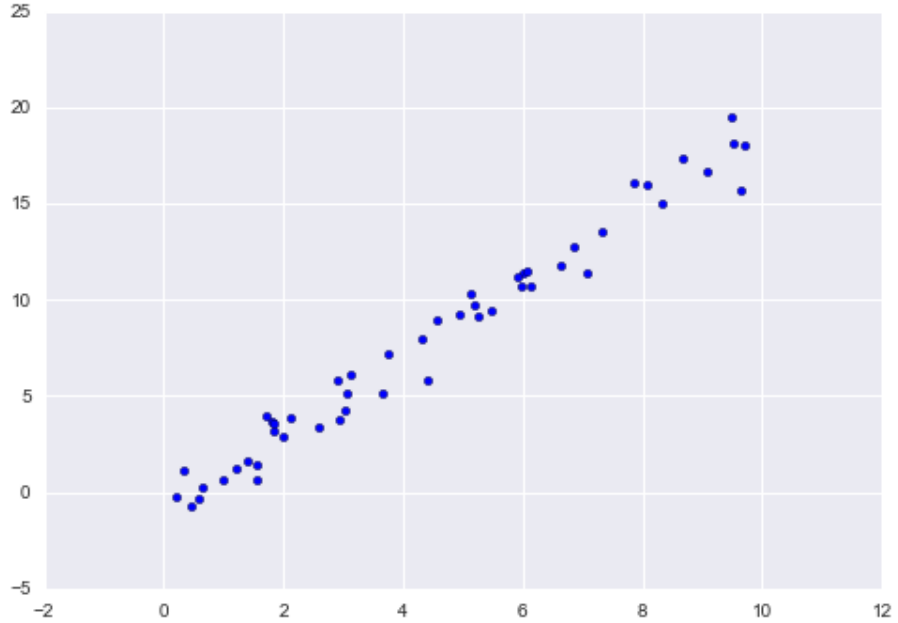
Linear Regression

In regression we predict values rather than discrete labels.



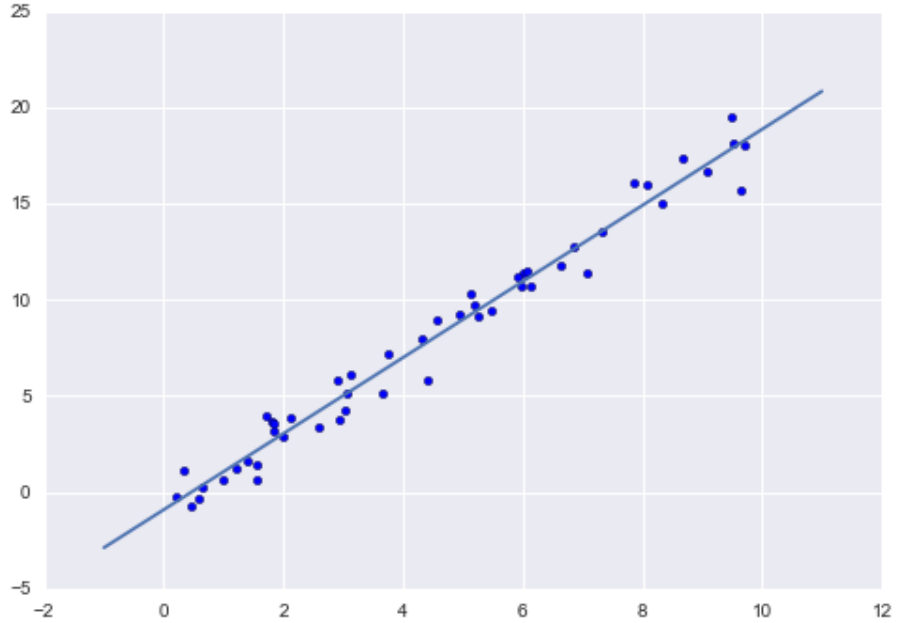
Linear Regression

In the simplest case we want to fit a line through the points.

A straight-line fit is a model of the form

$$y=ax+b$$

where a is commonly known as the slope, and b is commonly known as the intercept.



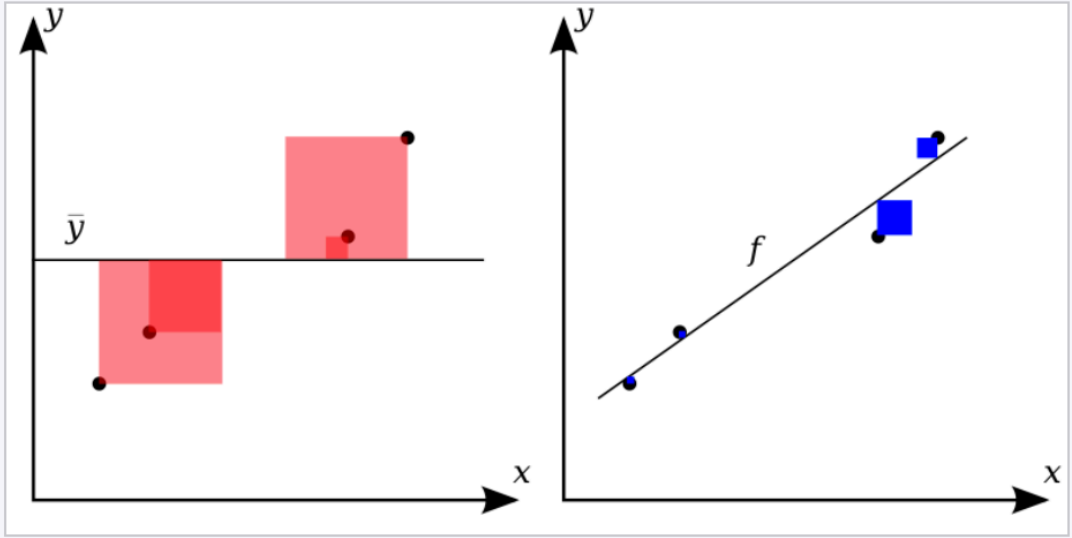
Here: b = -5 , a =2

Linear Regression

```
# generate the data
import matplotlib.pyplot as plt
import random
import pandas
x = pandas.DataFrame([10 * random.random() for __ in range(50)])
y = 2 * x - 1 + pandas.DataFrame([random.random() for __ in range(50)])
plt.scatter(x, y)
plt.show()

# pick model
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
model.fit(x, y)
print("Model slope:    ", model.coef_[0])
print("Model intercept:", model.intercept_)

# plot the model together with the data
xfit = pandas.DataFrame([i for i in range(-1, 12)])
yfit = model.predict(xfit)
plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.show()

# compute the R^2 score – good models: R^2 ~ 1
print("R^2 score: {}".format(model.score(x,y)))
```

```
Model slope:     [ 2.0306547]
Model intercept: [-0.64847945]
R^2 score: 0.9974134813061953
```

R^2 Score



$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

The better the linear regression (on the right) fits the data in comparison to the simple average (on the left graph), the closer the value of R^2 is to 1. The areas of the blue squares represent the squared residuals with respect to the linear regression. The areas of the red squares represent the squared residuals with respect to the average value.

Source: https://en.wikipedia.org/wiki/Coefficient_of_determination

Non-linear Regression

- Neural networks, trees, and knn can also be used for regression.
- Because of their underlying modeling algorithms they can not only model linear data sets but also non-linear ones.

Regression Trees

Trees can also be used as regression trees:

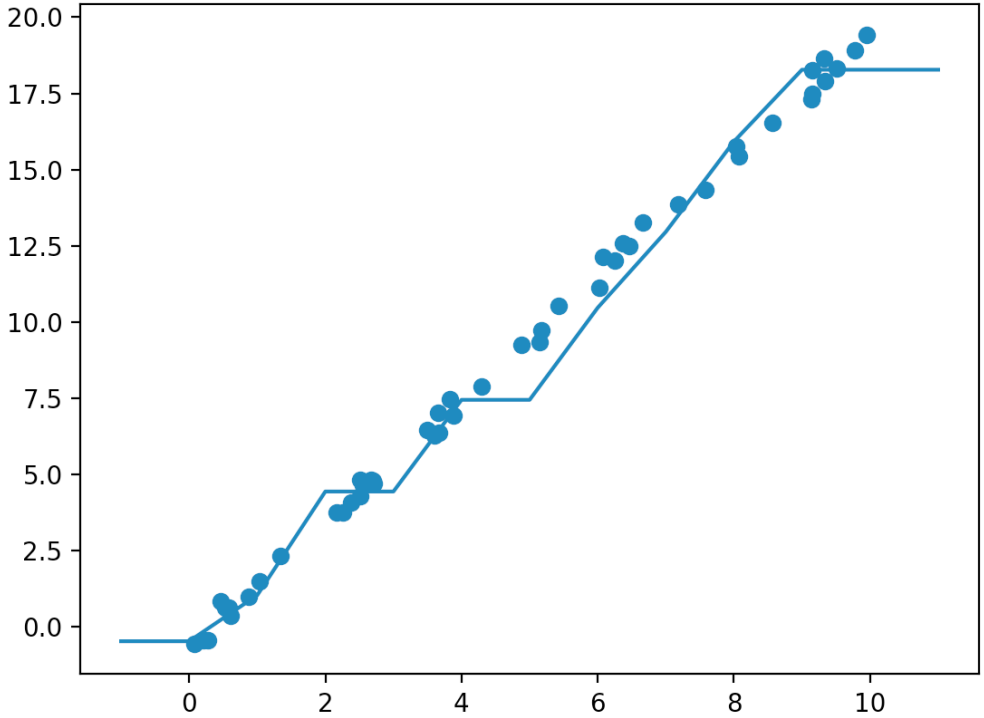
“piecewise linear regression”

Regression Trees

```
# generate the data
import matplotlib.pyplot as plt
import random
import pandas
x = pandas.DataFrame([10 * random.random() for __ in range(50)])
y = 2 * x - 1 + pandas.DataFrame([random.random() for __ in range(50)])

# pick model
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(max_depth=3)
model.fit(x, y)

# plot the model together with the data
xfit = pandas.DataFrame([i for i in range(-1, 12)])
yfit = model.predict(xfit)
plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.show()

# compute the R^2 score – good models: R^2 ~ 1
print("R^2 score: {}".format(model.score(x,y)))
```

tree_regression.py

Regression Trees



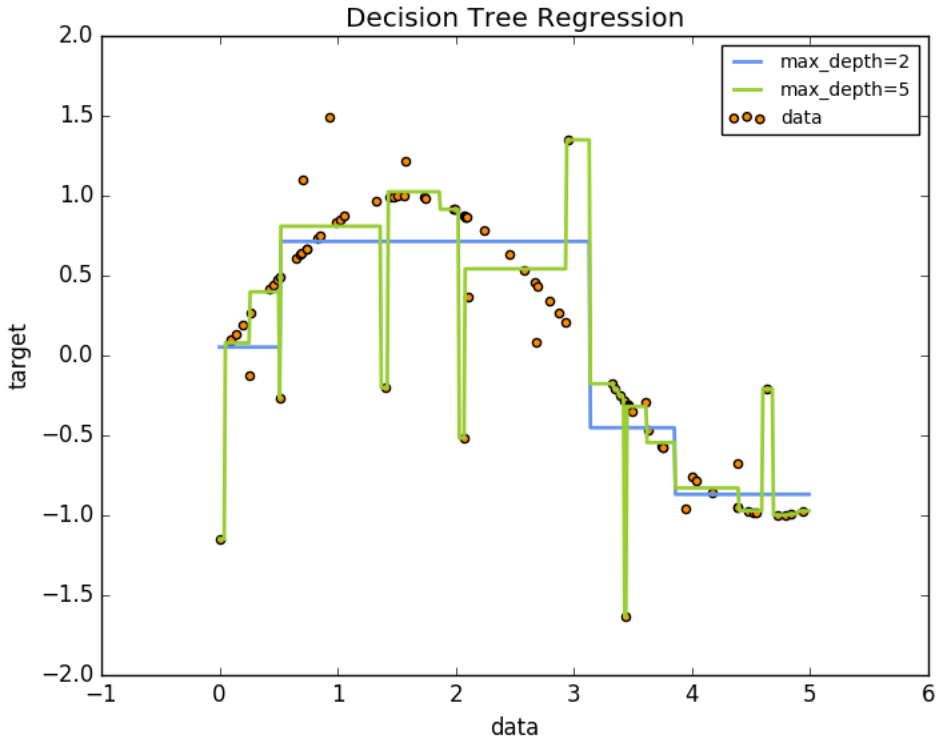
R^2 score: 0.9867856696126454

Regression Trees

Even with regression trees there is the possibility of *overfitting*.

Consider the noisy sine curve…

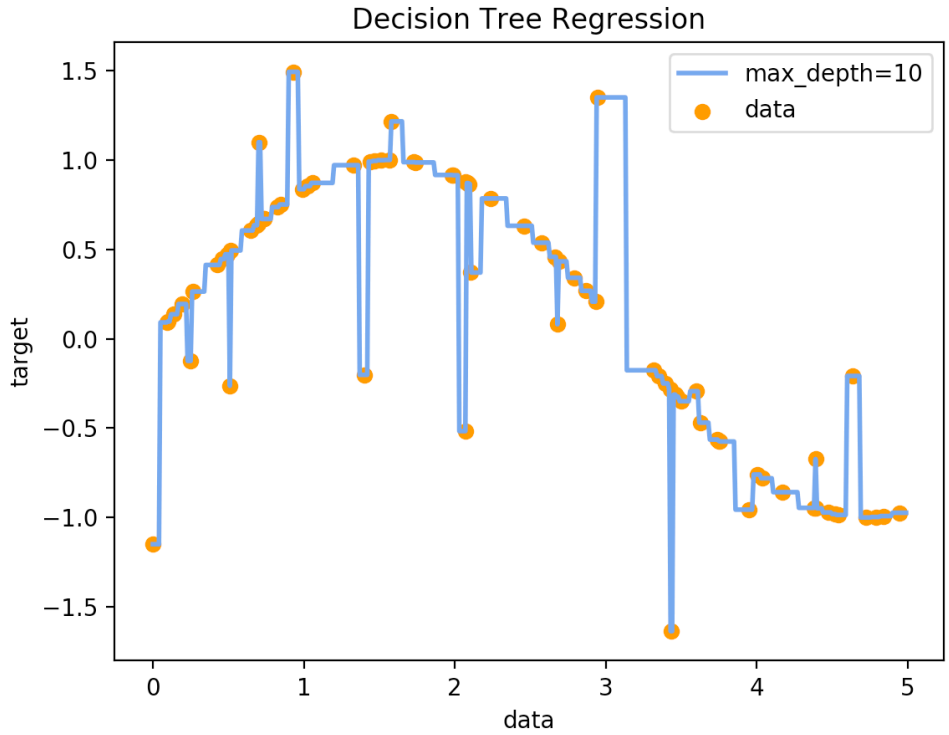
Demo - tree_regression2.py



Regression Trees

With max_depth=10 the tree memorizes the data set – noise and all.

Just like in classification, when evaluating a model on the training set it is always possible to build a model with a perfect score.



R^2= 1.0

Regression Trees

Perform a gridsearch over tree depth

The gridsearch will automatically adjust to use the R^2 score function

Demo - tree_regr_grid.py

```
...
# setting up the grid search
model = model = DecisionTreeRegressor()
param_grid = {'max_depth': list(range(1,25))}
grid = GridSearchCV(model, param_grid, cv=5)

# performing grid search
grid.fit(X,y)

# print out what we found
print("Best parameters: {}".format(grid.best_params_))
...
```

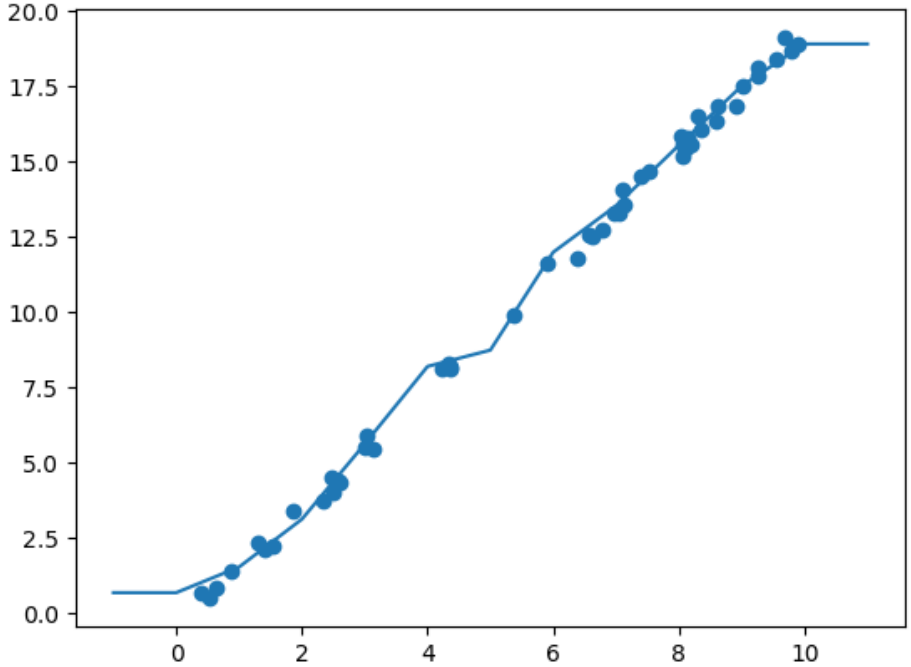
KNN Regression

We can use the k-Nearest Neighbor algorithm to do regression

knn_regression.py

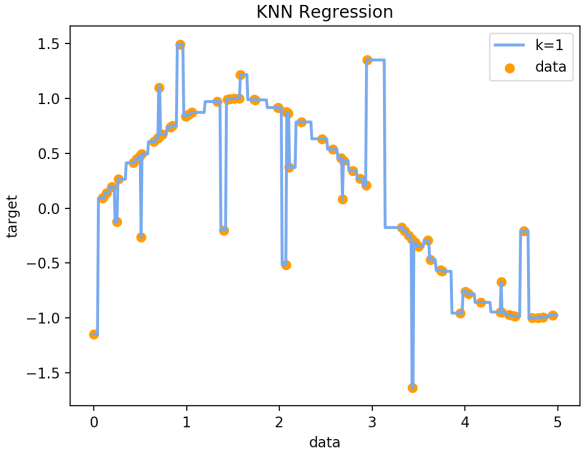
```
# generate the data
import matplotlib.pyplot as plt
import random
import pandas
x = pandas.DataFrame([10 * random.random() for __ in range(50)])
y = 2 * x - 1 + pandas.DataFrame([random.random() for __ in range(50)])

# pick model
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=3)
model.fit(x, y)

# compute the R^2 score
print("R^2 score: {}".format(model.score(x,y)))

# plot the model together with the data
xfit = pandas.DataFrame([i for i in range(-1, 12)])
yfit = model.predict(xfit)
plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.show()
```

KNN Regression

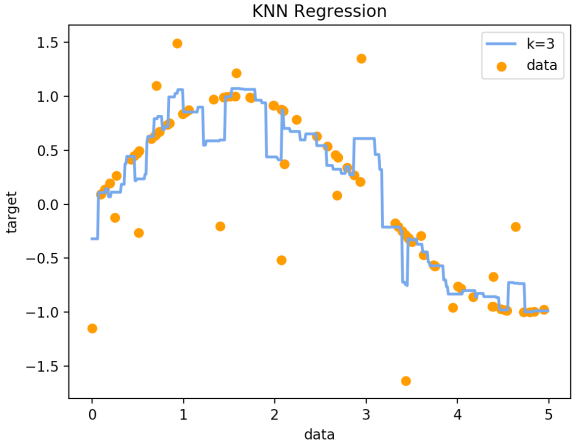


R^2 score: 0.9982075337874717

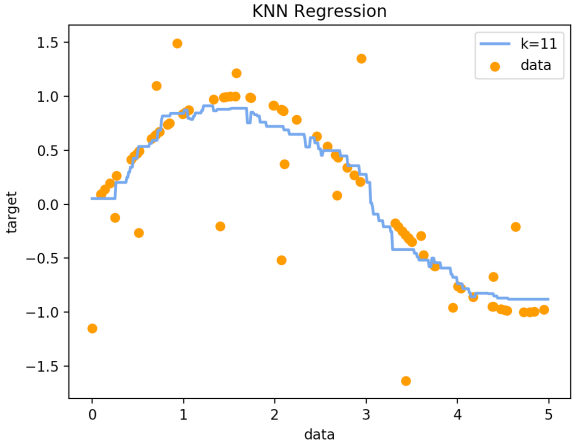
KNN Regression



R^2 = 1.0



R^2 = 0.841



R^2 = 0.775

knn_regression2.py

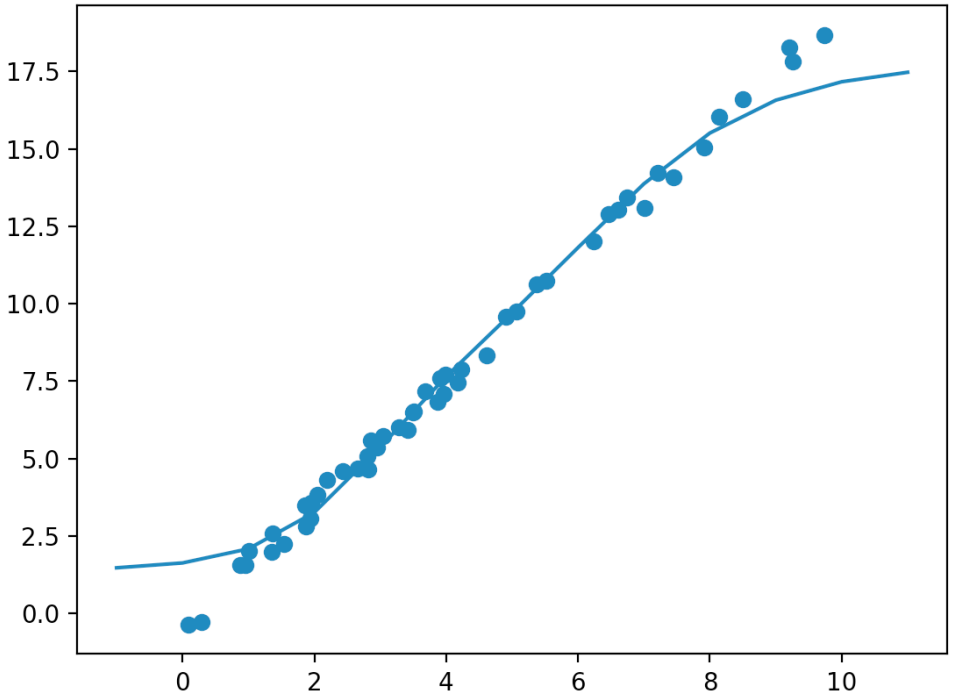
MLP Regression

We can use artificial neural networks to do regression

mlp_regression.py

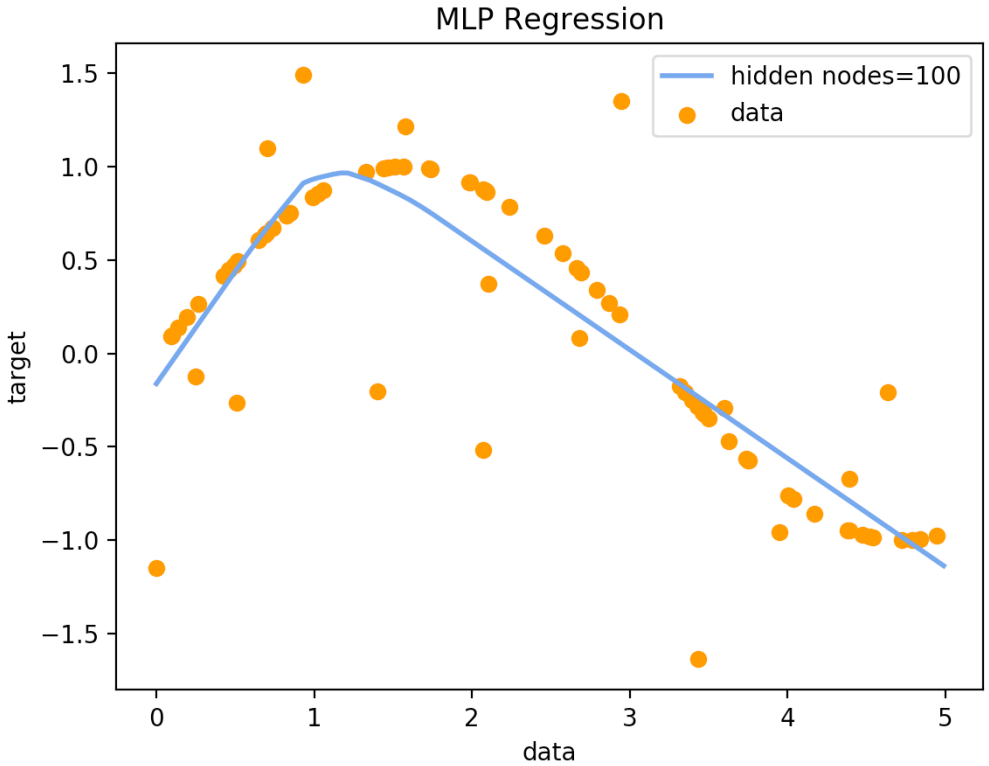
```
# generate the data
import matplotlib.pyplot as plt
import random
import pandas

n  = int(input('Choose a number for hidden nodes: '))

X = pandas.DataFrame([10 * random.random() for __ in range(50)])
y = 2 * X - 1 + pandas.DataFrame([random.random() for __ in range(50)])

# pick model
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(hidden_layer_sizes=(n,),
                     activation='logistic', max_iter=10000)
model.fit(X, y)

# compute the R^2 score
print("R^2 score: {}".format(model.score(X,y)))

# plot the model together with the data
Xfit = pandas.DataFrame([i for i in range(-1, 12)])
yfit = model.predict(Xfit)
plt.scatter(X, y)
plt.plot(Xfit, yfit)
plt.show()
```

MLP Regression



n = 3
R^2 score: 0.984

MLP Regression



R^2 score: 0.754

mlp_regression2.py

Exercise

Use the ‘cars’ data set from:

https://vincentarelbundock.github.io/Rdatasets/datasets.html

Drop the first column, target column is ‘dist’

Which one performs best on this data set: linear, mlp, tree, or knn regression? That is, which one gets the best cross-validated R^2 score?