Lecture 3: Introduction to Scikit-Learn

Scikit-Learn

[Scikit-Learn (http://scikit-learn.org/stable/)](http://scikit-learn.org/stable/) is an extremely popular python machine learning package.

Provides implementations of a number of different machine learning algorithms.

Scikit-Learn

[Scikit-Learn (http://scikit-learn.org/stable/)](http://scikit-learn.org/stable/) is an extremely popular python machine learning package.

Provides implementations of a number of different machine learning algorithms.

- Clean, uniform and streamlined API.
- Useful and complete online documentation.
- Straightforward to switch models or algorithms.

Two main general concepts:

- Data representation
- Estimator API

Data representations

Scikit-Learn includes a number of example data-sets

In [2]:

```
from sklearn import datasets
```

In [3]:

```
# Type datasets.<TAB> to see more
#datasets.
```

Data as a table

Best way to think about data in Scikit-Learn is in terms of tables of data.

Using the `seaborn` (http://seaborn.pydata.org/) library we can read example data-sets as a Pandas `DataFrame`.

In [4]:

```
import seaborn as sns
iris = sns.load_dataset('iris')
type(iris)
```

Out[4]:

```
pandas.core.frame.DataFrame
```

In [5]:

```
iris.head()
```

Out[5]:

	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

Iris data

Here we consider the Iris flower data (https://en.wikipedia.org/wiki/Iris_flower_data_set).

- Introduced by statistician and biologist Ronald Fisher in 1936 paper.
- Consists of 50 samples of three different species of Iris (Iris Setosa, Iris Virginica and Iris Versicolor).
- Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres.

Iris data

Here we consider the [Iris flower data (https://en.wikipedia.org/wiki/Iris_flower_data_set)](https://en.wikipedia.org/wiki/Iris_flower_data_set).

- Introduced by statistician and biologist Ronald Fisher in 1936 paper.
- Consists of 50 samples of three different species of Iris (Iris Setosa, Iris Virginica and Iris Versicolor).
- Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres.

In [6]:

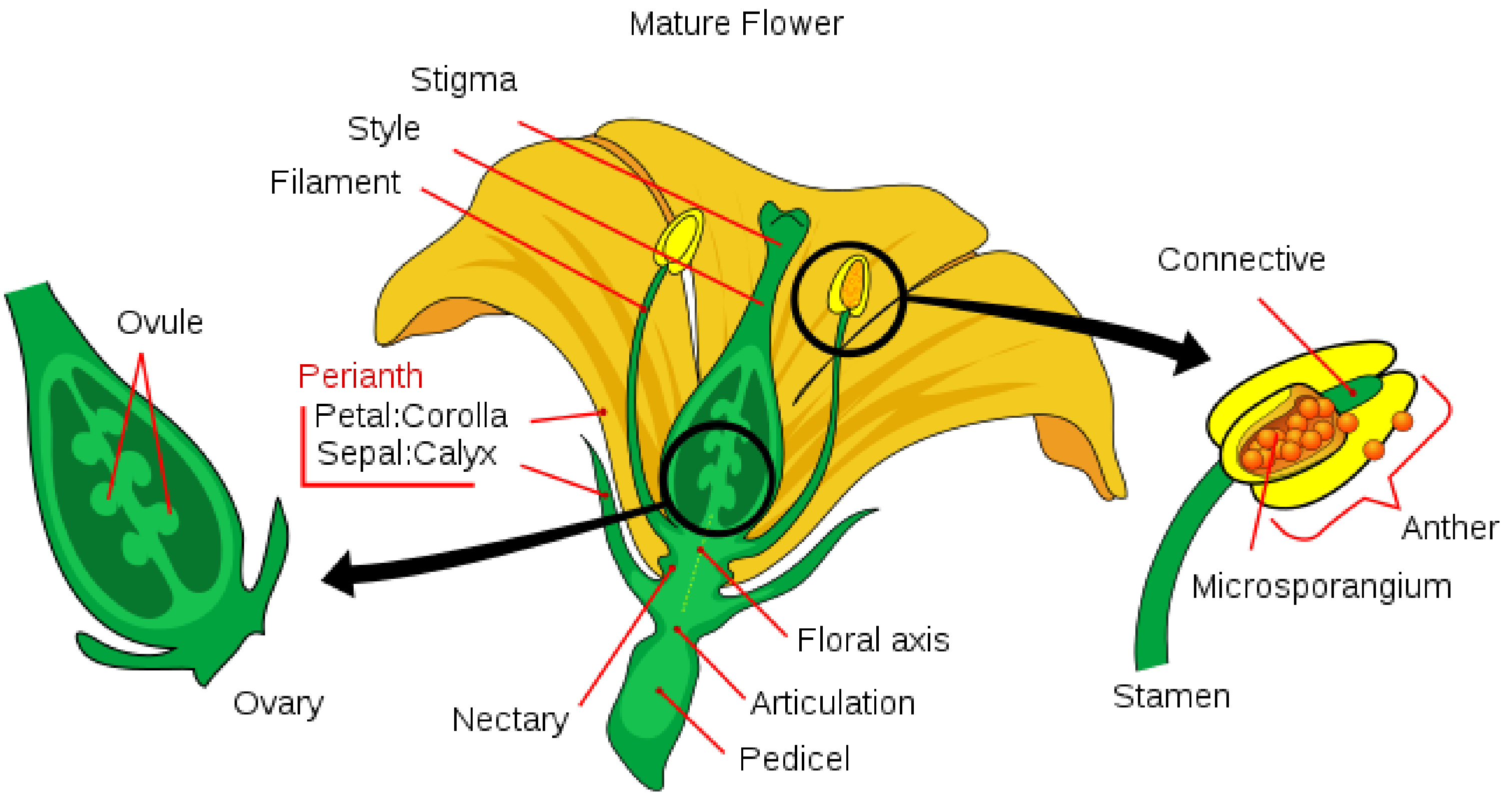
```
iris.tail()
```

Out[6]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Parts of a flower

Measured flower [petals (https://en.wikipedia.org/wiki/Petal)](https://en.wikipedia.org/wiki/Petal) and [sepals (https://en.wikipedia.org/wiki/Sepal)](https://en.wikipedia.org/wiki/Sepal).



[Image credit: [Mariana Ruiz (https://en.wikipedia.org/wiki/Sepal#/media/File:Mature_flower_diagram.svg)](https://en.wikipedia.org/wiki/Sepal#/media/File:Mature_flower_diagram.svg)]

Images of different species

		
Iris Setosa	Iris Versicolor	Iris Virginica

[Image source (https://github.com/jakevdp/sklearn_tutorial)]

Features matrix

Recall data represented to learning algorithm as "*features*".

Each row corresponds to an observed (*sampled*) flower, with a number of *features*.

In [7]:
```
iris.head()
```

Out[7]:

	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

In this example we extract a feature matrix, removing species (which we want to predict).

In [8]:
```
iris.head()
```

Out[8]:

	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

In [9]:

```
X_iris = iris.drop('species', axis='columns')
X_iris.head()
```

Out[9]:

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

In [10]:

```
type(X_iris)
```

Out[10]:

```
pandas.core.frame.DataFrame
```

Target array

Consider 1D *target array* containing labels or targets that we want to predict.

May be numerical values or discrete classes/labels.

In this example we want to predict the flower species from other measurements.

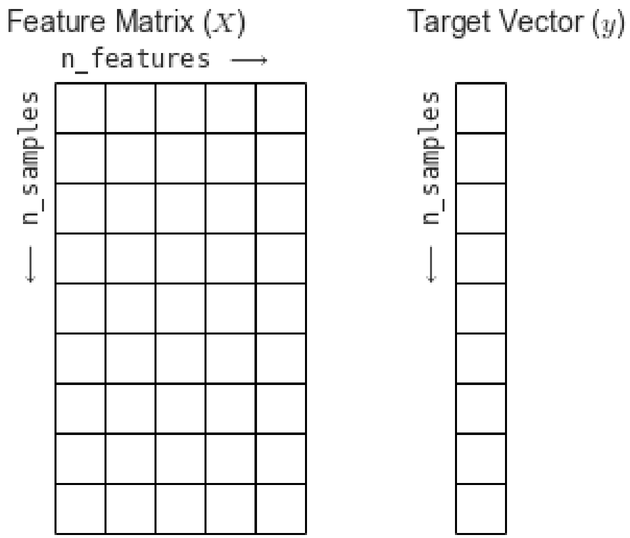
In [11]:
```
y_iris = iris['species']
y_iris.head()
```

Out[11]:
```
0    setosa
1    setosa
2    setosa
3    setosa
4    setosa
Name: species, dtype: object
```

In [12]:
```
type(y_iris)
```

Out[12]:
```
pandas.core.series.Series
```

Features matrix and target vector



[Image source (https://github.com/jakevdp/sklearn_tutorial)]

In [13]:
```
X_iris.shape
```

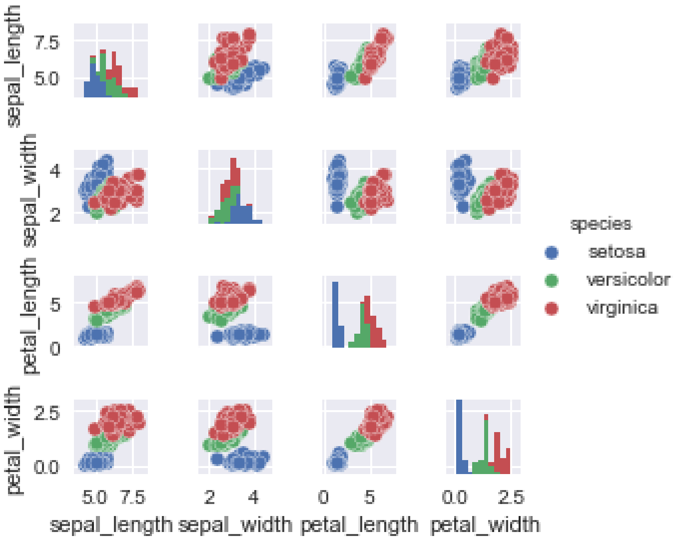
Out[13]:
```
(150, 4)
```

In [14]:
```
y_iris.shape
```

Out[14]:
```
(150,)
```

Visualizing the data

In [15]:
```
%matplotlib inline
import seaborn as sns; sns.set()
sns.pairplot(iris, hue='species', size=1.0);
```



Exercise: How well do you expect classification to perform with these features and why?

Exercise: How well do you expect classification to perform with these features and why?

Fairly well since the different classes are reasonably well separated in feature space.

Scikit-Learn’s Estimator API

Scikit-Learn API design principles

- Consistency: All objects share a common interface.
- Inspection: All specified parameter values exposed as public attributes.
- Limited object hierarchy: Only algorithms are represented by Python classes; data-sets/parameters represented in standard formats.
- Composition: Many machine learning tasks can be expressed as sequences of more fundamental algorithms.
- Sensible defaults: Library defines appropriate default value.

Impact of design principles

- Makes Scikit-Learn easy to use, once the basic principles are understood.
- Every machine learning algorithm in Scikit-Learn implemented via the Estimator API.
- Provides a consistent interface for a wide range of machine learning applications.

Typical Scikit-Learn Estimator API steps

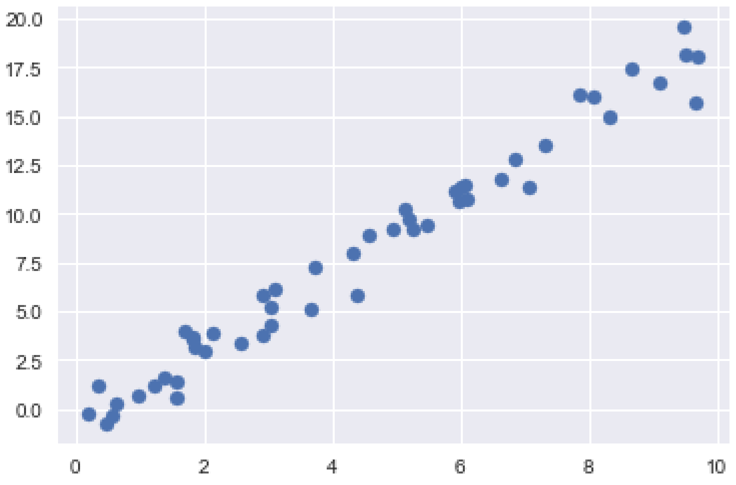
1. Choose a class of model (import appropriate estimator class).
2. Choose model hyperparameters (instantiate class with desired values).
3. Arrange data into a features matrix and target vector.
4. Fit the model to data (calling `fit` method of model instance).
5. Apply model to new data:
    - Supervised learning: often predict targets for unknown data using the `predict` method.
    - For unsupervised learning: often transform or infer properties of the data using the `transform` or `predict` method.

Linear regression as machine learning

In [16]:

```
import matplotlib.pyplot as plt
import numpy as np

n_samples = 50
rng = np.random.RandomState(42)
x = 10 * rng.rand(n_samples)
y = 2 * x - 1 + rng.randn(n_samples)
plt.scatter(x, y);
```



1. Choose a class of model

Every class of model is represented by a Python class.

In [17]:
```
from sklearn.linear_model import LinearRegression
```

2. Choose model hyperparameters

Make instance of model with defined hyperparameters (e.g. y-intersect, regularization).

In [18]:
```
model = LinearRegression(fit_intercept=True)
model
```

Out[18]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

3. Arrange data into a features matrix and target vector

In [19]:
```
X = x.reshape(n_samples,1)
X.shape
```

Out[19]:
```
(50, 1)
```

In [20]:
```
y.shape
```

Out[20]:
```
(50,)
```

4. Fit the model to data

In [21]:
```
model.fit(X, y)
```

Out[21]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

4. Fit the model to data

In [21]:
```
model.fit(X, y)
```

Out[21]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

All model parameters that were learned during the `fit()` process have *trailing underscores*.

In [22]:
```
model.intercept_
```

Out[22]:
```
-0.9033107255311164
```

In [23]:
```
model.coef_
```

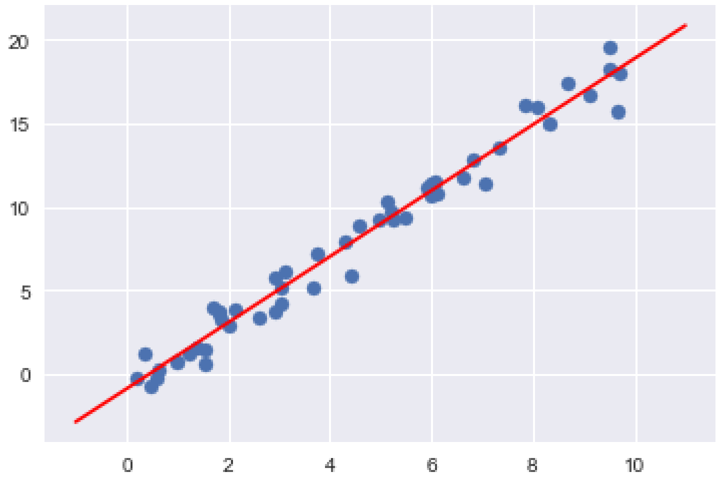
Out[23]:
```
array([1.9776566])
```

Intercept and slope are close to the model used to generate the data (-1 and 2 respectively).

5. Predict targets for unknown data

In [24]:
```
n_fit = 50
xfit = np.linspace(-1, 11, n_fit)
Xfit = xfit.reshape(n_fit,1)
yfit = model.predict(Xfit)
```

In [25]:
```
plt.scatter(x, y)
plt.plot(xfit, yfit, 'r');
```



Supervised learning example: classification

Consider Iris data-set again and predict species.

Exercise: set up data

Split data into training and test sets (hint: `train_test_split` (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) is a convenient scikit-learn function for this task).

Exercise: set up data

Split data into training and test sets (hint: [`train_test_split` (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) is a convenient scikit-learn function for this task).

In [26]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.5,
 random_state=1)
```

In [27]:
```
X_train.head()
```

Out[27]:

	sepal_length	sepal_width	petal_length	petal_width
74	6.4	2.9	4.3	1.3
116	6.5	3.0	5.5	1.8
93	5.0	2.3	3.3	1.0
100	6.3	3.3	6.0	2.5
89	5.5	2.5	4.0	1.3

Exercise: Classify species

Use a Gaussian Naive Bayes (`GaussianNB`) model to predict Iris species. Then evaluate performance on test data.

(Hint: choose, instantiate, fit and predict.)

See Scikit-Learn documentation on `GaussianNB` (http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html).

Evaluate performance using simple `accuracy_score` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.ac

(Do not set any priors.)

In [28]:

```
from sklearn.naive_bayes import GaussianNB  # 1. choose model class
model = GaussianNB()                        # 2. instantiate model
model.fit(X_train, y_train)                 # 3. fit model to data
y_model = model.predict(X_test)             # 4. predict on new data
```

In [28]:

```
from sklearn.naive_bayes import GaussianNB  # 1. choose model class
model = GaussianNB()                        # 2. instantiate model
model.fit(X_train, y_train)                 # 3. fit model to data
y_model = model.predict(X_test)             # 4. predict on new data
```

Evaluate performance on test data.

In [29]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_model)
```

Out[29]:

```
0.96
```

Unsupervised learning example: dimensionality reduction

Reduce dimensionality of Iris data for visualisation or to discover structure.

Recall the original Iris data has four features.

In [30]:
```
X_iris.head()
```

Out[30]:

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

In [31]:
```
X_iris.shape
```

Out[31]:
```
(150, 4)
```

Exercise: Iris dimensionality reduction

Compute principle component analysis (`PCA`), with 2 components, and apply transform. Plot data in PCA space.

(Hint: choose, instantiate, fit and transform.)

See Scikit-Learn documentation on [`PCA` (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html).

See Seaborn documentation on [`lmplot` (https://seaborn.pydata.org/generated/seaborn.lmplot.html)](https://seaborn.pydata.org/generated/seaborn.lmplot.html).

In [32]:

```
from sklearn.decomposition import PCA  # 1. Choose the model class
model = PCA(n_components=2)            # 2. Instantiate the model with hyperparame
ters
model.fit(X_iris)                      # 3. Fit to data. Notice y is not specifie
d!
X_2D = model.transform(X_iris)         # 4. Transform the data to two dimensions
```

In [33]:

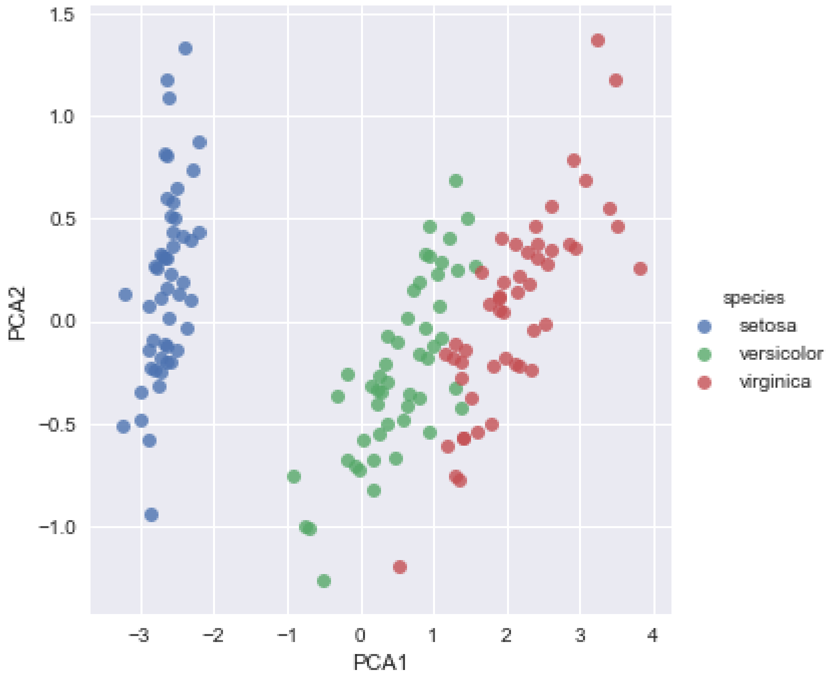
```
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
iris.head()
```

Out[33]:

	sepal_length	sepal_width	petal_length	petal_width	species	PCA1
0	5.1	3.5	1.4	0.2	setosa	-2.684126
1	4.9	3.0	1.4	0.2	setosa	-2.714142
2	4.7	3.2	1.3	0.2	setosa	-2.888991
3	4.6	3.1	1.5	0.2	setosa	-2.745343
4	5.0	3.6	1.4	0.2	setosa	-2.728717

In [34]:

```
sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False);
```



Exercise: How well do you expect classification to perform using PCA components as features and why?

Exercise: How well do you expect classification to perform using PCA components as features and why?

Very well since the different classes are well separated in PCA feature space.

Unsupervised learning example: clustering

Attempt to find "groups" in Iris data without given labels or training data.

Exercise: Cluster Iris data

Cluster Iris data into 3 components using Gaussian Mixture Model (GMM). Plot the 3 components separately in PCA space.

(Hint: choose, instantiate, fit and predict.)

See Scikit-Learn documentation on [`GaussianMixture` (http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html)](http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html).

Exercise: Cluster Iris data

Cluster Iris data into 3 components using Gaussian Mixture Model (GMM). Plot the 3 components separately in PCA space.

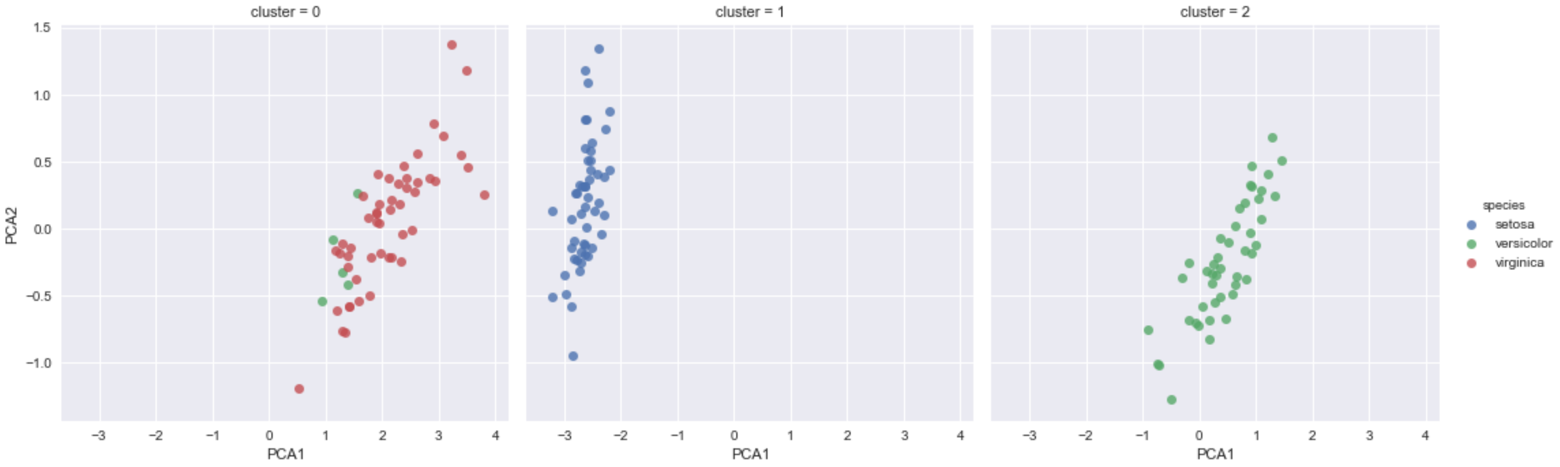
(Hint: choose, instantiate, fit and predict.)

See Scikit-Learn documentation on `GaussianMixture` (http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html).

In [35]:

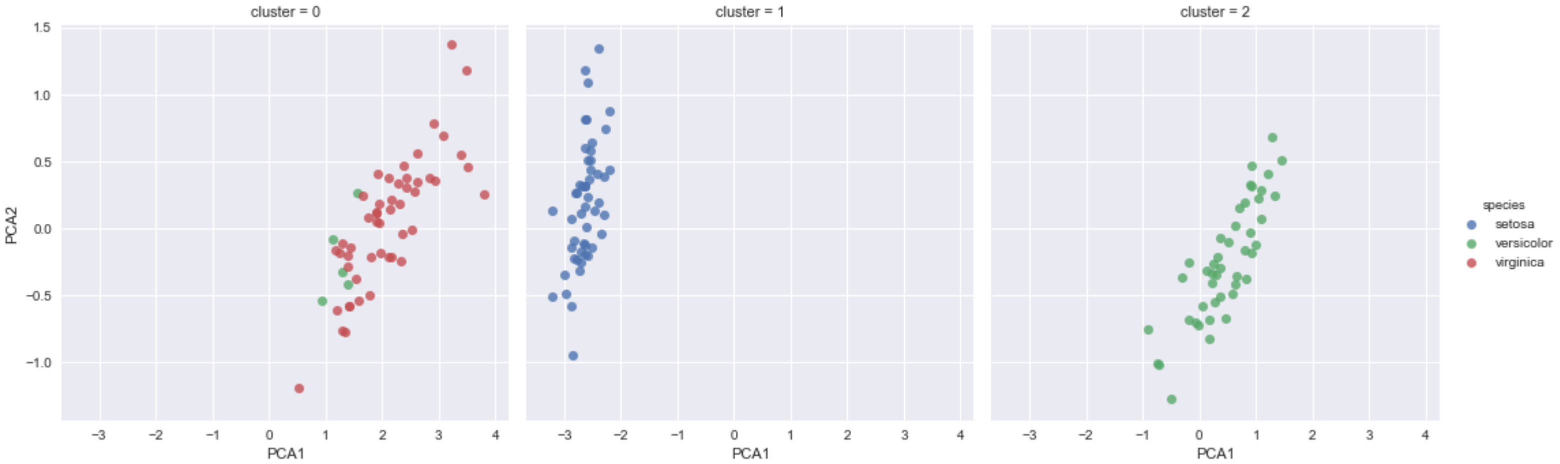
```
from sklearn.mixture import GaussianMixture        # 1. Choose the model class
model = GaussianMixture(n_components=3)            # 2. Instantiate the model with h
yperparameters
model.fit(X_iris)                                  # 3. Fit to data. Notice y is not
 specified!
y_gmm = model.predict(X_iris)                      # 4. Determine cluster labels
```

In [36]:
```
iris['cluster'] = y_gmm
sns.lmplot("PCA1", "PCA2", data=iris, hue='species',
           col='cluster', fit_reg=False);
```



In [36]:

```
iris['cluster'] = y_gmm
sns.lmplot("PCA1", "PCA2", data=iris, hue='species',
           col='cluster', fit_reg=False);
```



The GMM has done a reasonably good job of separating the different classes. Setosa is perfectly separated in one cluster, while there remains some mixing between versicolor and viginica.

Exercise: Classify hand-written digits

Load example Scikit-Learn data.

In [37]:
```
from sklearn.datasets import load_digits
digits = load_digits()
```

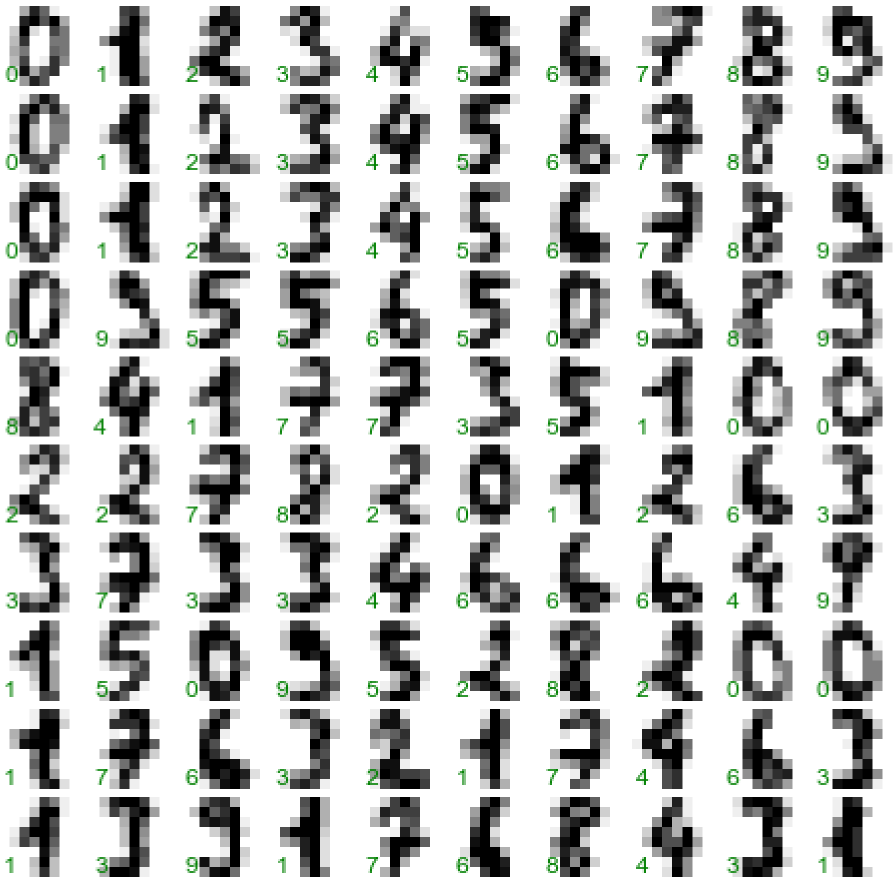
- Explore the data-set and plot some example images.
- Split the data-set into training and test sets.
- Train a logistic regression classifier with an ℓ_2 penalty.
- Compute the accuracy of predictions on the test set.

Plot example images

In [38]:

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```



Set up feature and target data

In [39]:
```
X = digits.data
X.shape
```

Out[39]:
```
(1797, 64)
```

In [40]:
```
y = digits.target
y.shape
```

Out[40]:
```
(1797,)
```

Create training and test sets

In [41]:
```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
```

In [42]:
```
Xtrain.shape, Xtest.shape
```

Out[42]:
```
((1347, 64), (450, 64))
```

Choose model, instantiate, fit and predict

In [43]:

```
from sklearn.linear_model import LogisticRegression # 1. choose model class
model = LogisticRegression(penalty='l2')            # 2. instantiate model
model.fit(Xtrain, ytrain)                           # 3. fit model to data
y_model = model.predict(Xtest)                      # 4. predict on new data
```

Evaluate accuracy on test data

In [44]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

Out[44]:
```
0.9533333333333334
```