# Python for Machine Learning and Data Analysis

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Spring 2019

# Machine Learning 01

Lecture 7

# Content

- Machine Learning
  - What is Machine Learning?
  - Categories of Machine Learning
  - Qualitative Examples of Machine Learning Applications
- Introducing Scikit-Learn
  - Data Representation in Scikit-Learn
  - Estimator API
  - Example

# Machine Learning

- In many ways, machine learning is the primary means by which data science manifests itself to the broader world.
- Machine learning is where the computational and algorithmic skills of data science meet the statistical thinking of data science, and the result is a collection of approaches to inference and data exploration that are not about effective theory so much as effective computation.
- While these methods can be incredibly powerful, to be effective they must be approached with a firm grasp of the strengths and weaknesses of each method, as well as a grasp of general concepts such as bias and variance, overfitting and underfitting, and more.

- This course will dive into practical aspects of machine learning, primarily using Python's Scikit-Learn package.
- This is not meant to be a comprehensive introduction to the field of machine learning; that is a large subject and necessitates a more technical approach than we take here.
- Nor is it meant to be a comprehensive manual for the use of the Scikit-Learn package. You can learn the online manual by yourself.
- Rather, the goals of this course are:
  - **To introduce the fundamental vocabulary and concepts of machine learning.**
  - **To introduce the Scikit-Learn API and show some examples of its use.**
  - **To take a deeper dive into the details of several of the most important machine learning approaches, and develop an intuition into how they work and when and where they are applicable.**

# What is machine learning?

- Machine learning is often categorized as a subfield of artificial intelligence, but that categorization can often be misleading at first.
- The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of *building models of data*.
- Fundamentally, machine learning involves building mathematical models to help understand data.
- "Learning" enters the fray when we give these models *tunable parameters* that can be adapted to observed data; in this way the program can be considered to be "learning" from the data.
- Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data.

# What is machine learning?

- Machine learn… categ… …ce, but that categorization ca…
- The study of… …the data … …nk of machine le…
- Fundamentally … …In understand…
- "Learning" enters the fr… …*eters* that can be adapted to observed d… …pro… …e cons… …o be "learning" from the data.
- Once these models have been fit … …viously seen … …ta, they can be used to predict and understand aspects of newly o… …erved data.

Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

# Categories of Machine Learning

- At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning.
- *Supervised learning* involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into *classification* tasks and *regression* tasks:
  - in classification, the labels are discrete categories, while in regression, the labels are continuous quantities.
- *Unsupervised learning* involves modeling the features of a dataset without reference to any label, and is often described as "letting the dataset speak for itself." These models include tasks such as *clustering* and *dimensionality reduction.*
  - Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data.

# Qualitative Examples of Machine Learning Applications

- Classification: Predicting discrete labels

two-dimensional data:

- two *features* for each point, represented by the *(x,y)* positions of the points on the plane.
- two *class labels* for each point, represented by the colors of the points.

Assumption: that the two groups can be separated by drawing a straight line through the plane between them, such that points on each side of the line fall in the same group.

- The *model* is a quantitative version of the statement "a straight line separates the classes“
- The *model parameters* are the particular numbers describing the location and orientation of that line for our data.

The following figure shows a visual representation of what the trained model looks like for this data:

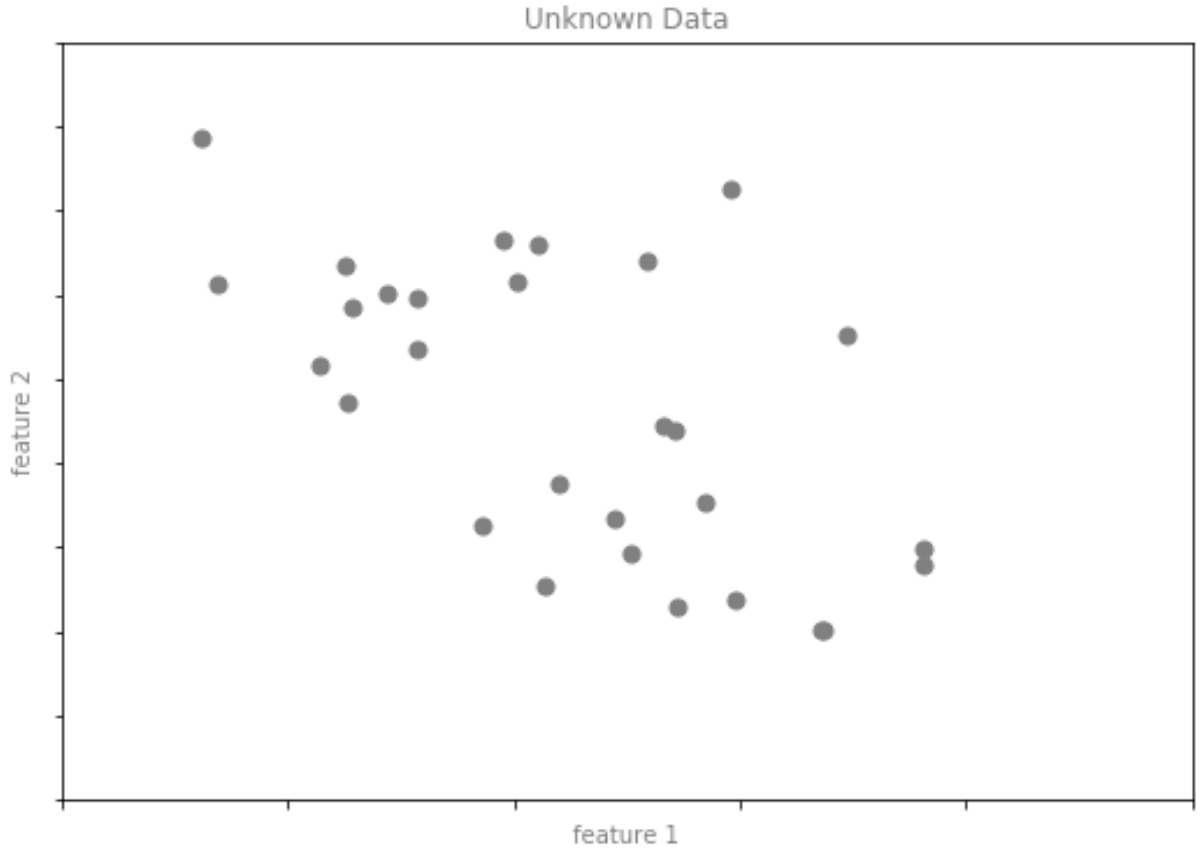

The optimal values for these model parameters are learned from the data (this is the "learning" in machine learning), which is often called ***training the model***.

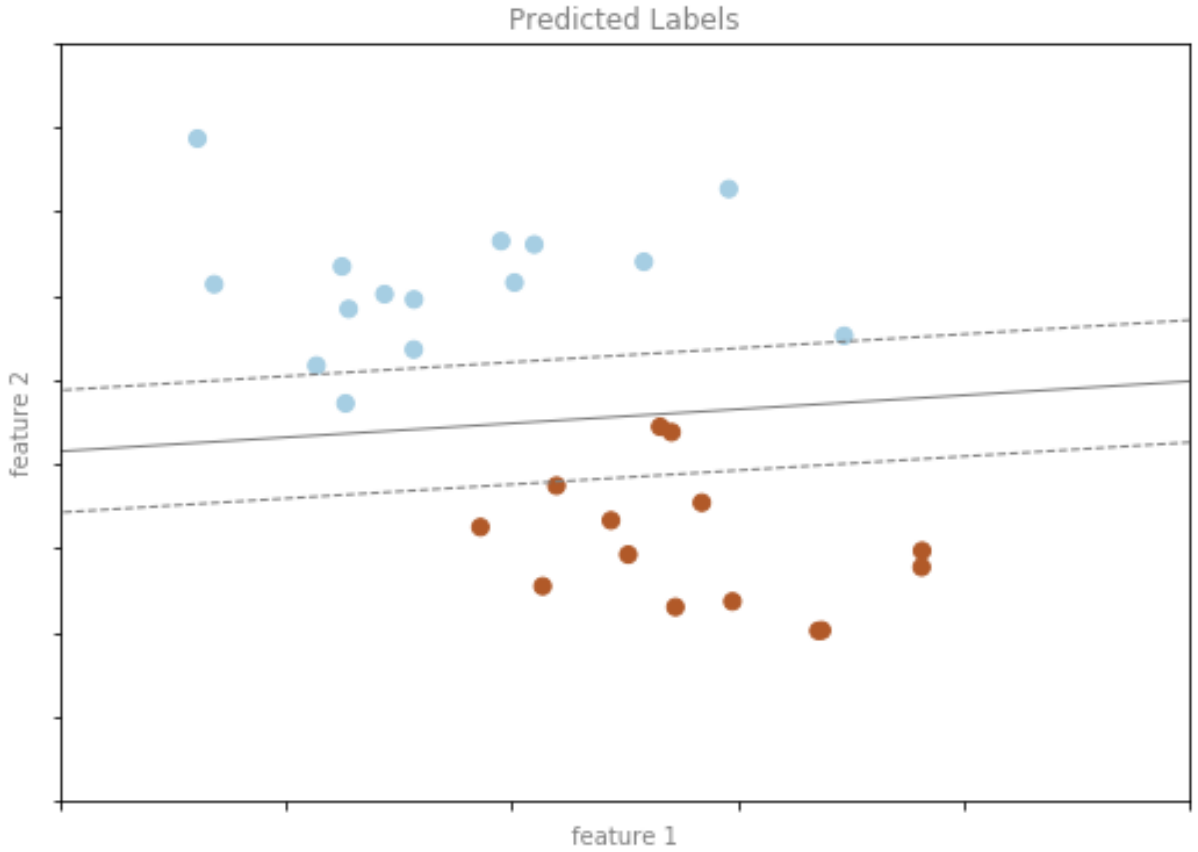
- Now that this model has been trained, it can be generalized to new, unlabeled data.

  - Now, a new set of data is following, we can draw this model line through it, and assign labels to the new points based on this model.

This stage is usually called ***prediction***.

- Then the data points can be labeled "blue" or "red":



- This is the basic idea of a classification task in machine learning, where "classification" indicates that the data has discrete class labels.
- At first glance this may look fairly trivial. A benefit of the machine learning approach, however, is that it can generalize to much larger datasets in many more dimensions.
- For example, this is similar to the task of automated spam detection for email; in this case, we might use the following features and labels:
  - feature 1, feature 2, etc.  normalized counts of important words or phrases (“Congrats", “Gift Card", etc.)
  - label  "spam" or "not spam"
  - For the training set, these labels might be determined by individual inspection of a small representative sample of emails; for the remaining emails, the label would be determined using the model.
  - For a suitably trained classification algorithm with enough well-constructed features (typically thousands or millions of words or phrases), this type of approach can be very effective.
- Some important classification algorithms that we will discuss in more detail in this course are Gaussian naive Bayes, support vector machines, and random forest classification.

# Regression: Predicting continuous labels

In contrast with the discrete labels of a classification algorithm, in a simple *regression* task, the labels are continuous quantities.

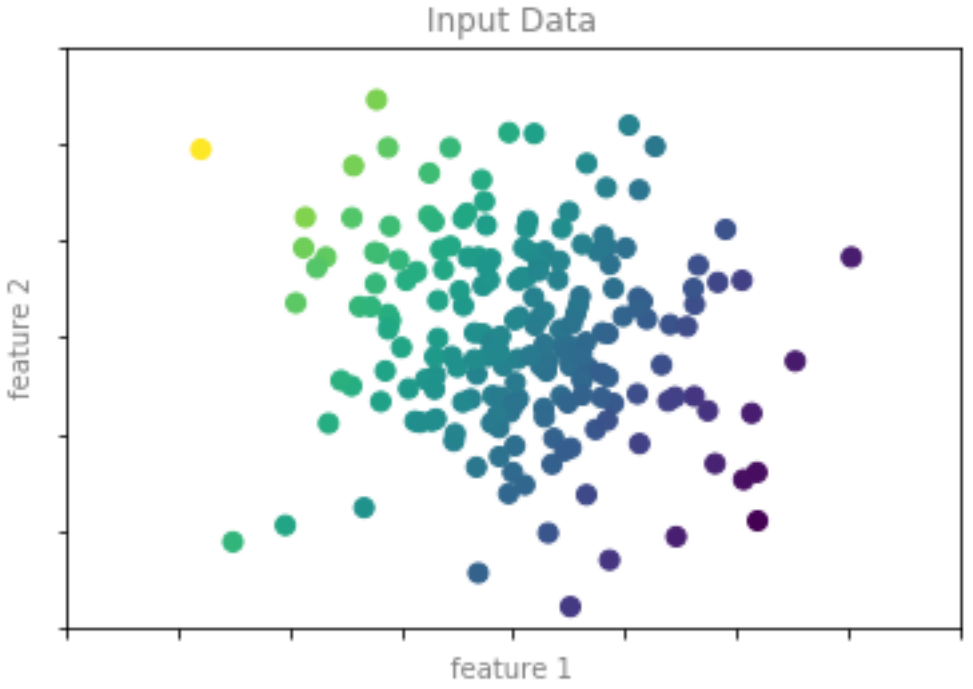
two-dimensional data:

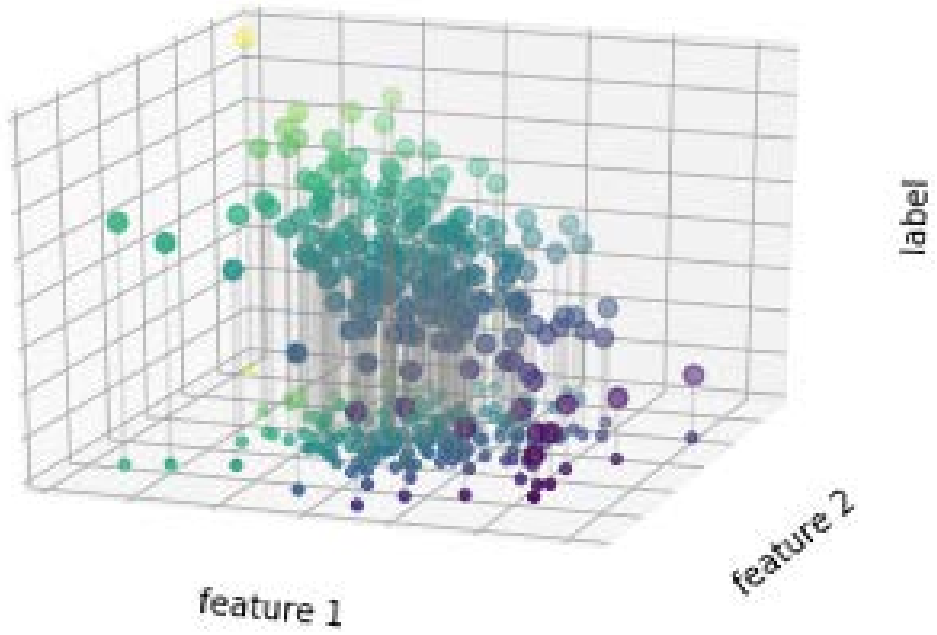
- two features describe each data point.
- color of each point represents the continuous label for that point.

Assume that:

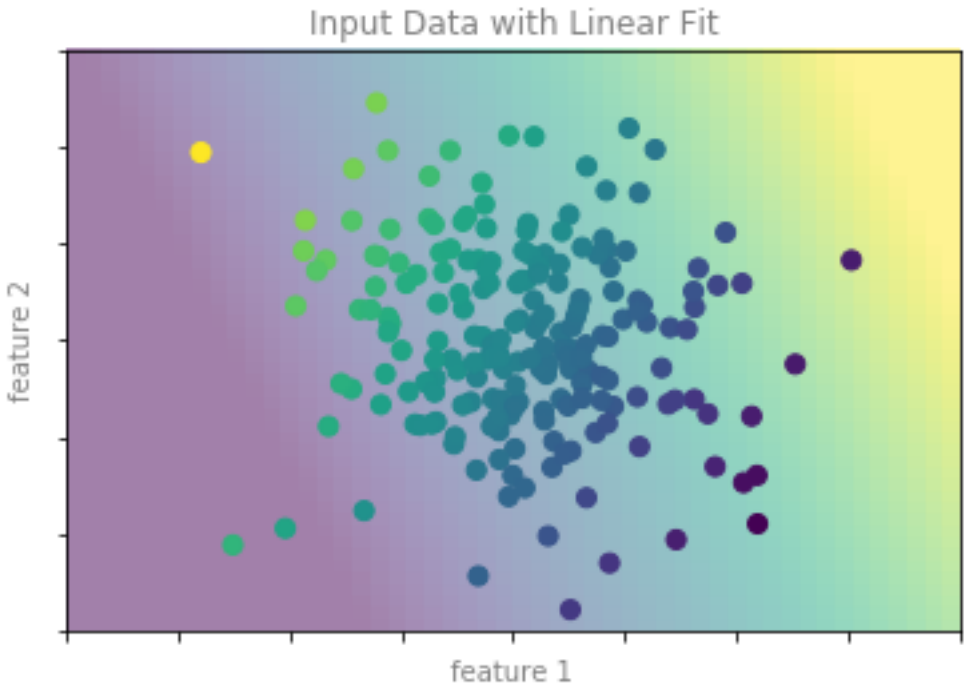
- If the label is treated as a third spatial dimension, we can fit a plane to the data.
- A simple linear regression can be used to predict the points here.
- This is a higher-level generalization of the well-known problem of fitting a line to data with two coordinates.

Consider the data shown in the figure, which consists of a set of points each with a continuous label:






Returning to the two-dimensional projection, when we fit such a plane we get the result shown in the following figure:



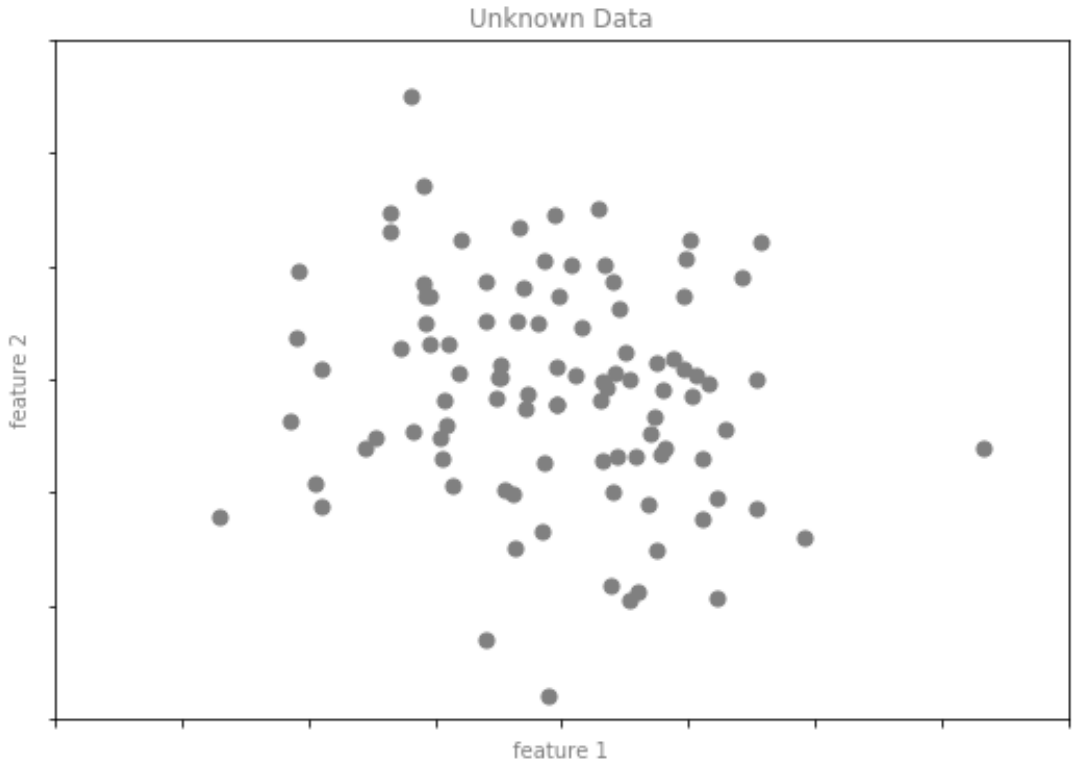

Notice that:

- the *feature 1 - feature 2* plane here is the same as in the two-dimensional plot from before;
- the labels are represented by both color and three-dimensional axis position.

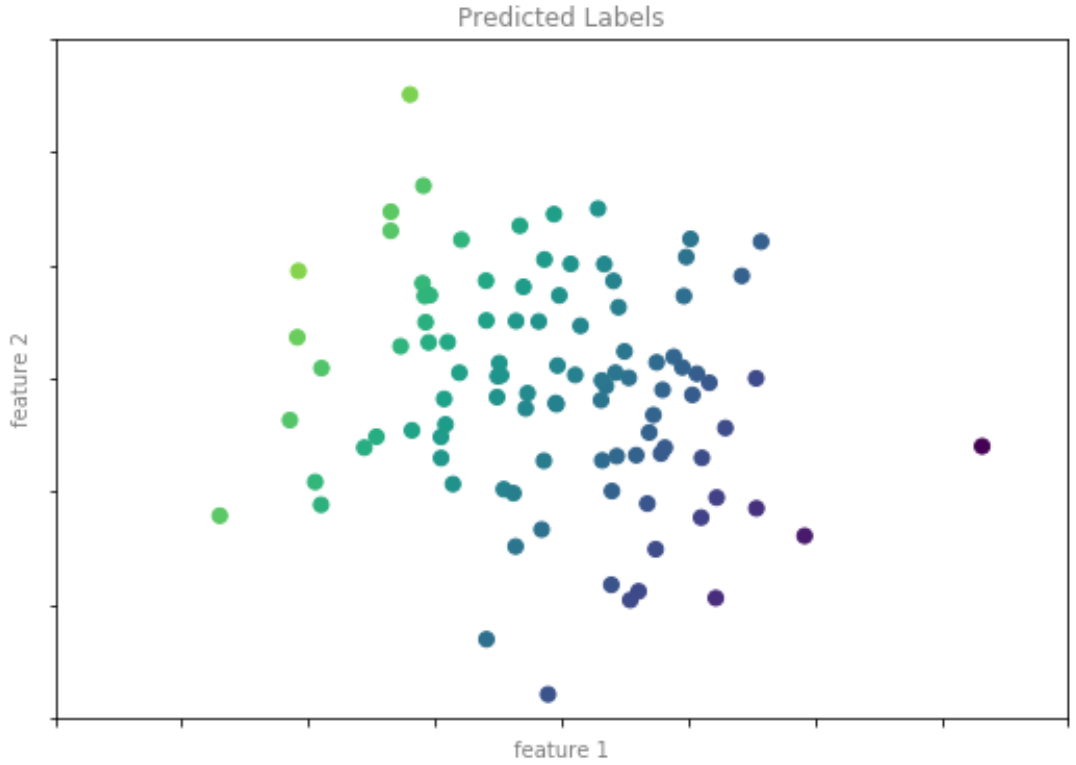
- From this view, it seems reasonable that fitting a plane through this three-dimensional data would allow us to predict the expected label for any set of input parameters.

# This plane of fit gives us what we need to predict labels for new points.

Now, a new set of data is following,



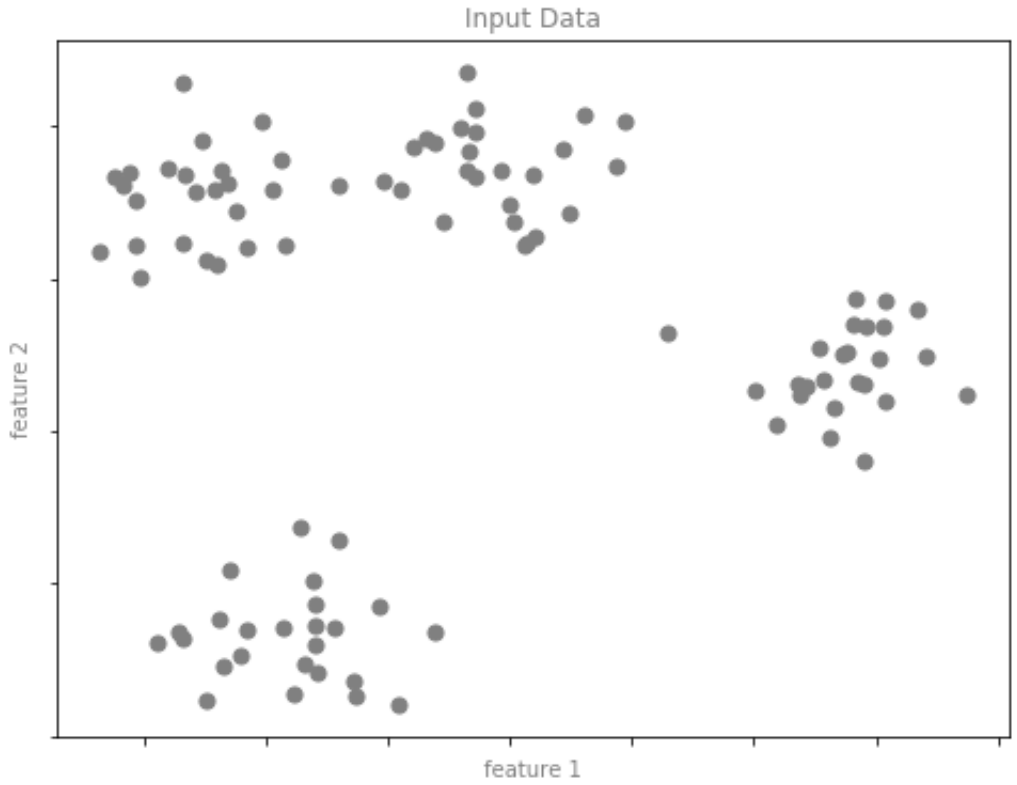

Visually, we find the results shown in the following figure:



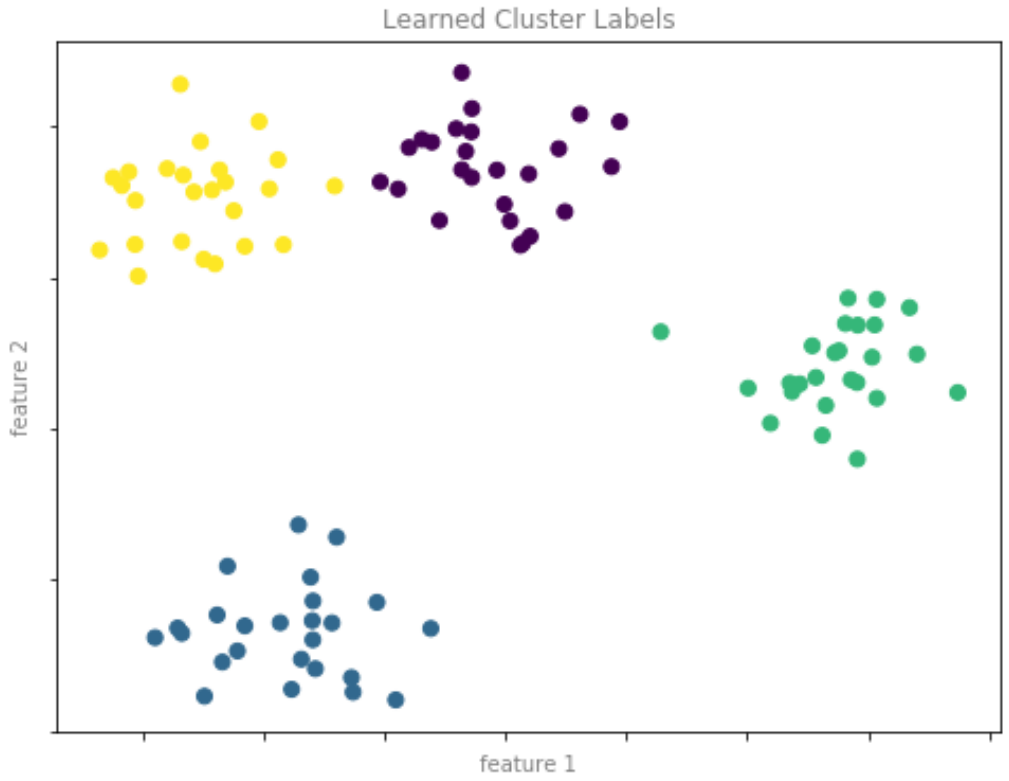

As with the classification example, this may seem rather trivial in a low number of dimensions. But the power of these methods is that they can be straightforwardly applied and evaluated in the case of data with many, many features.

# Clustering: Inferring labels on unlabeled data

- The classification and regression are the supervised learning algorithms, which predict labels for new data.
- Unsupervised learning involves models that describe data without reference to any known labels.
- One common case of unsupervised learning is "clustering," in which data is automatically assigned to some number of discrete groups.
- For example:



- It is clear that each of these points is part of a distinct group.
- Given this input, a clustering model will use the intrinsic structure of the data to determine which points are related.
- Using the very fast and intuitive *k*-means algorithm, we find the clusters shown in the following figure:

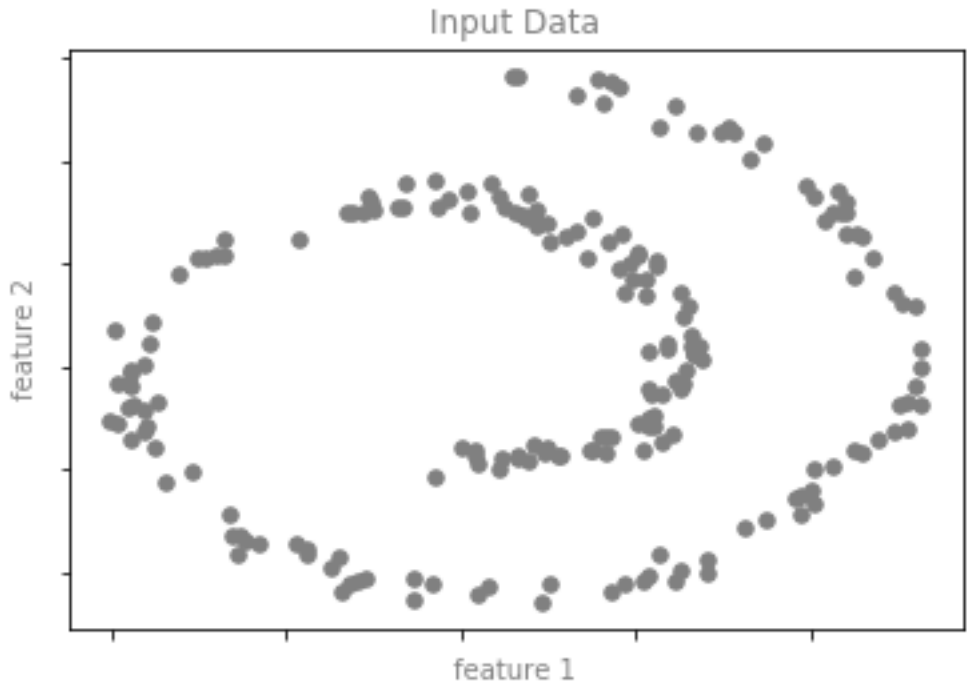



Clurtering: *k*-means

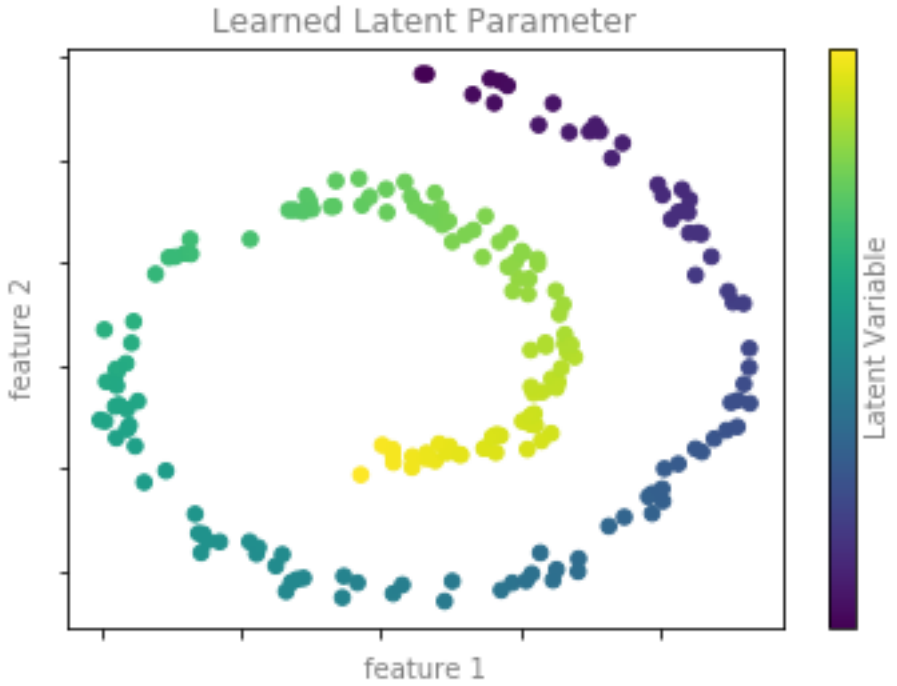
- *k*-means fits a model consisting of *k* cluster centers;
- the optimal centers are assumed to be those that minimize the distance of each point from its assigned center.
- Again, this might seem like a trivial exercise in two dimensions, but as our data becomes larger and more complex, such clustering algorithms can be employed to extract useful information from the dataset.

# Dimensionality reduction: Inferring structure of unlabeled data

- Dimensionality reduction is another example of an unsupervised algorithm, in which labels or other information are inferred from the structure of the dataset itself.
- Dimensionality reduction is a bit more abstract than the examples we looked at before, but generally it seeks to pull out some low-dimensional representation of data that in some way preserves relevant qualities of the full dataset.
- Different dimensionality reduction routines measure these relevant qualities in different ways, as we will see in a class about Manifold Learning.




The following figure shows a visualization of the results of the Isomap algorithm, a manifold learning algorithm that does exactly this:




- It is drawn from a one-dimensional line that is arranged in a spiral within this two-dimensional space.
- In a sense, you could say that this data is "intrinsically" only one dimensional, though this one-dimensional data is embedded in higher-dimensional space.
- A suitable dimensionality reduction model in this case would be sensitive to this nonlinear embedded structure, and be able to pull out this lower-dimensionality representation.

- Notice that the colors (which represent the extracted one-dimensional latent variable) change uniformly along the spiral, which indicates that the algorithm did in fact detect the structure we saw by eye.
- As with the previous examples, the power of dimensionality reduction algorithms becomes clearer in higher-dimensional cases.
  - For example, we might wish to visualize important relationships within a dataset that has 100 or 1,000 features. Visualizing 1,000-dimensional data is a challenge, and one way we can make this more manageable is to use a dimensionality reduction technique to reduce the data to two or three dimensions.
- Some important dimensionality reduction algorithms that we will discuss are principal component analysis and various manifold learning algorithms, including Isomap and locally linear embedding.

# Summary

- Supervised learning: Models that can predict labels based on labeled training data
  - *Classification*: Models that predict labels as two or more discrete categories
  - *Regression*: Models that predict continuous labels

- Unsupervised learning: Models that identify structure in unlabeled data
  - *Clustering*: Models that detect and identify distinct groups in the data
  - *Dimensionality reduction*: Models that detect and identify lower-dimensional structure in higher-dimensional data

# Introducing Scikit-Learn

- Scikit-Learn provides efficient versions of a large number of common algorithms for machine learning.
- Scikit-Learn is characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation.
- A benefit of this uniformity is that once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward.
- This class provides an overview of the Scikit-Learn API; a solid understanding of these API elements will form the foundation for understanding the deeper practical discussion of machine learning algorithms and approaches in the following classes.
- We will start by covering data representation in Scikit-Learn, followed by covering the Estimator API, and finally go through a more interesting example of using these tools for exploring a set of images of hand-written digits.

# Data Representation in Scikit-Learn

- Machine learning is about creating models from data.
- How data can be represented in order to be understood by the computer?
- The best way to think about data within Scikit-Learn is in terms of tables of data.
- Data as table:
  - A basic table is a two-dimensional grid of data, in which the rows represent individual elements of the dataset, and the columns represent quantities related to each of these elements.

# Example: *Iris dataset*

- Consider the Iris dataset, famously analyzed by Ronald Fisher in 1936.
  - https://en.wikipedia.org/wiki/Iris_flower_data_set
- Download this dataset in the form of a Pandas DataFrame using the seaborn library: http://seaborn.pydata.org/

| | 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 |

Here,

- each row of the data refers to a single observed flower, and the number of rows is the total number of flowers in the dataset.
- In general, we will refer to the rows of the matrix as *samples*, and the number of rows as *n_samples*.

Likewise,

- each column of the data refers to a particular quantitative piece of information that describes each sample.
- In general, we will refer to the columns of the matrix as *features*, and the number of columns as *n_features*.

# Cont'd: Features matrix

- This table layout makes clear that the information can be thought of as a two-dimensional numerical array or matrix, which is called the *features matrix*.
- By convention, this features matrix is often stored in a variable named X.
- The features matrix is assumed to be two-dimensional, with *shape [n_samples, n_features]*, and is most often contained in a NumPy array or a Pandas DataFrame, though some Scikit-Learn models also accept SciPy sparse matrices.
- The samples (i.e., rows) always refer to the individual objects described by the dataset.
  - For example, the sample might be a flower, a person, a document, an image, a sound file, a video, or anything else you can describe with a set of quantitative measurements.
- The features (i.e., columns) always refer to the distinct observations that describe each sample in a quantitative manner. Features are generally real-valued, but may be Boolean or discrete-valued in some cases.

# Cont'd: Target array

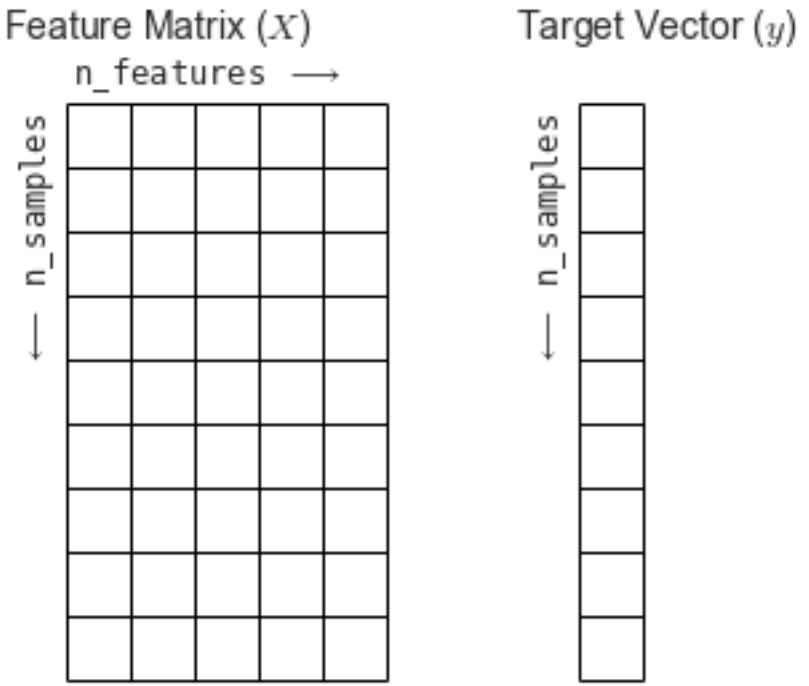
- The *target array* (or *a label*) is called *y* by convention.
- The target array is usually one dimensional, with length *n_samples*, and is generally contained in a NumPy array or Pandas Series.
- The target array may have continuous numerical values, or discrete classes/labels.
- Often one point of confusion is how the target array differs from the other features columns:
  - The distinguishing feature of the target array is that it is usually the quantity we want to predict from the data: in statistical terms, it is the dependent variable.
  - For example, in the preceding data we may wish to construct a model that can predict the species of flower based on the other measurements; in this case, the species column would be considered the target array.

# Cont'd

- For use in Scikit-Learn, we will extract the features matrix and target array from the DataFrame, which we can do using some of the Pandas DataFrame operations discussed in the Pandas class.

- For use in Scikit-Learn, we will extract the features matrix and target array from the DataFrame, which we can do using some of the Pandas DataFrame operations discussed in the Pandas class.

To summarize, the expected layout of features and target values is visualized in the following diagram:




- With this data properly formatted, we can move on to consider the *estimator* API of Scikit-Learn:

# Scikit-Learn's Estimator API

- The Scikit-Learn API is designed with the following guiding principles in mind ,
  - *Consistency*: All objects share a common interface drawn from a limited set of methods, with consistent documentation.
  - *Inspection*: All specified parameter values are exposed as public attributes.
  - *Limited object hierarchy*: Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas DataFrames, SciPy sparse matrices) and parameter names use standard Python strings.
  - *Composition*: Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.
  - *Sensible defaults*: When models require user-specified parameters, the library defines an appropriate default value.
- In practice, these principles make Scikit-Learn very easy to use, once the basic principles are understood.
- Every machine learning algorithm in Scikit-Learn is implemented via the Estimator API, which provides a consistent interface for a wide range of machine learning applications.

# Cont'd: Basics of the API

Most commonly, the steps in using the Scikit-Learn estimator API are as follows:

1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn.
2. Choose model hyperparameters by instantiating this class with desired values.
3. Arrange data into a features matrix and target vector following the discussion above.
4. Fit the model to your data by calling the fit() method of the model instance.
5. Apply the Model to new data:
   - For supervised learning, often we predict labels for unknown data using the predict() method.
   - For unsupervised learning, we often transform or infer properties of the data using the transform() or predict() method.

# Supervised learning example: Simple linear regression

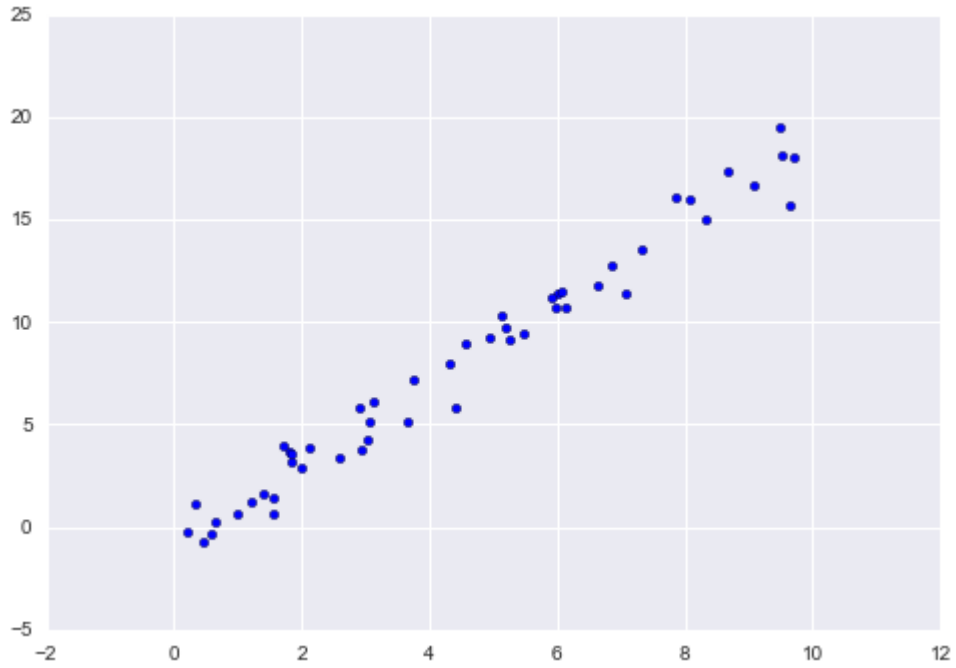
**1. Choose a class of model**

- In Scikit-Learn, every class of model is represented by a Python class. So, for example,
  - to compute a simple linear regression model
  - to import the linear regression class:

```
from sklearn.linear_model import LinearRegression
```

Note that:

other more general linear regression models exist as well; you can read more about them in the sklearn.linear_model online documentation.



## 2. Choose model hyperparameters

- An important point is that *a class of model* is not the same as *an instance of a model*.
- Once a model class is decided, there are still some options open, you might need to answer one or more questions like the following:
  - Would we like to fit for the offset (i.e., y-intercept)?
  - Would we like the model to be normalized?
  - Would we like to preprocess our features to add model flexibility?
  - What degree of regularization would we like to use in our model?
  - How many model components would we like to use?
- These are examples of the important choices that must be made once the model class is selected. These choices are often represented as **hyperparameters**, or **parameters** that must be set before the model is fit to data. In Scikit-Learn, hyperparameters are chosen by passing values at model instantiation.
- For our linear regression example, we can instantiate the LinearRegression class and specify that we would like to fit the intercept using the fit_intercept hyperparameter:

```
model = LinearRegression(fit_intercept=True)
```

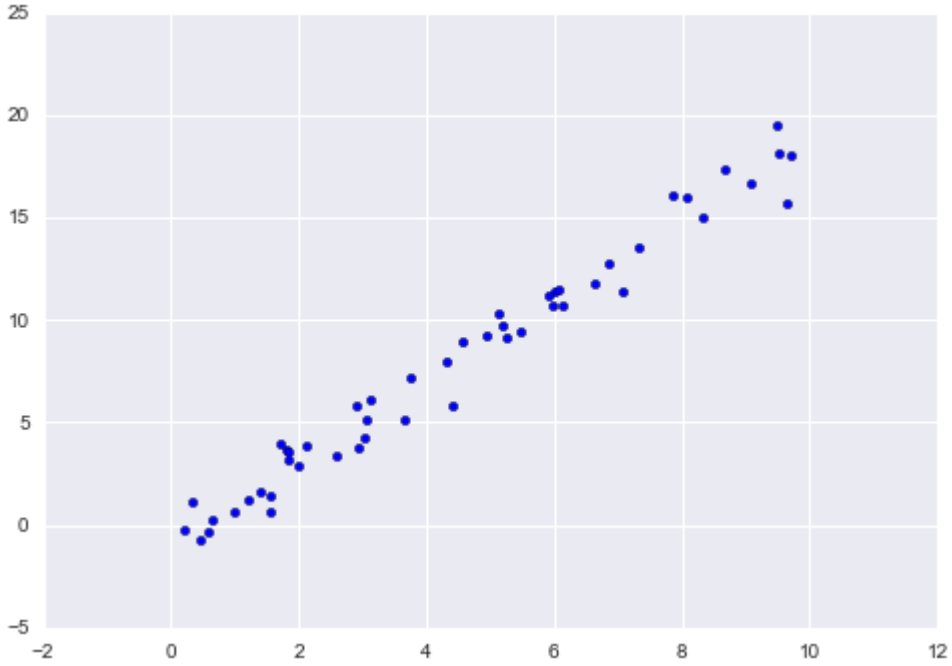
- Keep in mind that when the model is instantiated, the only action is the storing of these hyperparameter values. In particular, we have not yet applied the model to any data: the Scikit-Learn API makes very clear the distinction between *choice of model* and *application of model to data*.

# Cont'd

**3. Arrange data into a features matrix and target vector**

- Here the target variable *y* is already in the correct form (a length-*n_samples* array)
- Make the data x a matrix of size *[n_samples, n_features]*
- In this case, this amounts to a simple reshaping of the one-dimensional array:

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



# 4. Fit the model to your data

- Using fit() method to apply the model to data: model.fit(X, y)
  - This fit() command causes a number of model-dependent internal computations to take place.
  - The results of these computations are stored in model-specific attributes that the user can explore.
  - In Scikit-Learn, by convention all model parameters that were learned during the fit() process have trailing underscores; for example in this linear model, we have the following:
    *model.coef_  model.intercept_*
    - These two parameters represent the slope and intercept of the simple linear fit to the data. Comparing to the data definition, we see that they are very close to the input slope of 2 and intercept of -1.
- One question that frequently comes up regards the uncertainty in such internal model parameters. In general, Scikit-Learn does not provide tools to draw conclusions from internal model parameters themselves: interpreting model parameters is much more a statistical modeling question than a machine learning question.
- Machine learning rather focuses on what the model predicts. If you would like to dive into the meaning of fit parameters within the model, other tools are available, including the Statsmodels Python package.

## 5. Predict labels for unknown data

- Once the model is trained, the main task of supervised machine learning is to evaluate it based on what it says about new data that was not part of the training set.
- In Scikit-Learn, this can be done using the predict() method. For the sake of this example, our "new data" will be a grid of x values, and we will ask what y values the model predicts:

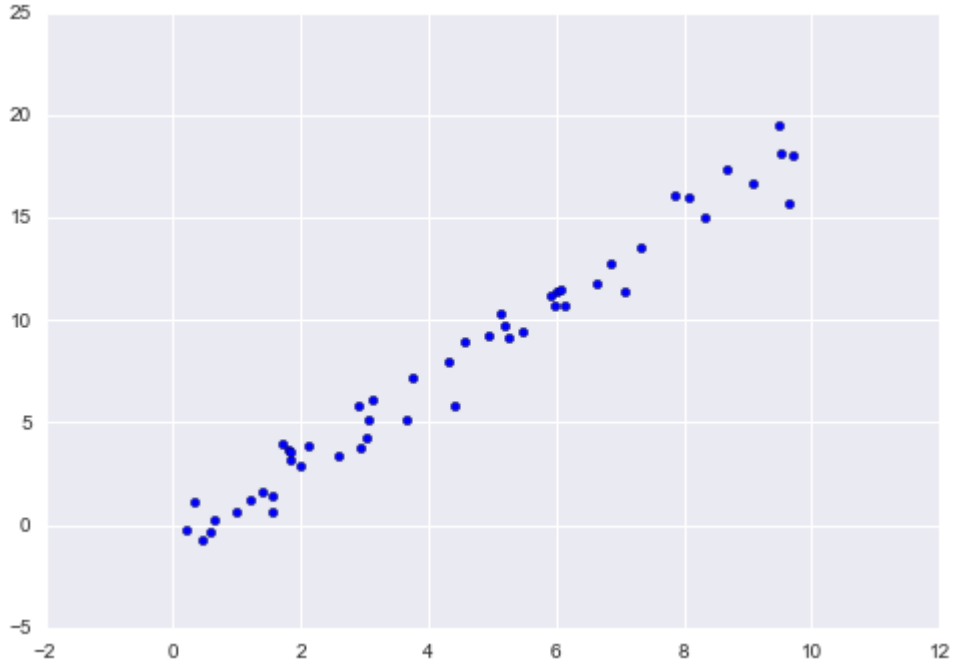
```
xfit = np.linspace(-1, 11)
```

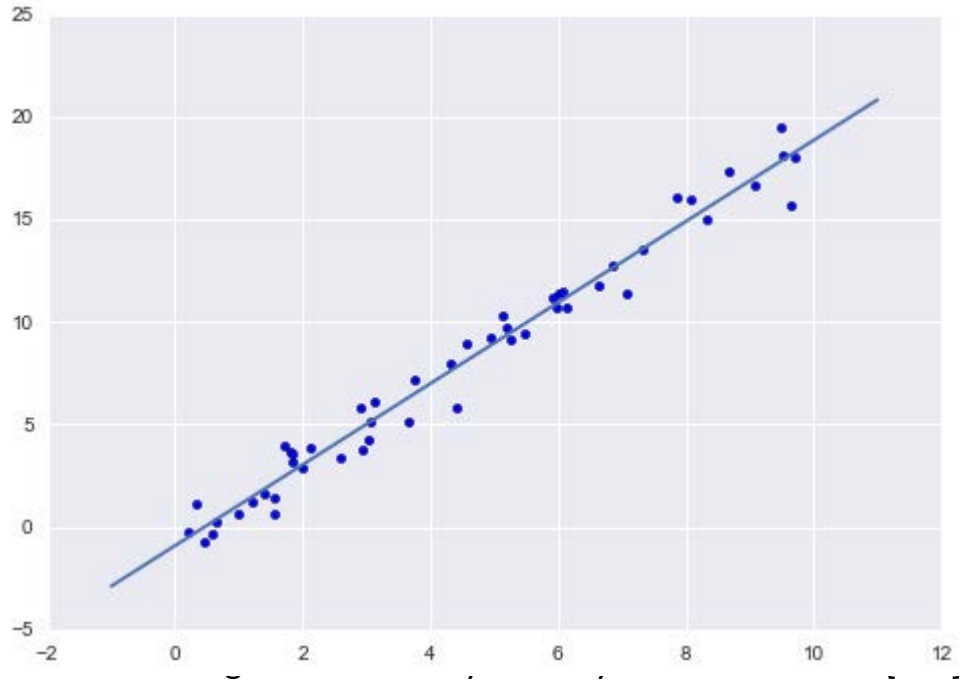
- As before, we need to coerce these x values into a [n_samples, n_features] features matrix, after which we can feed it to the model:

```
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

- Finally, let's visualize the results by plotting first the raw data, and then this model fit:

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```



# Supervised learning example: Iris classification

- The question is: given a model trained on a portion of the Iris data, how well can we predict the remaining labels?
- Gaussian naive Bayes, an extremely simple generative model, proceeds by assuming each class is drawn from an axis-aligned Gaussian.
  - Because it is so fast and has no hyperparameters to choose, Gaussian naive Bayes is often a good model to use as a baseline classification, before exploring whether improvements can be found through more sophisticated models.
- Split a new data into a *training set* and a *testing set*.
  - This could be done by hand, but it is more convenient to use the train_test_split utility function:

# Cont'd

- With the data arranged, we can follow our recipe to predict the labels:
  - from sklearn.naive_bayes import GaussianNB  ***# 1. choose model class***
  - model = GaussianNB()  ***# 2. instantiate model***
  - model.fit(Xtrain, ytrain)  ***# 3. fit model to data***
  - y_model = model.predict(Xtest)  ***# 4. predict on new data***

- Finally, using the accuracy_score utility to see the fraction of predicted labels that match their true value: accuracy_score(ytest, y_model) ~ 0.97368
- With an accuracy topping 97%, we see that even this very naive classification algorithm is effective for this particular dataset!

# Unsupervised learning example: Iris dimensionality

- Recall that the Iris data is four dimensional: four features recorded for each sample.
- The task of dimensionality reduction is to ask whether there is a suitable lower-dimensional representation that retains the essential features of the data.
  - Often dimensionality reduction is used as an aid to visualizing data: after all, it is much easier to plot data in two dimensions than in four dimensions or higher!
- Principal Component Analysis (PCA) is a fast linear dimensionality reduction technique. The model will be asked to return two components—that is, a two-dimensional representation of the data.

# Cont'd

- Following the sequence of steps outlined earlier, we have:
  - from sklearn.decomposition import PCA  ***# 1. Choose the model class***
  - model = PCA(n_components=2)  ***# 2. Instantiate the model with hyperparameters***
  - model.fit(X_iris)  ***# 3. Fit to data. Notice y is not specified!***
  - X_2D = model.transform(X_iris)  ***# 4. Transform the data to two dimensions***
- Now let's plot the results. A quick way to do this is to insert the results into the original Iris DataFrame, and use Seaborn's lmplot to show the results:
- In the two-dimensional representation, the species are fairly well separated, even though the PCA algorithm had no knowledge of the species labels!
- This indicates that a relatively straightforward classification will probably be effective on the dataset, as we saw before.

# Unsupervised learning: Iris clustering

- Applying clustering to the Iris data.
- A clustering algorithm attempts to find distinct groups of data without reference to any labels.
- Here we will use a powerful clustering method called a Gaussian mixture model (GaussianMixture).
- A GaussianMixture attempts to model the data as a collection of Gaussian blobs.

# Cont'd

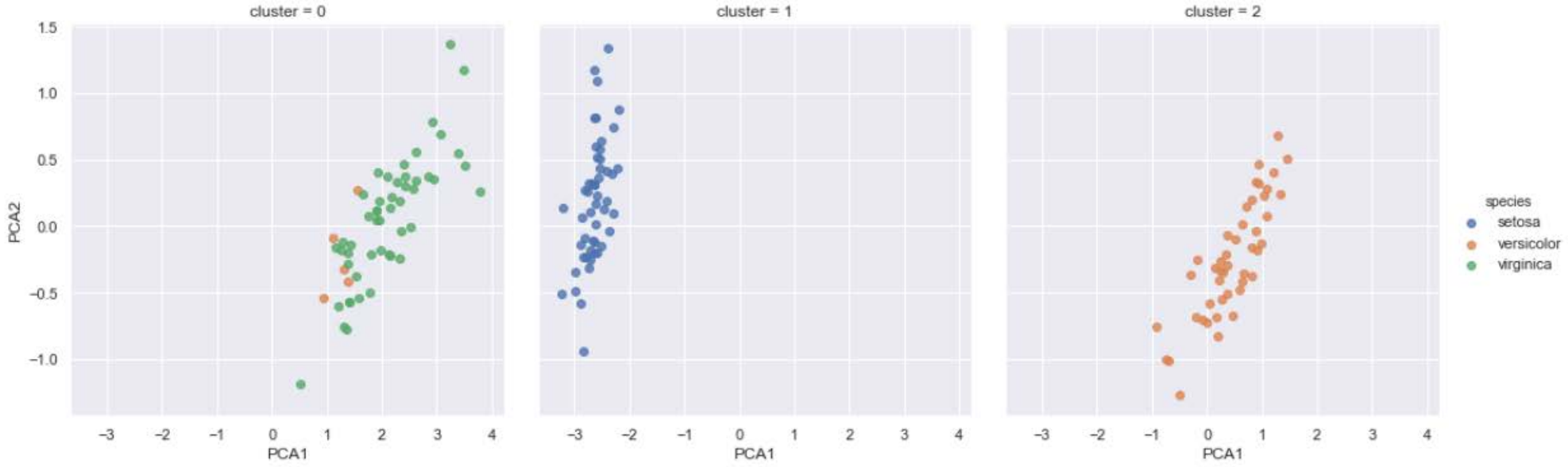
- We can fit the Gaussian mixture model as follows:

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

- As before, we will add the cluster label to the Iris DataFrame and use Seaborn to plot the results:

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

- By splitting the data by cluster number, how exactly well the GaussianMixture algorithm has recovered the underlying label:
  - The *setosa* species is separated perfectly within cluster 0, while there remains a small amount of mixing between *versicolor* and *virginica*.
  - This means that even without an expert to tell us the species labels of the individual flowers, the measurements of these flowers are distinct enough that we could *automatically* identify the presence of these different groups of species with a simple clustering algorithm!
  - This sort of algorithm might further give experts in the field clues as to the relationship between the samples they are observing.

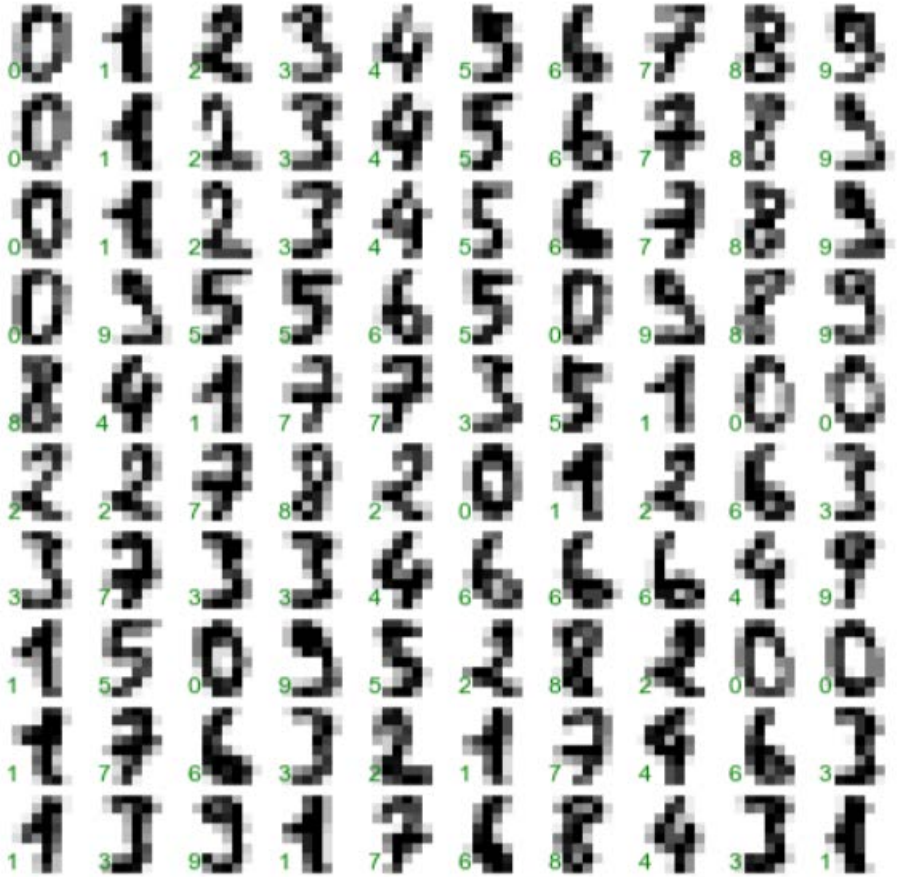


# Example: Exploring Hand-written Digits

- To demonstrate these principles on a more interesting problem, let's consider one piece of the optical character recognition problem: the identification of hand-written digits.
- In the wild, this problem involves both locating and identifying characters in an image.
- Using Scikit-Learn's set of pre-formatted digits, which is built into the library.

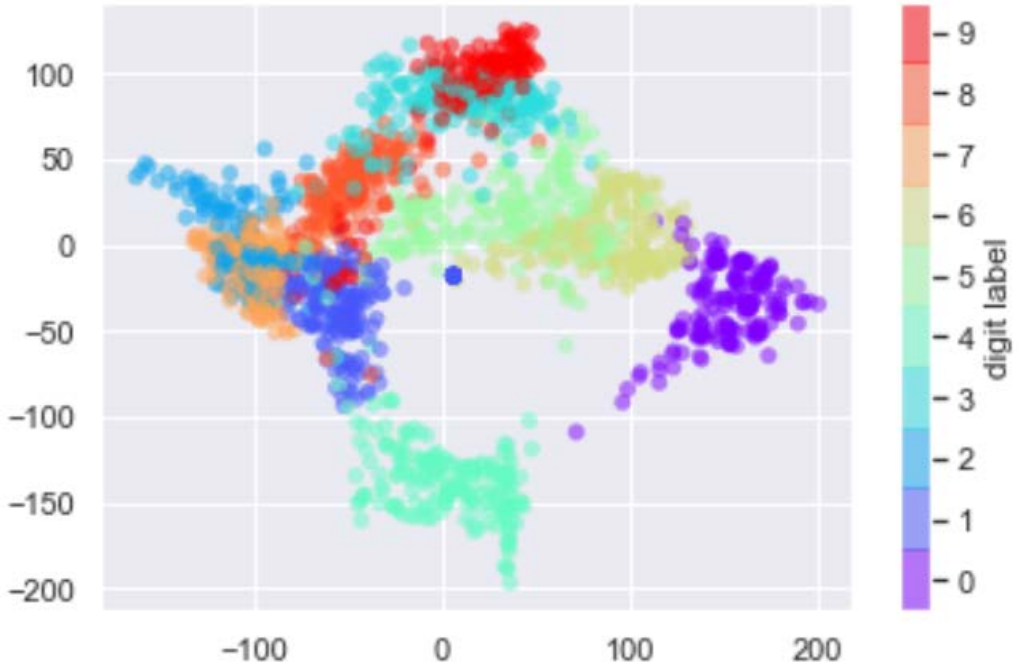
# 1. Loading and visualizing the digits data

- First, need a two-dimensional [n_samples, n_features] representation.
- To accomplish this, treating each pixel in the image as a feature: that is, flattening out the pixel arrays so as to have a length-64 array of pixel values representing each digit.
- Additionally, need the target array, which gives the previously determined label for each digit.
- These two quantities are built into the digits dataset under the data and target attributes, respectively:



# 2. Unsupervised learning: Dimensionality reduction

- Reduce the dimensions from 64 to 2, using an unsupervised method.
- Here, a manifold learning algorithm called *Isomap* will be used, and transformed the data to two dimensions:

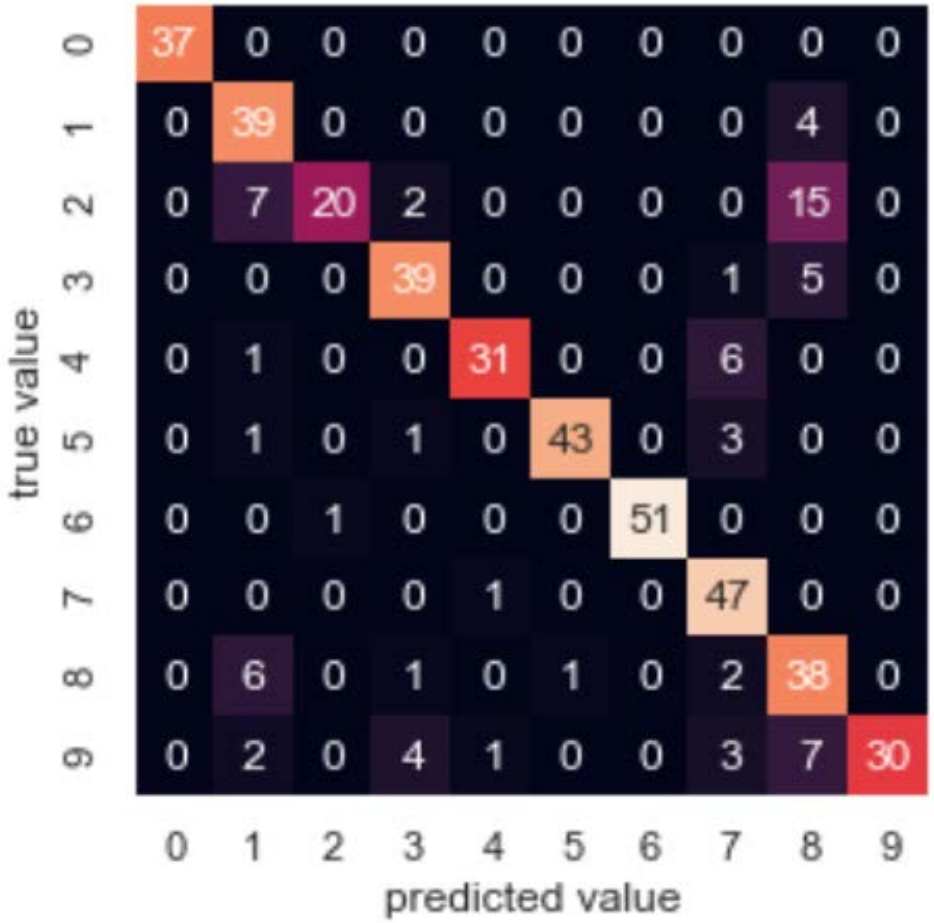



- This plot gives us some good intuition into how well various numbers are separated in the larger 64-dimensional space.
- For example, zeros (in purple) and ones (in dark blue) have very little overlap in parameter space. Intuitively, this makes sense: a zero is empty in the middle of the image, while a one will generally have ink in the middle.
- There is a more or less continuous spectrum between ones and fours because some people draw ones with "hats" on them, which cause them to look similar to fours.

Overall, however, the different groups appear to be fairly well separated in the parameter space: this tells us that even a very straightforward supervised classification algorithm should perform suitably on this data.

# 3. Classification on digits

- Apply a classification algorithm to the digits: split the data into a training and testing set, and fit a Gaussian naive Bayes model:
- Then check its accuracy by comparing the true values of the test set to the predictions: accuracy_score(ytest, y_model) ~ 0.8333
- Use *confusion matrix* to find where is the wrong.




This shows us where the mis-labeled points tend to be: for example, a large number of twos here are mis-classified as either ones or eights.

# Cont'd

- Another way to gain intuition into the characteristics of the model is to plot the inputs again, with their predicted labels. Use green for correct labels, and red for incorrect labels:
- Examining this subset of the data, we can gain insight regarding where the algorithm might be not performing optimally.
- To go beyond our 80% classification rate, we might move to a more sophisticated algorithm such as support vector machines, random forests or another classification approach. I will talk these algorithms and examples in the next classes.

# Summary

- In this class we have covered the essential features of the Scikit-Learn data representation, and the estimator API. Regardless of the type of estimator, the same import/instantiate/fit/predict pattern holds.
- Armed with this information about the estimator API, you can explore the Scikit-Learn documentation and begin trying out various models on your data.
- In the next class, we will explore perhaps the most important topic in machine learning: how to select and validate your model.

# The End!

Assignment: Practice the example code I ran in this class.

# Adjustment

- In the following classes, I will teach machine learning.
- In the original course design, I was going to teach a little bit of parallel computing (1 class) and deep learning (1 class).
- I will open a new course in summer or fall semester, which would be called “high performance deep learning”.
- In this course, I will focus on data analysis and machine learning in order for students to learn in depth.