Scikit-learn

Machine Learning

- Learning: using experience to improve performance.
- Machine learning: a class of algorithms that uses data (experience) to improve performance on a task

Kinds of Tasks

- Classification: identify the correct label for an instance
  - Is this a picture of a dog?
  - Which radio emitted the signal we received?
  - Will this customer respond to this advertisement?
- Clustering: identify the groups into which instances fall
  - What are the discernible groups of . . . customers, cars, colors in an images
- Agent behavior
  - Given the state, which action should the agent take to maximize its goal attainment?

Categories of Machine Learning Algorithms

- Supervised
  - Learn from a training set of labeled data – the supervisor
  - Generalize to unseen instances
- Unsupervised
  - Learn from a set of unlabeled data
  - Place an unseen instance into appropriate group
  - Infer rules describing the groups
- Reinforcement learning
  - Learn from a history of trial-and-error exploration
  - Output is a *policy* – a mapping from states to actions (or probabolity distributions over actions)

Classification using supervised learning methods makes up the lion's share of machine learning.

Scikit-learn

```
$ conda install scikit-learn
```

```
>>> import sklearn
```

Scikit-learn Data Representation

The basic supervised learning setup in Scikit-learn is:

- Feature Matrix
  - Rows are instances
  - Columns are features
- Target array
  - An array of len(rows) containing the training labels for each instance

We can easily obtain these with a Pandas DataFrame.

Scikit-learn Recipe

1. Set up feature matrix and target array
2. Choose (import) model class
3. Set model parameters via arguments to model constructor
4. Fit model to data
5. Apply model to new data

Let's apply this recipe to a data set.

The Iris Data Set

It's a rite of passage to apply supervised learning to the Iris data set. The canonical source for the Iris data set is the UCI Machine Learning Repository. Download iris.data. The data set contains 150 instance of Iris flowers with

- 4 features:
  - sepal_length
  - sepal_width
  - petal_length
  - petal_width

and

- 3 classes:
  - Iris-setosa
  - Iris-versicolour
  - Iris-virginica

Let's apply the Scikit-learn recipe.

Step 1: Iris feature matix and target array

From the description on the Iris Data Set page we know that the Iris instances have four features – (sepal_length, sepal_width, petal_length, petal_width) – and three classes – (Iris-setosa, Iris-versicolour, Iris-virginica). We can read these into a DataFrame with

```
iris = pd.read_csv("iris.data", names=["sepal_length",
                                   "sepal_width",
                                   "petal_length",
                                   "petal_width",
                                   "species"])
```

For Scikit-learn we need a feature matrix `X` and target array `y`:

```
X_iris = iris.drop("species", axis=1)
y_iris = iris["species"]
```

We can check that the number of samples in the feature matrix equals the number of labels in the target array with

```
X_iris.shape[0] == y_iris.shape[0] # True
```

There are 150 samples and 150 target labels.

Step 2: Choose a model

In your machine learning class you'll learn that no hypothesis class (aka model class, aka algorithm, aka estimator) is best for all data [1]. You must choose your model class based on the data. Things to consider:

- What's the dimensinalty of your data?
- Are your features linearly separable?
- Are your features numeric or categorical?

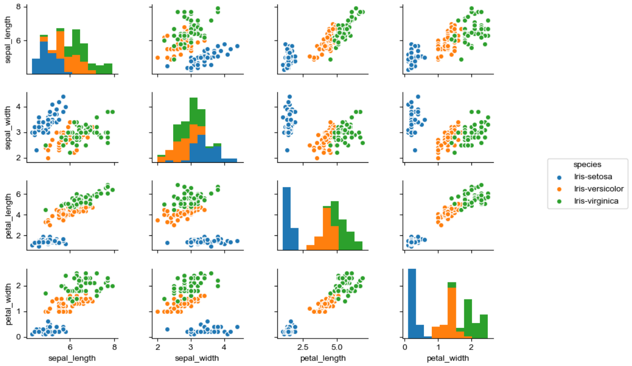
Scikit-learn calls models estimators.

[1] Wolpert and Macready, *No Free Lunch Theorems for Optimization*

Step 2: Visualizing the Iris data

You can begin to explore your data with a pairplot:

```
import seaborn as sns
sns.pairplot(iris, hue="species", size=1.5)
```



These look linearly separable, so we'll use a linear discriminant classifier, SVM.

Step 3: Set model parameters

```
from sklearn import svm
model = svm.SVC(kernel="linear")
```

Most parameters are optional, with reasonable default values. Beacuse we know the Iris data set is so well-suited to liner classifiers we choose a `linear` kernel (deafult is `rbf` – radial basis function)

Step 4: Fit model to data

We want to separate our data into non-overlapping training and test subsets. Since the data in our data set are arranged in a neat order, we should randomize the samples and split in a way that represents each class equally in the training and test sets. Scikit-learn provides a library functoin to do this:

```
from sklearn.model_selection import train_test_split
X_iris_train, X_iris_test, y_iris_train, y_iris_test = train_test_split(X_iris,
                                                                        y_iris,
                                                                        random_state=1)
```

Now we can train our classifier on the training data (fit the model to the training data).

```
model.fit(X_iris_train, y_iris_train)
```

Step 5: Apply model to new data

To apply the trained model to new (unseen) data, pass an array of instances to `predict`:

```
y_iris_model = model.predict(X_iris_test)
```

We can test the generalization error (how well the classifier performs on unseen data) using the built-in accuracy score:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_iris_test, y_iris_model)
1.0
```

As you can see, a linear SVM classifier works perfectly on the Iris data. Try out different classifiers to see how well they perform.
Remember, a Scikit-learn estimator is an object that has `fit` and `predict` methods.