Machine Learning and Prediction

LECTURE 10

Organization of Lecture 9

- Machine Learning
- Prediction
- Linear Models
- Linear Regression
- Other Regression Analysis
- K-Means Clustering
- Decision Trees

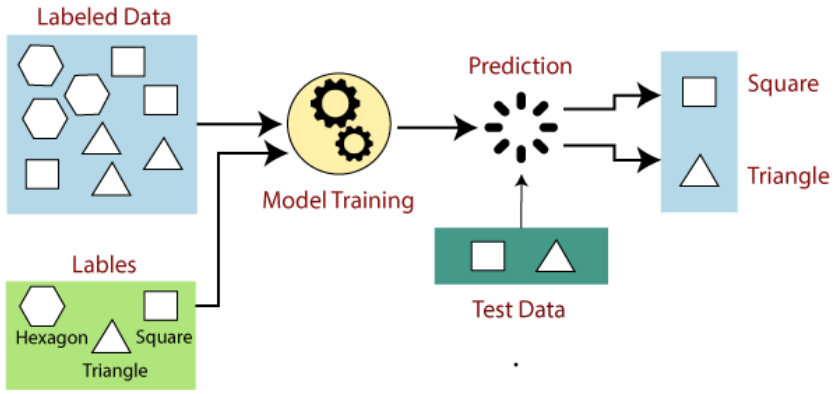
Machine Learning

- Field of study that looks at and constructs algorithms that can learn from and make predictions on experimental data.

- Two major classes of machine learning:
  - Supervised
  - Unsupervised

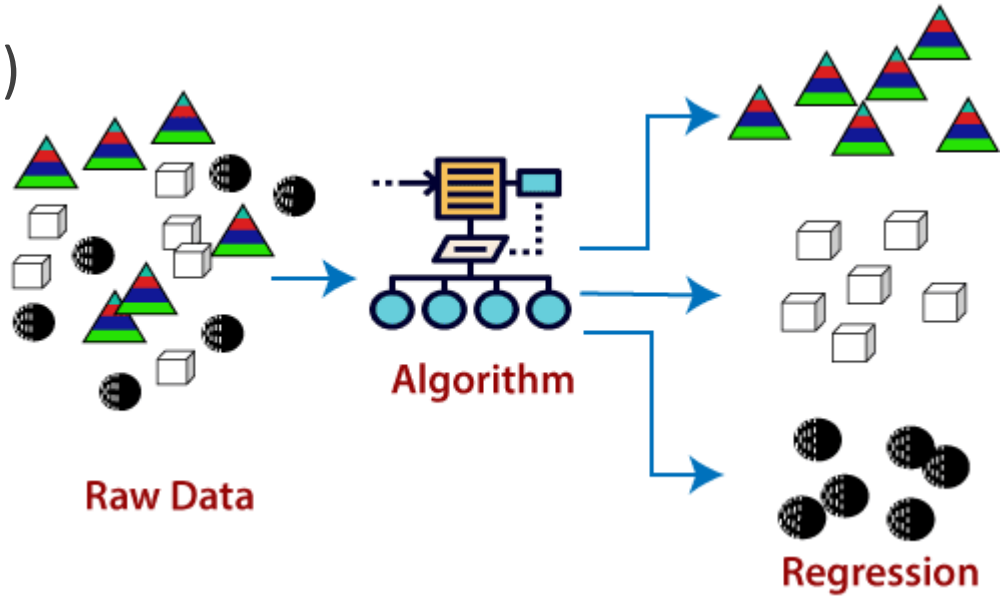
Supervised Learning

- Attempts to infer a predictive function from a *labeled* training set
  - Each observation in data set is known to belong to a certain class
- Methods
  - Linear Regression
  - Random Decision Forests
  - Naïve Bayes Classification
  - Support Vector machines
  - Linear Discriminant Analysis
  - Neural Networks



Unsupervised Learning

- Tries to find hidden structure in *unlabeled* data
- Methods
  - K-means Clustering
  - Community Detection (in networks chapter)
  - Hierarchical Clustering
  - Principal Component Analysis



Python- Machine Learning

- Module **`SciKit-Learn`** has both types of machine learning tools for exploratory and predictive analysis.

- If your ***goal* is to predict** something that you haven't seen yet (rather than explain something you already see), you must set up a predictive experiment.

Designing a Predictive Experiment

- Predictive data analysis is a real scientific experiment
  - Must be organized
  - Important part of the experiment is the assessment and validation of its predictive power.
- Four steps to build, assess and validate a model:
  1. Split input data into training and testing sets (70:30 split recommended).
  2. Build a model using only the training data.
  3. Apply the new model to the testing data.
  4. Evaluate model quality with a “confusion matrix” or quality assurance tool. If model passes the test, it is adequate. If not, repeat steps 2-4.

Confusion matrix

- **Table** used to describe the performance of a classification model on a set of test data for which the true values are known.

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

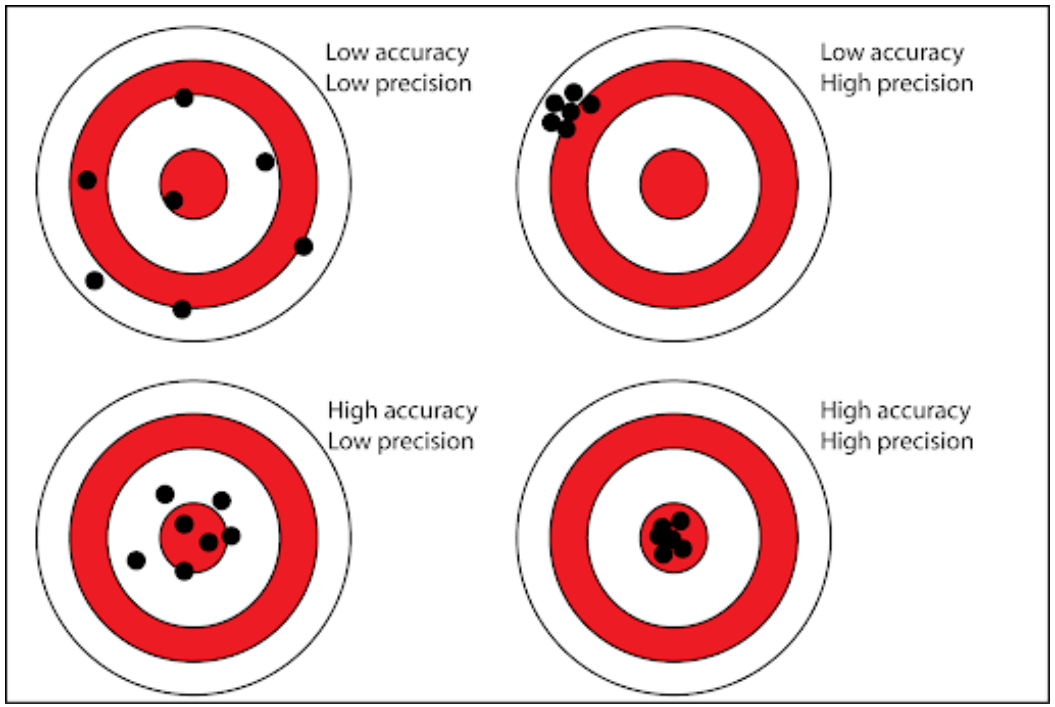
Threshold = 10% 18 components	Prediction (System Test)	
	Fault-prone	Normal
Development Fault-prone	10 (5 new)	8 (1 new)
Development Normal	8 (2 new)	154
Chi-square Results	$\chi^2 = 46.1180$; $p \leq 0.001$	

Release 1 of Medical Record System – showing Significant results in making predictions of fault-prone components after release based on fault-prone components in development..
https://cs.msutexas.edu/~stringfellow/papers/fp.pdf

Quantitative Measures Summarizing Matrix

- *Accuracy* – proportion of correctly classified items:
  - $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
  - Ex. $Acc = \frac{10+154}{10+154+8+8} = 0.91$
  - Most important property

- *Precision* – proportion of true positive to all classified positives:
  - $Precision = \frac{TP}{TP+FP}$
  - Ex. $Prec = \frac{10}{10+8} = 0.56$



Quantitative Measures Summarizing Matrix

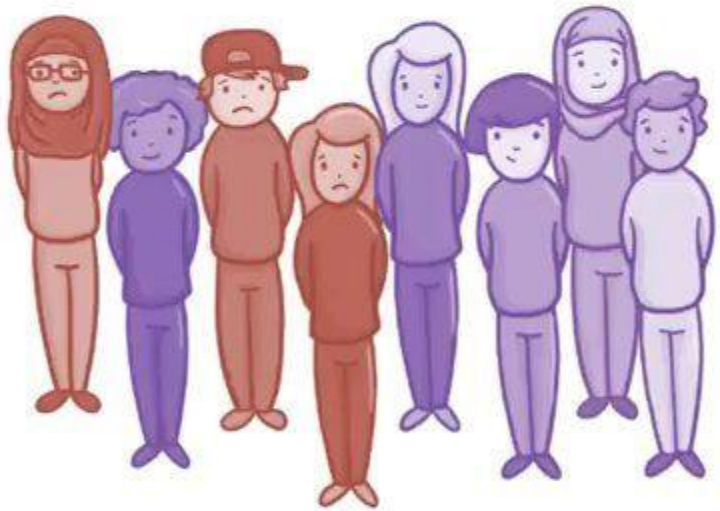
- *Sensitivity* (or recall) – proportion of true positive to all real positives:
  - $sensitivity = \frac{TP}{TP+FN}$ Ex. $sens = \frac{10}{10+8} = 0.56$
  - Describes how good the model is in recognizing the *presence of the property*.
- *Specificity* – proportion of true negatives to all real negatives:
  - $specificity = \frac{TN}{TN+FP}$ Ex. $spec = \frac{154}{154+8} = 0.95$
  - How good is the model at capturing the *absence of the property.*
- If a predictor has both poor specificity and sensitivity, invert it to be a good predictor.

CAN the TEST CORRECTLY IDENTIFY if a PERSON has DIABETES or NOT ?

= VALIDITY

↑↑↑

① SENSITIVITY

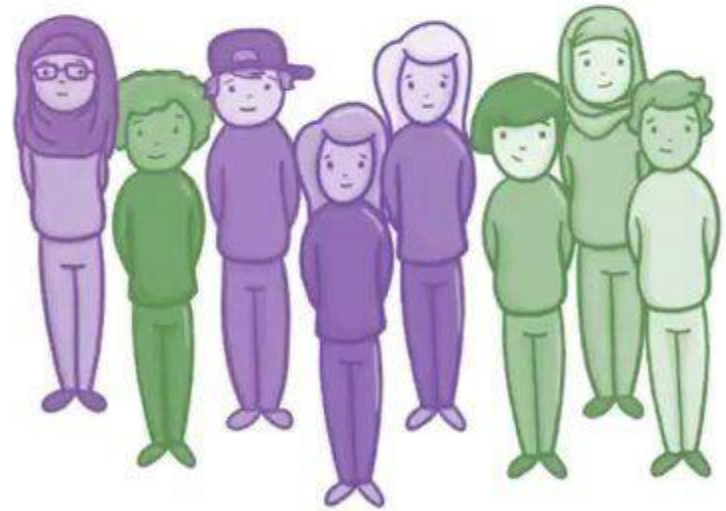


IDENTIFIES MOST PEOPLE WITH the **CONDITION**

NEW SCREENING TEST

↑↑↑

② SPECIFICITY

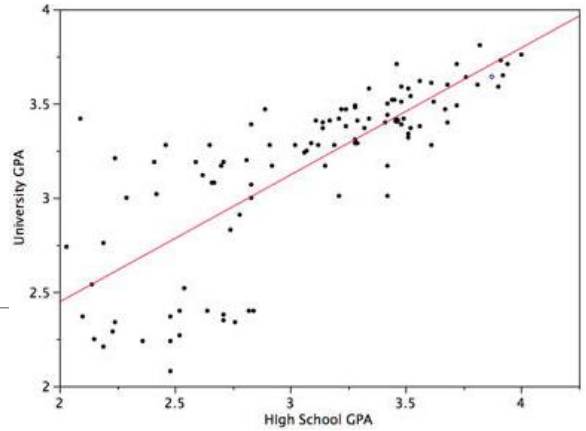


IDENTIFIES MOST PEOPLE WHO **DON'T HAVE the CONDITION**

Approximation Methods

- Data (computed or gathered) is imperfect or incomplete
- Important to have several different methods to display and analyze the data.
- There are three approximation methods that approximate, fit and compress data.
  - Least Squares - presumes flawed data and finds best approximation
  - Cubic Splines – assumes precise information that can be filled in
  - Principal Component Analysis – collapses information in way to describe a large dataset's features without maintaining the entire dataset

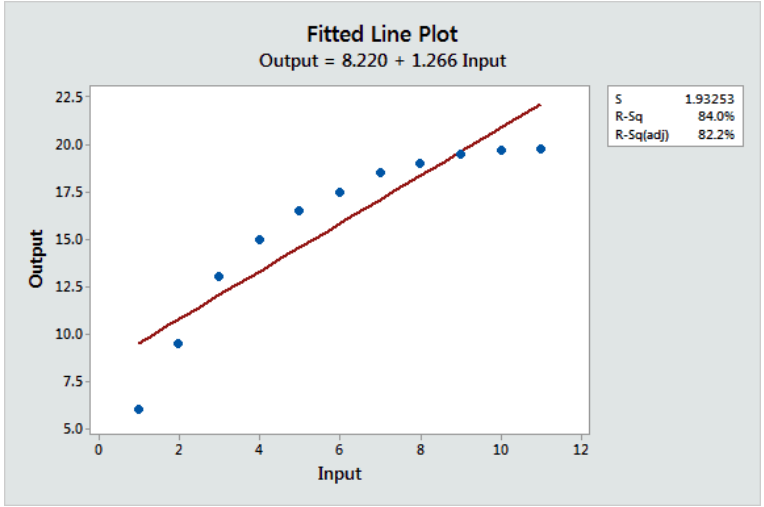
Linear Models



- Definition
  - We can assume parameters a_0, a_1, a_2, …a_n define an approximating function f.
  - If we want to approximate the data with a 2nd order polynomial, then our model $f(x) = a_0 + a_1x + a_2x^2$.
  - While this is not a linear function…it is linear in its defining parameters.
  - This means no **parameter** appears as an exponent, nor is multiplied or divided by another **parameter**.
- A linear relationship (or linear association) is a statistical term **used to describe a straight-line relationship between two variables**.

Fitting a Linear Regression

- Linear Regression is a form of predictive statistical modeling that aims to explain a variable's *variance using a linear model.*
- It is *supervised modeling technique*
  - You must train ("fit") the model before using it for prediction



Least Squares Method

- Optimally approximates disparate data with an assumed *functional model*
- **Standard approach in regression analysis to approximate the solution of overdetermined systems** (sets of equations in which there are more equations than unknowns).
- Relates independent variables (predictors) and dependent values (predictions)
- Treats the predicted value as a linear combination of predictors.

Least Squares Method (2)

- View Demonstration (1:30 minutes)
- Note this method finds values of the intercept and slope coefficient that minimize the sum of the squared errors.
  - The deviations between the actual and predicted values are called *errors*, or *residuals*.
  - If a perfect fit, all the residuals are 0.
  - Another quality of fit is R^2 , where $0 \leq R^2 \leq 1$, shows how much variance the fitted model explains. If a perfect fit, $R^2 = 1$.

How do you calculate least squares?

- Our aim is to calculate the values **m** (slope) and **b** (y-intercept) in the equation of a line : $y = mx + b$

How do you calculate least squares?

- Step 1: For each (x,y) point calculate x^2 and xy.
- Step 2: Sum all x, y, x^2 and xy, which gives us Σx, Σy, Σx^2 and Σxy
- Step 3: Calculate Slope m:
- $m = \frac{N \Sigma(xy) - \Sigma x \Sigma y}{N \Sigma(x^2) - (\Sigma x)^2}$

https://www.mathsisfun.com/data/least-squares-regression.html

- Step 4: Calculate Intercept b:

$$b = \frac{\Sigma y - m \Sigma x}{N}$$

Step 5: Assemble the equation of a line.

Least Squares Example:

- Sam found how many **hours of sunshine** vs how many **ice creams** were sold at the shop from Monday to Friday:

"x" Hours of Sunshine	"y" Ice Creams Sold
2	4
3	5
5	7
7	10
9	15

Step 1

For each (x,y) calculate x^2 and xy:

x	y	x^2	xy
2	4	4	8
3	5	9	15
5	7	25	35
7	10	49	70
9	15	81	135

Step 2

- Sum x, y, x^2 and xy

x	y	x^2	xy
2	4	4	8
3	5	9	15
5	7	25	35
7	10	49	70
9	15	81	135
Σx: 26	**Σy: 41**	**Σx^2: 168**	**Σxy: 263**

Step 3

- Calculate Slope **m**:

$$m = \frac{N\,\Sigma(xy) - \Sigma x\,\Sigma y}{N\,\Sigma(x^2) - (\Sigma x)^2}$$

$$= \frac{5 \times 263 - 26 \times 41}{5 \times 168 - 262}$$

$$= \frac{1315 - 1066}{840 - 676}$$

$$= \frac{249}{164}$$

$$= 1.5183...$$

Step 4

- Calculate Intercept **b**:

$$b = \frac{\Sigma y - m\,\Sigma x}{N}$$

$$= \frac{41 - 1.5183 \times 26}{5}$$

$$= 0.3049\ldots$$

Step 5

- Assemble the equation of a line:

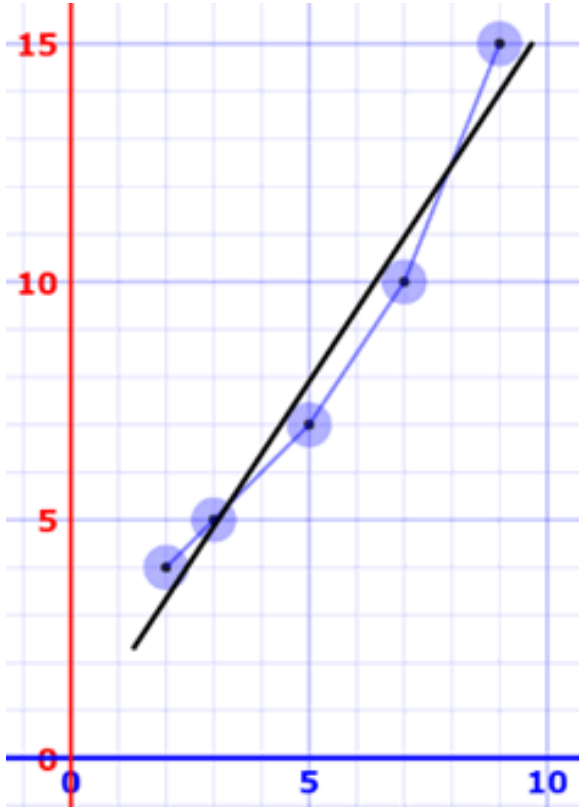
$$y = mx + b$$

$$y = 1.518x + 0.305$$

Let’s see how well it fits

x	y	y = 1.518x + 0.305	error
2	4	3.34	−0.66
3	5	4.86	−0.14
5	7	7.89	0.89
7	10	10.93	0.93
9	15	13.97	−1.03

Here are the (x,y) points and the line **y = 1.518x + 0.305** on a graph:



Let’s see a prediction

- Sam hears the weather forecast which says "we expect 8 hours of sun tomorrow", so he uses the above equation to estimate that he will sell
- $y = mx + b = 1.518 \times 8 + 0.305 = 12.45$ Ice Creams
- Sam makes fresh waffle cone mixture for 14 ice creams just in case.
- Yum.

Python – Linear Regression

- Constructor `LinearRegression()` from `sklearn.linear_model` module creates an Ordinary Least Squares object.

- The `fit()` function takes a 1xn matrix of predictors (independent vars).
  - If more than one independent variable and the predictors form a vector, slice with `numpy.newaxis` object to create another dimension.
  - After fitting, use `predict()` to calculate predicted values
  - Check the fit score using the function `score()`.

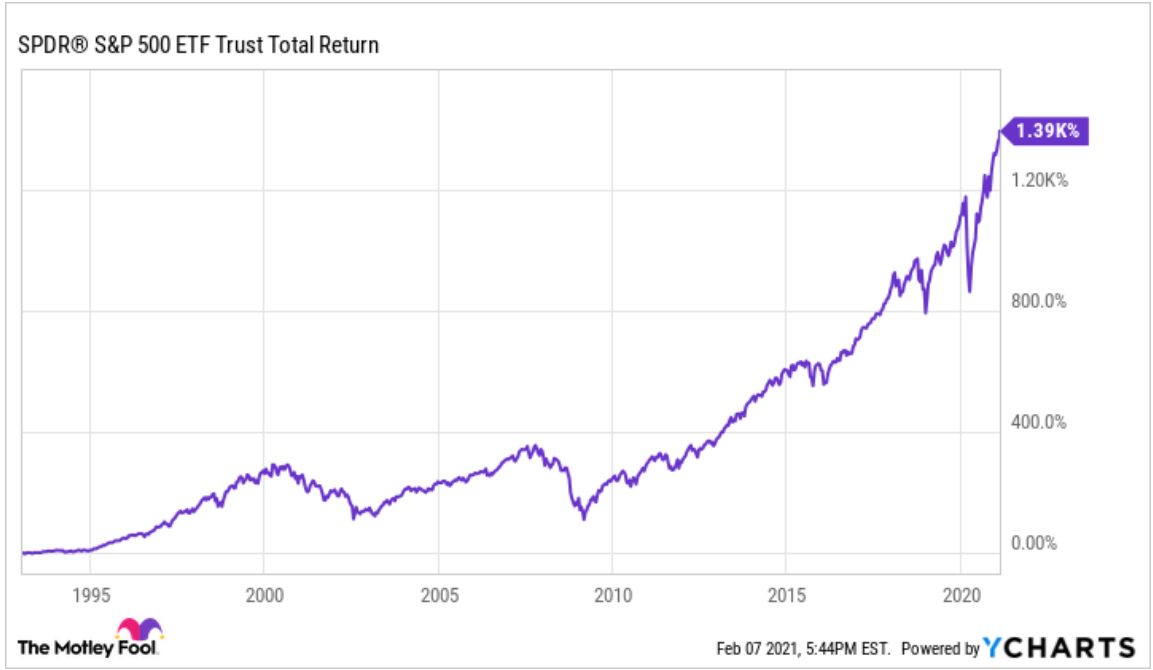
Simple OLS Regression Example

```
import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3

reg = LinearRegression().fit(X, y)

print(reg.score(X, y))
print(reg.coef_)
print(reg.intercept_)
print(reg.predict(np.array([[3, 5]])))
```

```
X is
[[1 1]
 [1 2]
 [2 2]
 [2 3]]
y is  [ 6  8  9 11]
Score:  1.0
Coef:  [1. 2.]
Intercept:  3.0000000000000018
Prediction:  [16.]
```

Example

- Use the historical S&P 500 closing prices from Yahoo! Finance to build, fit and evaluate a linear regression model.
- Data is saved in sapXXI.csv.



```
import numpy, pandas as pd
import matplotlib, matplotlib.pyplot as plt
import sklearn.linear_model as lm

# Get the data
sap = pd.read_csv("sapXXI.csv").set_index("Date")

# Select a "linearly looking" part
sap.index = pd.to_datetime(sap.index)
sap_linear = sap.loc[sap.index > pd.to_datetime('2009-01-01')]

# Prepare the model and fit it
olm = lm.LinearRegression()
X = numpy.array([x.toordinal() for x in  \
        sap_linear.index])[:, numpy.newaxis]
y = sap_linear['Close']
olm.fit(X, y)

# Predict values
yp = [olm.predict(x.toordinal())[0] for x in sap_linear.index]
```

```
# Evaluate the model
olm_score = olm.score(X, y)

# Select a nice plotting style
matplotlib.style.use("ggplot")

# Plot both data sets
X = numpy.array([x for x in sap_linear.index])
plt.plot(X, y)
plt.plot(X, yp)

# Add decorations
plt.title("OLS Regression")
plt.xlabel("Year")
plt.ylabel("S&P 500 (closing)")
plt.legend(["Actual", "Predicted"], loc="lower right")
plt.annotate("Score=%.3f" % olm_score, \
    xy=(pd.to_datetime('2010-06-01'), 1900))

plt.savefig("sap-linregr.pdf")
```

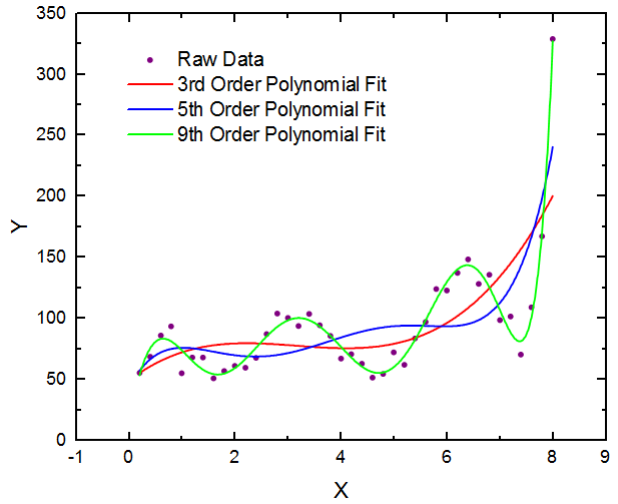
https://replit.com/@CSREPLIT/sapLinearRegression#main.py

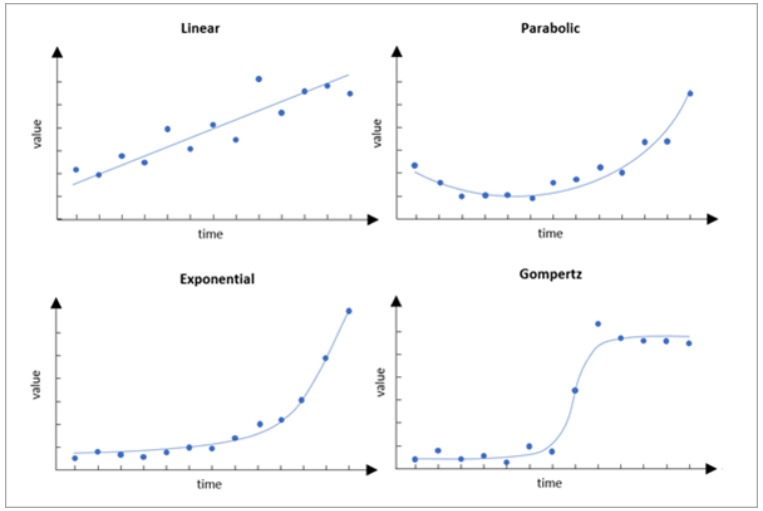
Results

- Unfortunately, SciKit-Learn does not calculate the p-value of the fit.

Curve Fitting

- Not all data is fitted well with a linear equation
- Some data have the shape of an polynomial, exponential or S-shaped curve…





- `scipy.optimize` module has a `curve_fit ()` function

curve_fit() parameters

- Fit for the parameters a, b, c of the function f (default bounds on a, b, and c are (-*inf*, *inf*) each)

```
popt, pcov = curve_fit(f, xdata, ydata)
print (popt)
→ array([2.56274217, 1.37268521, 0.47427475])
```

- Constrain the optimization to the region of 0 <= a <= 3, 0 <= b <= 1 and 0 <= c <= 0.5:

```
popt, pcov = curve_fit(f, xdata, ydata, bounds=([0, 0, 0], [3., 1., 0.5]))
print (popt)
→ array([2.43736712, 1.        , 0.34463856])
```

curve_fit()

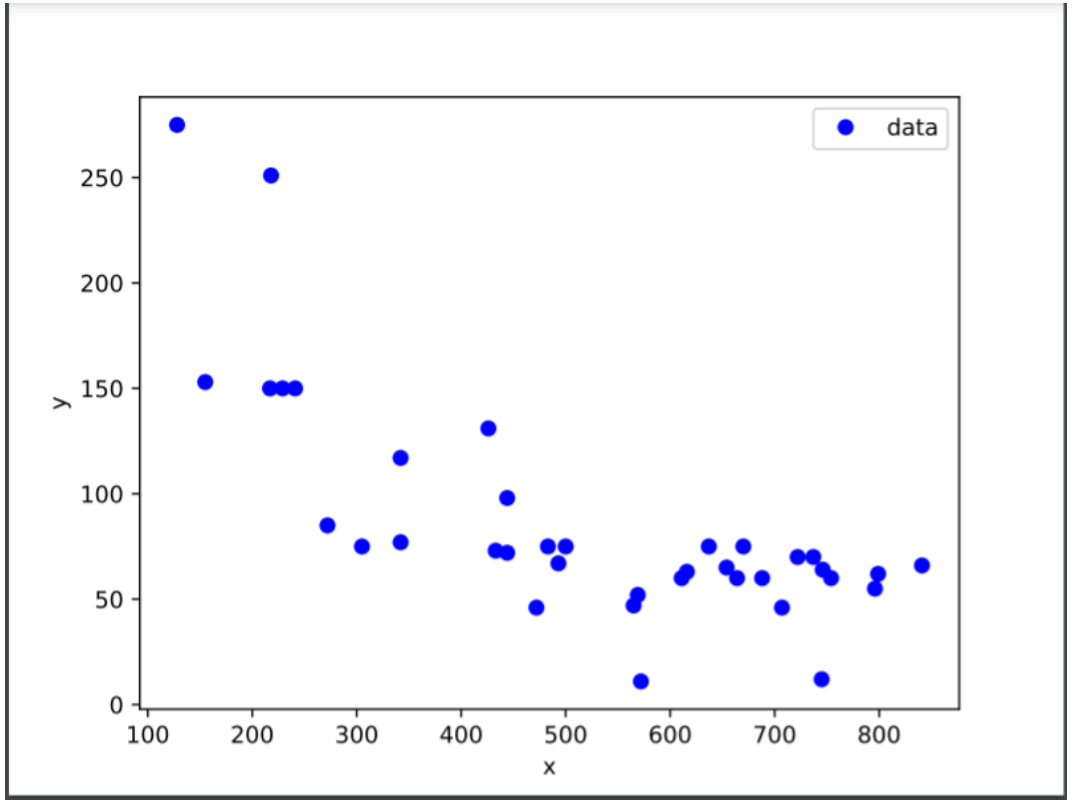
Returns

Popt *array* -
Optimal values for the parameters so that the sum of the squared residuals of `f(xdata, *popt) - ydata` is minimized.

pcov *2-D array*
The estimated covariance of `popt`. The diagonals provide the variance of the parameter estimate.

Estimate parameter a

- a – usually corresponds to asymptote
- Asymptote is the value that y approaches as x goes to infinity.
- What would a good guess be for an asymptote in the plot?

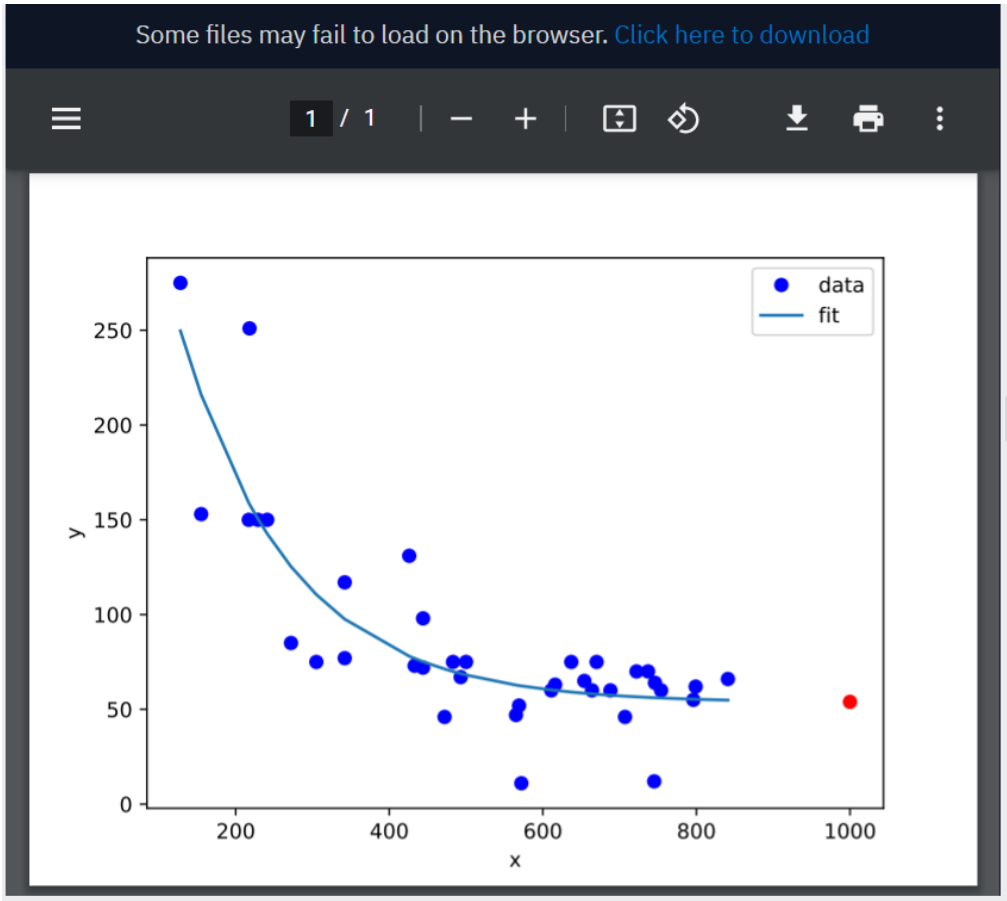


```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

#get the data and do an indirect sort on the
indices of the data along the x axis
my_data = np.genfromtxt('CurveData.csv',\
                        delimiter=',')
my_data = my_data[my_data[:,0].argsort()]

#make it 1xN data instead of Nx1
xdata = my_data[:,0].transpose()
ydata = my_data[:,1].transpose()

# define an exponential function for fitting
def func(x, a, b, c, d):
  return a * np.exp(-b * (x - c)) + d

#make initial guesses for a, b, c, d
init_vals = [50, 0, 90, 63]
```

https://replit.com/@CSREPLIT/sapLinearRegression#main.py

```
# fit your data and get fit parameters
popt, pcov = curve_fit(func, xdata, ydata,\
  p0=init_vals, bounds=([0, 0, 90, 0], \
                        [1000, 0.1, 200, 200]))

# predict new data for some x based on the fit
x_guess = 1000
y_pred = func(x_guess, *popt)
print("Prediction: (", x_guess, ',', y_pred, ")")

#plot the data
plt.plot(xdata, ydata, 'bo', label='data')
plt.plot(xdata, func(xdata, *popt), '-',\
        label='fit')

#plot the prediction
plt.scatter ([x_guess], [y_pred], color = 'red')

plt.xlabel('x')
plt.ylabel('y')
plt.legend()

plt.savefig("Curvefig.pdf")
```

Results of curve fitting / Predicting




```
Matplotlib created a temporary config/cache dire
at /tmp/matplotlib-ga8ghohg because the default path (
/config/matplotlib) is not a writable directory; it is
 highly recommended to set the MPLCONFIGDIR environmen
t variable to a writable directory, in particular to s
peed up the import of Matplotlib and to better support
 multiprocessing.
Prediction: ( 1000 , 53.94363024719651 )
```