This document is used as a brief introduction for students to use scikit-learn in machine learning scenarios.

1. Scikit-learn: A Brief Introduction

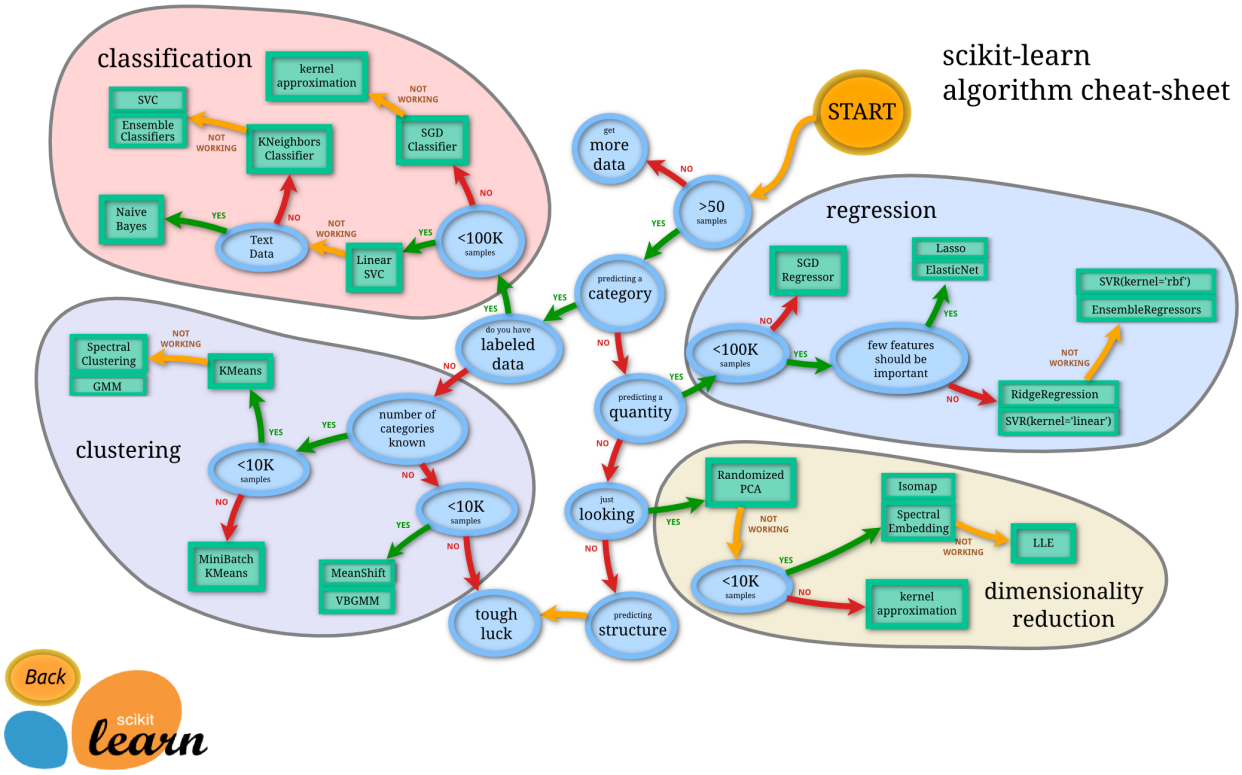
`sklearn` (short for scikit-learn) is a module basing on `numpy` and `scipy`. This project is initialized by `David Cournapeau` in 2007. After years development by the community, it is already one of the most powerful package in Python.

Basically, this package contains all the well-used tools in a complete data mining or machine learning process.

Link to the project: [Github/scikit-learn]

1.1. What does sklearn contains?

As the picture shows, this package contains all the well-used tools in a complete data mining or machine learning process.



1.2. Installation

```
pip3 install -U scikit-learn
```

In a conda environment, sklearn is pre-installed.

2. Procedures for a classification task: an example

In this section, we use a classification problem as an example to reveal a complete machine learning process by using `sklearn`. Several very basic functions are introduced. We won't include too many details for a specific function (parameters, structures, etc.), and if you are interested, please check the relevant documents.

2.1. Before the training (Data)

2.1.1. Collect data

`sklearn` contains many built-in datasets and tools to load data. It is convenient to use the functions in `sklearn.datasets`.

Here we take iris dataset as an example.

```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target
```

For the data we collected or created by ourself, we need to make sure it satisfies the requirement of dimensionality. For normal data vectors, `X` should have a dimension of (m,n), and `Y` should have a dimension of (m,), where m is the number of data instances, and n is the number of features.

2.1.2. Data preprocessing

The dataset usually needs to be processed before training, including but not limited to:

- Standardization/Normalization
- Missing value completion
- Features processing / Dimension selection

For this iris dataset, we need to normalize it. There is corresponding module in the sklearn.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = scaler.fit_transform(X)

# For the new data instance, we still use this scalar to normalize.
New_data_standard = scaler.fit_transform(New_data)
```

2.1.3. Construct training set and test set

Some benchmark datasets provide their train/test splitting. However, if the data is not split yet, we need to manually divide it.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=2)
```

2.1.4. Data visualization

Sometimes we need to visualize the data to check its characterization. Normally we need to transfer the data to a 2-dim or 3-dim vector. This could be achieved by feature selection or feature reduction.

```
X_2d = X[:, :2]  # Choose two dims

# Use PCA to reduce the dimensions.
pca = PCA(n_components=2)
pca.fit(X)
X_2d = pca.transform(X)
```

Then the data points could be plotted. Here, we need to use the package `matplotlib`, please check this document.

2.2. In the training (Model)

2.2.1. Choose model to use

This article only discuss a classification problem. This is a supervise learning setting.

sklearn contains many built-in models, eg. LinearRegression, SVC, LinearSVC, DecisionTreeClassifier, etc.

Here we took SVC as an example.

```
from sklearn.svm import SVC

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

2.2.2. Model parameter tuning and training

Different models use different hyper-parameters. Those parameters influence the performance of the models very much. Normally, we use cross validation to tune the hyperparameter and train the models.

Here we introduce a built-in parameter selection module. We still use SVC as an example. We search for two hyper-parameter C and Gamma (for RBF kernel).

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

C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
param_grid = dict(gamma=gamma_range, C=C_range)

cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
clf = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
clf.fit(X, y)

print("The best parameters are %s with a score of %0.2f" % (grid.best_params_,
grid.best_score_))
```

2.3. After the training (Performance)

2.3.1. Evaluate the performance

After the training, we need to evaluate the performance on the test set. Common used evaluation methods: accuracy/recall/precision/F1/ROC/AUC etc.

For a trained model, we could use score function to evaluate. It would return an accuracy score by default.

```
score = clf.score(X_test, y_test)
```

sklearn also provides classification_report, which contains more information.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

2.3.2. Visualize the performance

Sometimes we also need to visualize to see the performance. Here we usually add the label margins to the initial visualization. Please check the document of `matplotlib` for more information.