ML_SciKit-Learn

September 2, 2020

2019DMB02

Short note on getting started with Machine Learning using Python.

Machine learning is a means if building models of data. It involves building mathematical models to help understand the data. **'Learning'** happens when we provide tunable parameters that are adapted to the observed data - this is fundamentally known as learning from the data. Once the models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data.

1. **Supervised Learning** - This involves modeling the relationship between measured features of data and some label associated with the data. Once the model is determined, labels can be applied to new data. Classification has discrete labels whereas regression has continuous labels.

2. **Unsupervised Learning** - This involves modeling the features of a dataset without any reference label. Used to identify groupings amongst the data.

When it comes to representing data, we usually do that in terms of tables, where each row represents an observation and each column represents the measurable quantities. Rows are generally referred to as **samples** whereas columns are generally referred to as **features**. Here on note that the number of rows will be referred to as **n_samples** and number of columns will be referred to as **n_features**

[3]:

```python
import seaborn as sns
import pandas as pd
```

[2]:

```python
iris = sns.load_dataset('iris')
iris.head()
```

[2]:

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

From the above table we can gather that information or data can be represented as a 2 dimensional numerical array or **matrix** and is often called the **feature matrix**. The shape of the features matrix is traditionally given by **(n_samples, n_features)**. Feature matrix is generally denoted as **X**.

Now apart from the feature matrix **X** we also look at a target array of labels called **y**. This target array is usually a one dimensional array or a vector. The target array is the quantity that we want to predict from the data.

Before getting into modeling, we will first extract the features matrix and target vector from our data given above.

NOTE: Data is represented using numpy and pandas objects. Therefore to get a proper grasp of various transformations and manipulations in the data like adding/dropping columns and such, refer to pandas documentations.

[4]:
```
X_iris = iris.drop('species', axis=1)
X_iris.shape
```

[4]:
```
(150, 4)
```

[5]:
```
y_iris = iris['species']
y_iris.shape
```

[5]:
```
(150,)
```

Since we will be primarily using the **SciKit-Learn API** of Python to fits models on our data, we must understand the basic process behind designing a model.

1. Since each machine learning algorithm is implemented using the **Estimator** class within the API, we first choose our model by importing the appropriate estimator class from the SciKit Learn.
2. Choose the model hyperparameters by instantiating the class with desired values.
3. Arrange the data into features matrix and target vector.
4. Fit the model by calling the **fit()** method of the model instance.
5. Apply the model to new data by using the **transform** or **predict** methods.

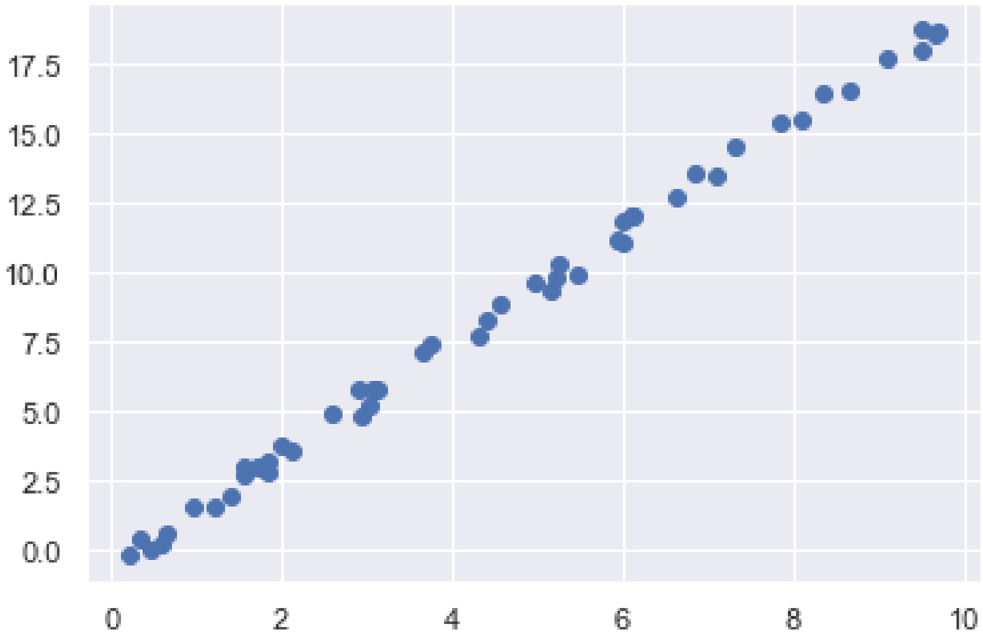
Below we will look at fitting a **simple linear regression** model. First we create a random dataset by using a random number generator.

[6]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[8]:
```
%matplotlib inline
sns.set()
```

[9]:
```
rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.rand(50)
plt.scatter(x, y)
```

[9]:
```
<matplotlib.collections.PathCollection at 0x1a2380e650>
```



Now first, import the **linear regression** model class.

```
[10]: from sklearn.linear_model import LinearRegression
```

Now we basically select **hyperparameters** (which basically lay down the number of model components or parameters-to-be-estimated) by first instantiating the model class and specifying that we would infact like to fit the intercept using the **fit_intercept** hyperparameter.

```
[13]: model = LinearRegression(fit_intercept=True)
      model
```

```
[13]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Now we will arrange the data into features matrix and target vector.

```
[15]: X = x[:, np.newaxis]
      X.shape
```

```
[15]: (50, 1)
```

Now we can fit the model.

```
[20]: model.fit(X, y)
```

```
[20]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
[22]: model.coef_
```

```
[22]: array([2.00660766])
```

```
[23]: model.intercept_
```

```
[23]: -0.5350275750800026
```

The above two parameters represent the slope and intercept of the linear regression fit. Now before we move on to predicting on a new dataset, we can first create the new dataset as follows.

```
[24]: xfit = np.linspace(-1, 11)
```

```
[25]: Xfit = xfit[:, np.newaxis]
      Xfit.shape
```

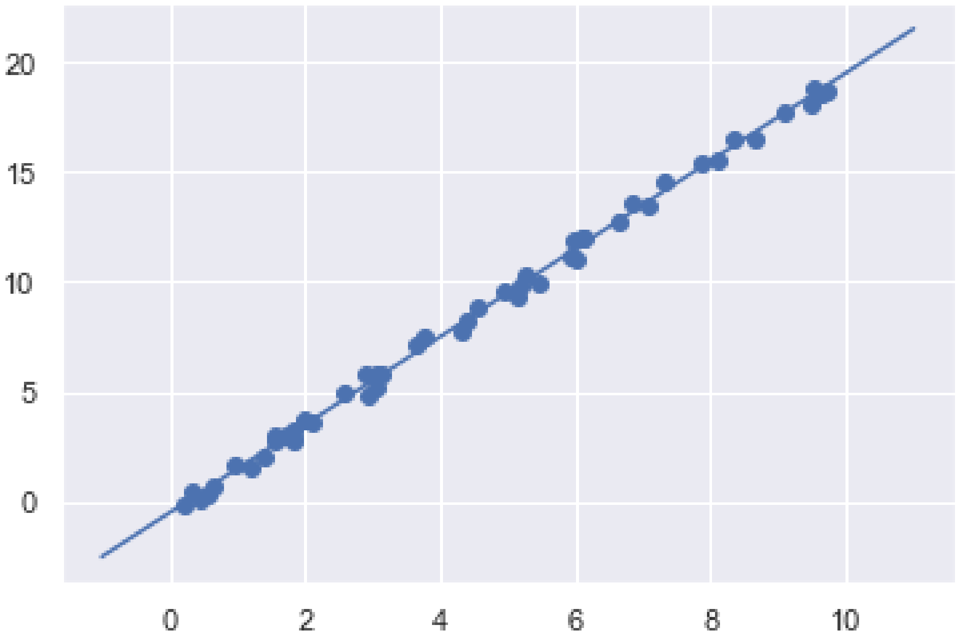
```
[25]: (50, 1)
```

```
[26]: yfit = model.predict(Xfit)
```

Finally, we will visualize the plots of the original data and then the model fit.

```
[27]: plt.scatter(x, y)
      plt.plot(xfit, yfit)
```

```
[27]: [<matplotlib.lines.Line2D at 0x1a24510410>]
```



Now we look at supervised learning from a **classification** standpoint with **Naive Bayes**. This is a generative algorithm that assumes that each class comes from a **gaussian distribution**. Before fitting the model we split the data into a training and testing set.

[30]:
```python
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X_iris, y_iris, random_state=1)
```

[31]:
```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, Ytrain)
y_model = model.predict(Xtest)
```

[32]:
```python
from sklearn.metrics import accuracy_score
```

[33]:
```python
accuracy_score(Ytest, y_model)
```

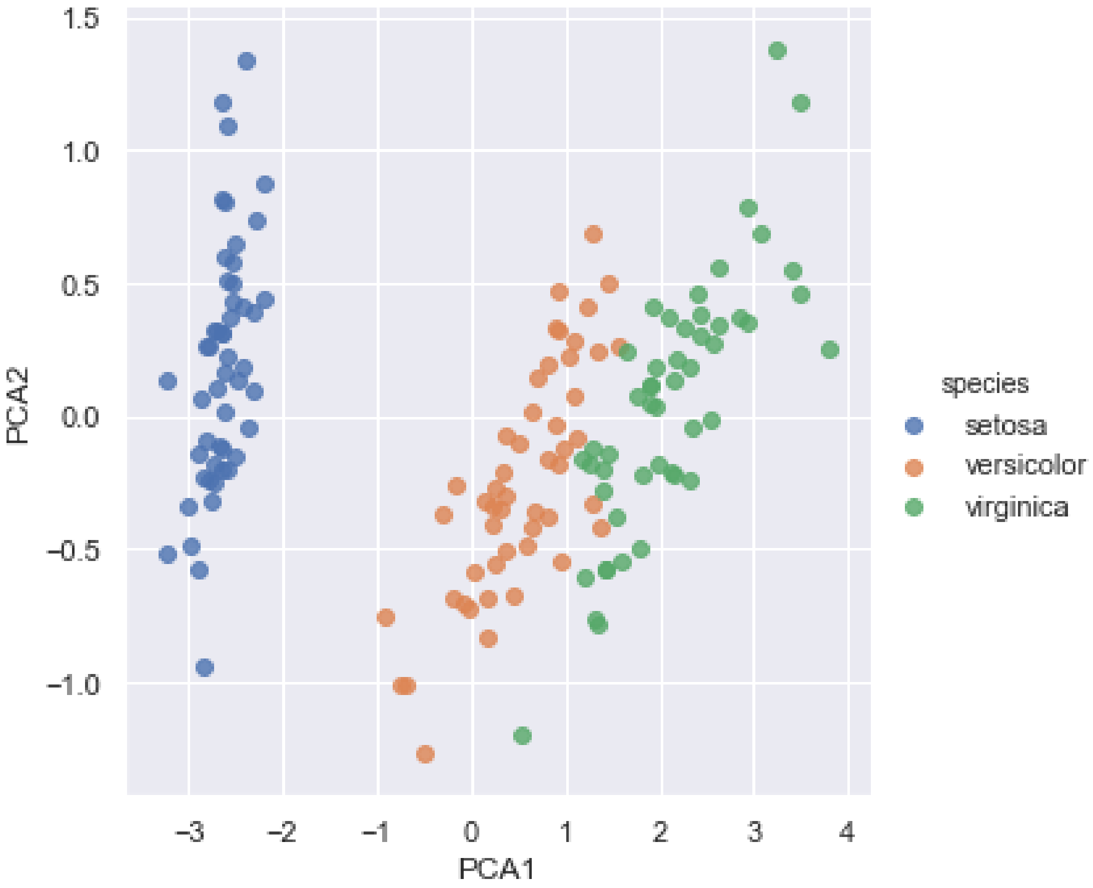
[33]:
```
0.9736842105263158
```

Now we will look at the task of **dimensionality reduction** - which basically involves getting a suitable lower dimensional representation of the data such that essential features are retained. This technique helps us visualize data much better.

[34]:
```python
from sklearn.decomposition import PCA
model = PCA(n_components=2)
model.fit(X_iris)
X_2D = model.transform(X_iris)
```

[36]:
```python
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False)
```

[36]:
```
<seaborn.axisgrid.FacetGrid at 0x1a24cd1c10>
```


1.5
1.0
0.5
0.0
−0.5
−1.0
PCA2
−3
−2
−1
0
1
2
3
4
PCA1
species
● setosa
● versicolor
● virginica