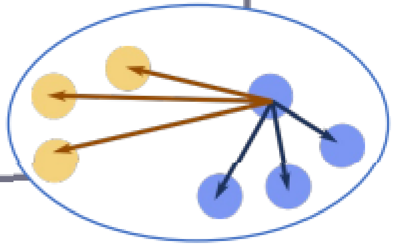
1.0	5	1.5
1.4	10	0.3
...	...	...

3
1
1

Clustering

- Assessing clustering results
  - Internal metrics: use only the information of the features matrix
    - E.g. Silhouette score

```
from sklearn.metrics import silhouette_score, silhouette_samples
silh_avg = silhouette_score(X, clusters)
silh_i = silhouette_samples(X, clusters)
```



  - Silhouette is a number in the range [-1, 1]
  - Higher values mean higher cluster quality
    - Clusters are well separated and cohesive

References

- User guide of Scikit-learn
  - https://scikit-learn.org/stable/user_guide.html
- API reference of Scikit-learn
  - https://scikit-learn.org/stable/api/index.html