EML4930/EML6934: Lecture 11

Scikit-Learn: Machine learning and **Regression**

Charles Jekel

November 9, 2017

Quiz feedback

```python
# let's consider an arbitrary function
def my_fun():
    return

# this won't execute the function, nor will it pass an error
my_fun

# infact I can even create an alias name of the function
new = my_fun

# again this won't execute the function
new

# You need to use () to execute a function
new()
```

Quiz feedback plotting

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

x = np.linspace(0,20)
y = 2.0*x - .3

plt.figure()
plt.plot(x,y)
plt.show # this doesn't display the figure,
# it's just the name of the function!

# you need to execute the function in order
# to display the figure
plt.show()
```

Topic for today: Scikit-Learn

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Dr. Jake VanderPlass Python Data Science Handbook: Essential Tools for Working with Data.
`http://shop.oreilly.com/product/0636920034919.do`

The textbook available for free in the form of Jupyter notebooks which can be viewed at
`https://github.com/jakevdp/PythonDataScienceHandbook` or
`http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/Index.ipynb`

Scikit-Learn has great documentation

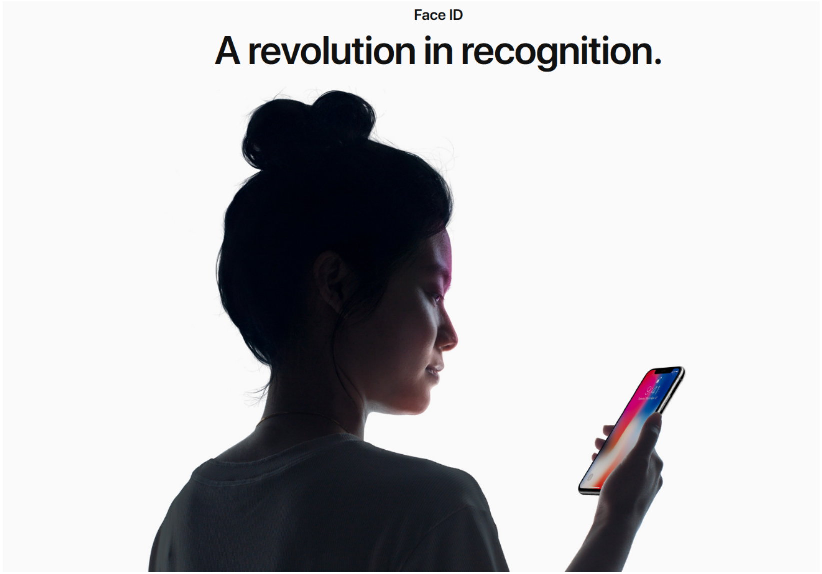
with plenty of tutorials and examples for each class... You should take a look at it

http://scikit-learn.org/

What is classification? - Supervised learning

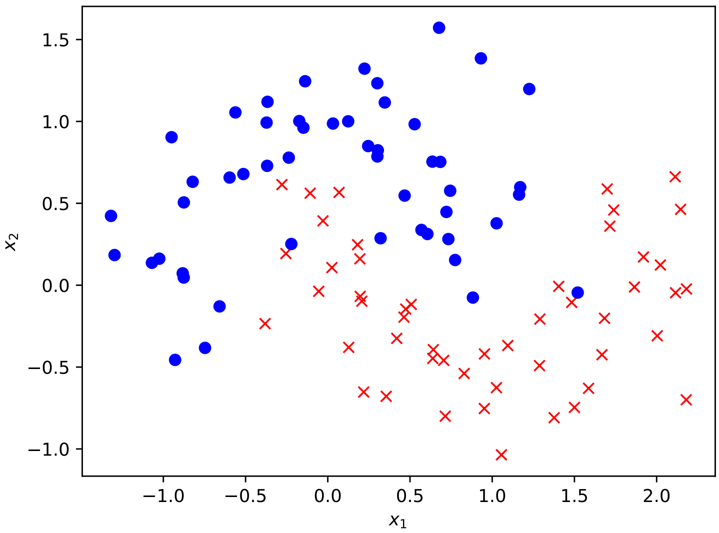
- Given a collection of labeled features: Can we find a pattern in the features to predict the labels?
- Classification is used everywhere!
- Ex: You are applying for a loan. The bank knows your income, job stability, and debt. Are you approved for the load?
- Features: income, job stability, debt
- Labels: approve or reject
- Binary classification problem!

New iPhone X - unlock with your face



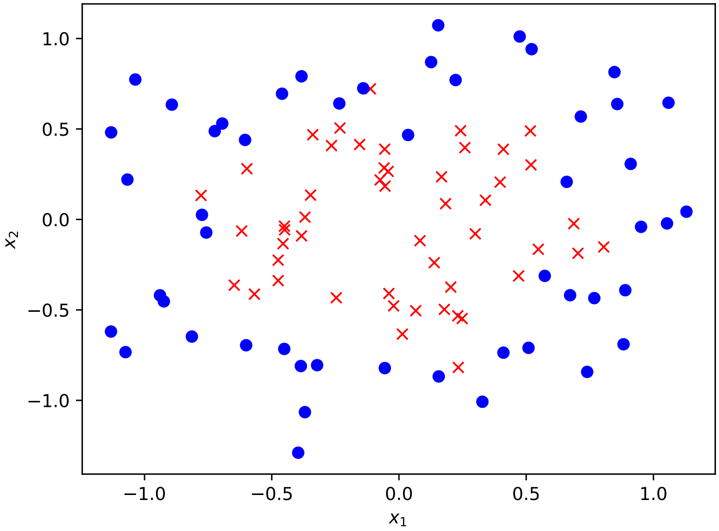
Classification example 1

What $f(x_1, x_2)$ sepeates the data?

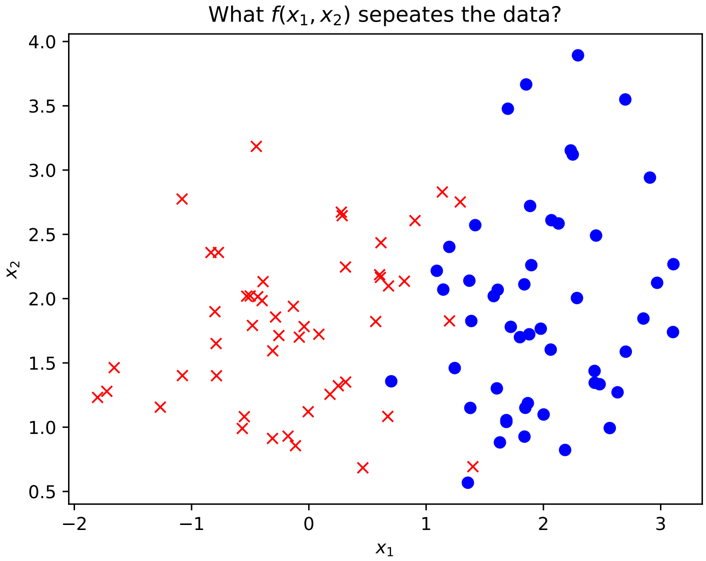


Classification example 2

What $f(x_1, x_2)$ sepeates the data?



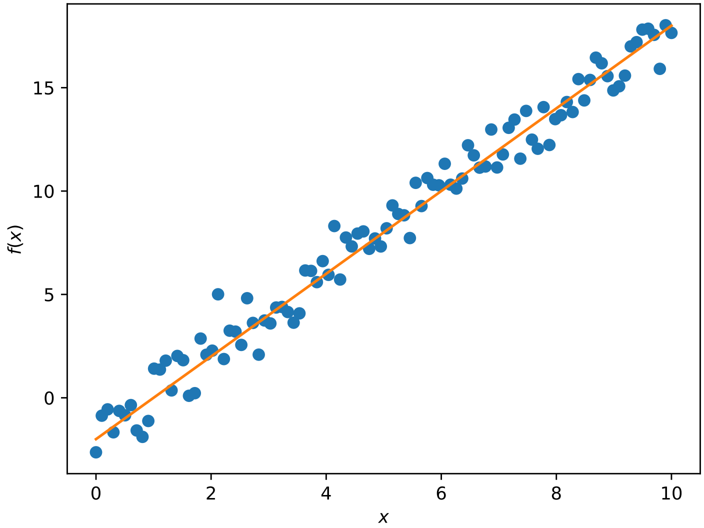
Classification example 3



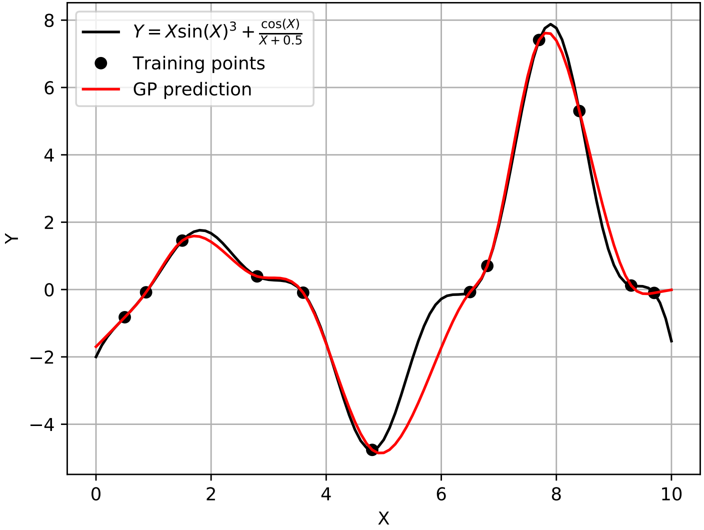
What is regression?

- Can we find the trend in data?
- Can we predict the values in-between data points?
- Classification is used everywhere!
- Ex: You are applying for a loan. The bank knows your income, job stability, and debt. How much of a loan can you get approved for?
- Variables: income, job stability, debt
- Output: How much $ you get?

Linear regression: fitting a line to data



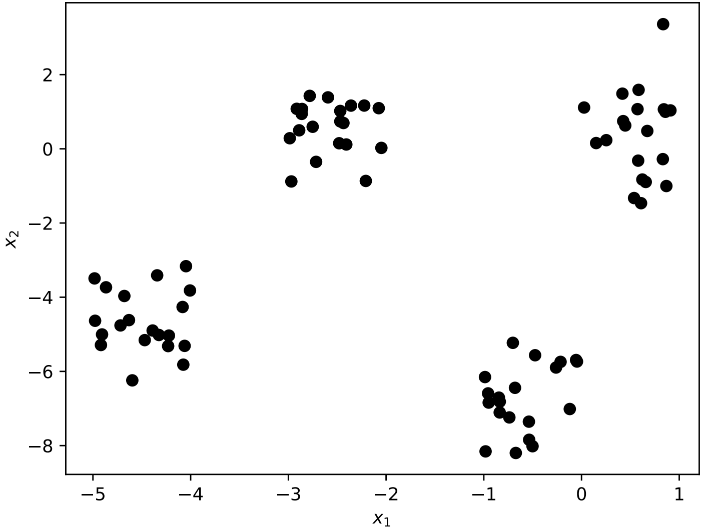
Kriging: using correlation between data



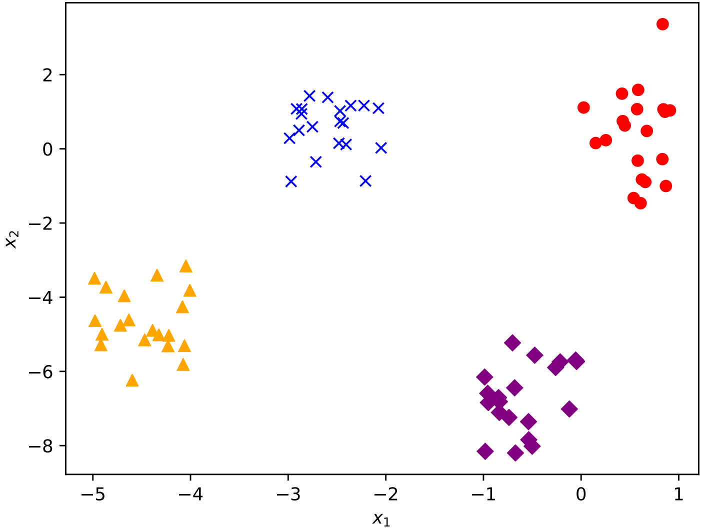
What is clustering? Unsupervised learning

- Can we find the structure in data?
- Can we detect and identify groups in data?
- Ex: We have a bunch of data related to income, job stability, and debt. Can we identify a pattern in the data?
- The key is that we don't know the label or value.
- Ex: Used to identify similarities in protein chains

Clustering: Can we categorize this data?



Clustering output



So what does the data look like for Scikit-Learn

Out input data will be a matrix (or numpy array) with n number of samples and d number of design variables (or **features**)

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2d} \\ x_{31} & x_{32} & x_{33} & \cdots & x_{3d} \\ \vdots & \vdots * \vdots & \vdots & \vdots & \\ x_{n1} & x_{n2} & x_{n3} & \cdots & x_{nd} \end{bmatrix} \tag{1}$$

Each row of $\mathbf{X}$ represents a single sample.

Out output data will be a vector $\mathbf{y}$ (usually a numpy array) with n number samples.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \tag{2}$$

Let’s consider a simple linear regression example

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

# let’s create some arbitrary data
# and fit a line to it

np.random.seed(33)
x = 10.0*np.random.random(100)
noise = np.random.normal(loc=0.5, size=100)
y = 2.3*x - 3 + noise
plt.figure()
plt.plot(X,y, ’o’)
plt.show()
```

Building a linear regression model

```
from sklearn.linear_model import LinearRegression
# LinearRegression is a Python class that we'll use to create
# our linear model object

# we can't pass a vector into scikit-learn
# instead we need to pass a feature matrix
# we can do this by reshaping x
X = x.reshape(-1,1)

# we initialize a model object from the LinearRegression class
# to begin the process. With the initialization, we can pass
# model hyper-parameters, in this case we pass that a
# y intercept be determined
model = LinearRegression(fit_intercept=True)
```

Fitting the linear regression model

```
# simply pass your training data into the model's fit function
# which will run an 'optimization' selecting the best parameters
# == this is the same as an ordinary least squares fit ==
model.fit(X,y)

# in this case there are two parameters for the line
# y = mx + b
print('m = ', model.coef_)
print('b = ', model.intercept_)
# score calculates the R^2 (Coefficient of determination)
# for a given input and output
my_score = model.score(X,y)

print('R^2 = ', my_score)
```

Predicting for new x values

```
# let's generate new x values from slightly smaller than min(x)
# to the slightly larger than the max(x)
x_hat = np.linspace(np.min(x)-1.0,  np.max(x)+1.0, 100)

# Again x_hat is a vector, and must be reshaped
# to be a 2 dimensional array
X_hat = x_hat.reshape(-1, 1)

# we can find new y_hat values by using the model.predict functi
y_hat = model.predict(X_hat)

# let's plot the results
plt.figure()
plt.plot(X,y, 'o')
plt.plot(x_hat, y_hat, '-')
plt.show()
```

What about polynomials?

So let's consider a toy polynomial problem where

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \tag{3}$$

on the domain $-10.0 \leq x \leq 10.0$

```
# generate artificial data
np.random.seed(19)
x = (10.0 + 10.0)* np.random.random(100) - 10.0
noise = np.random.normal(scale=5.0,size=100)
y = -1.0 + 2.2*x - 0.5*x**2 + noise

plt.figure()
plt.plot(x,y, 'o')
plt.show()
```

Scikit-Learn includes a polynomial prepossessing tool

So in order the least squares regression, we need to construct the regression matrix **X**. Scikit-Learn includes a Polynomial class for doing this easily.

```
from sklearn.preprocessing import PolynomialFeatures
# again we need to transfer x to a matrix...
# the reason is that sklearn doesn't deal with 1D data...
X = x.reshape(-1,1)

# create a poly object from the PolynomialFeatures class
# in our case we'll be fitting a second order polynomial
poly = PolynomialFeatures(degree=2)

# now we use poly to transform our X into a Regression matrix!
X = poly.fit_transform(X)
print(X)
# note this transformation includes the y intercept!
```

Fitting the second order polynomial

```
# create quad object from linear regression
# the default is that fit_intercept = False
# we don't want fit_intercept because it's included in X
quad = LinearRegression(fit_intercept=False)

# perform the least squares fit
quad.fit(X,y)

# predict for new values of x
x_hat = np.linspace(np.min(x), np.max(x), 100)
x_hat = x_hat.reshape(-1,1)
X_hat = poly.fit_transform(x_hat)
y_hat = quad.predict(X_hat)

# plot the results
plt.figure()
plt.plot(x,y, 'o')
```

What are the resulting parameters?

Recall

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \tag{4}$$

You β parameters are stored in the quad object.

```
print('beta0 =', quad.coef_[0])
print('beta1 =', quad.coef_[1])
print('beta2 =', quad.coef_[2])
```

What about polynomials for higher dimensions?

Consider a polynomial response surface

$$f(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_1 x_2 + \beta_5 x_2^2 \tag{5}$$

```
# let's generate a 2 dimensional polynomial problem
np.random.seed(121)
X = np.random.random((100,2))
# since this is a matrix, there is no need to reshape
y = 1.0 + X[:,0] + 0.9*X[:,1] + 1.2*X[:,0]**2  \
-1.3*X[:,0]*X[:,1] - 3.0*X[:,1]**2
# plot the data
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:,0], X[:,1], y, 'o')
ax.set(xlabel='$x_1$', ylabel='$x_2$', zlabel='$f(x_1,x_2$')
fig.show()
```

Same process for creating regression matrix and fitting

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
X_train = poly.fit_transform(X)



# fit the model
model = LinearRegression(fit_intercept=False)
model.fit(X_train,y)
# print the model coefficients - you'll see they are exact!
print(model.coef_)

# Let's generate sampeles to predict over the domain
x = np.linspace(np.min(X), np.max(X), 10)
x1,x2 = np.meshgrid(x,x)
X_hat = np.zeros((100,2))
X_hat[:,0] = x1.flatten()
X_hat[:,1] = x2.flatten()
```

Predicting and comparing the result

```
X_test = poly.fit_transform(X_hat)

# predict for X_test
Y_hat = model.predict(X_test)

# reshape y_hat
y_hat = Y_hat.reshape((10,10))

# plot the results
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x1, x2, y_hat)
ax.scatter(X[:,0], X[:,1], y, 'o', color='black')
ax.set(xlabel='$x_1$', ylabel='$x_2$', zlabel='$f(x_1,x_2$')
fig.show()
```

Classifcation with the famous flower database...

Naive Bayes Classification of Iris dataset...

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

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

# sklearn includes tools for validation and model selection
from sklearn.cross_validation import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris,
# this code splits the entire dataset into two
# a training set and a testing set
```

The process for fitting models is same: apply NB

```
# 1. choose model class
from sklearn.naive_bayes import GaussianNB

# 2. instantiate model
model = GaussianNB()

# 3. fit model to data
model.fit(Xtrain, ytrain)

# 4. predict on new data
y_hat = model.predict(Xtest)

# 5. score the model
print(model.score(Xtest,ytest))
```