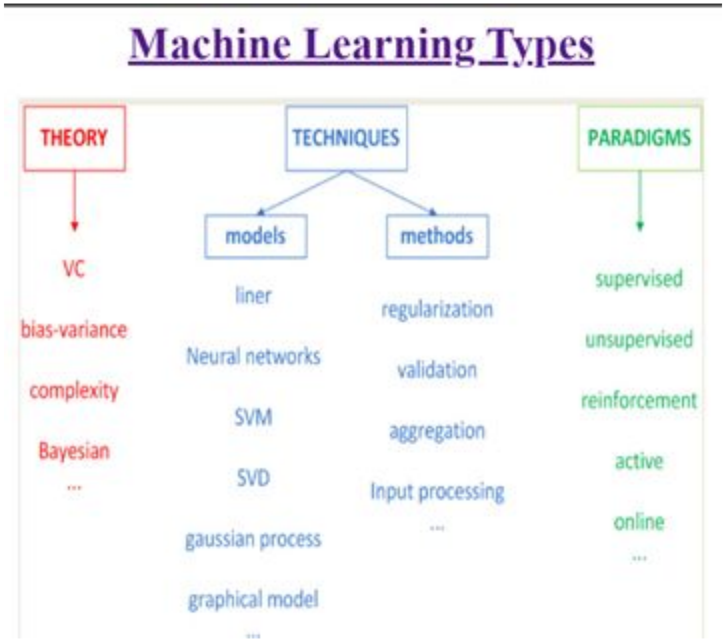
MACHINE LEARNING
Linear Regression using Normal

Fatema Nagori 19635

Table of Content:

INTRODUCTION

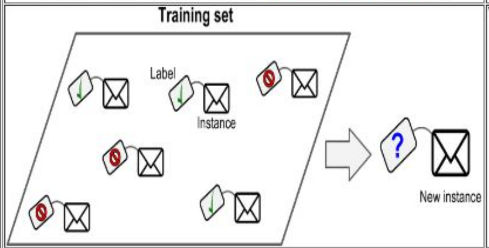
Machine learning is used in internet search engines, email filters to sort out spam, websites to make personalised recommendations, banking software to detect unusual transactions, and lots of apps on our phones such as voice recognition.

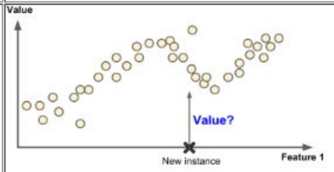


TYPES OF Machine Learning

Supervised Learning

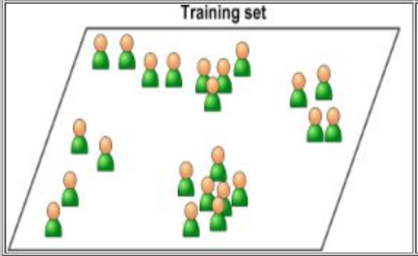
Need a lot of labeled data	
Classification: A labeled training set for supervised learning - predicate classes	Regression - predicate values

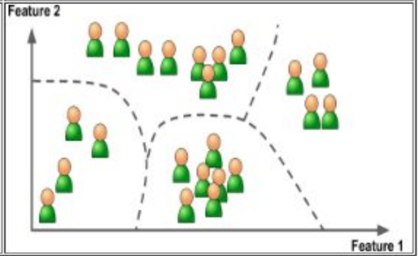




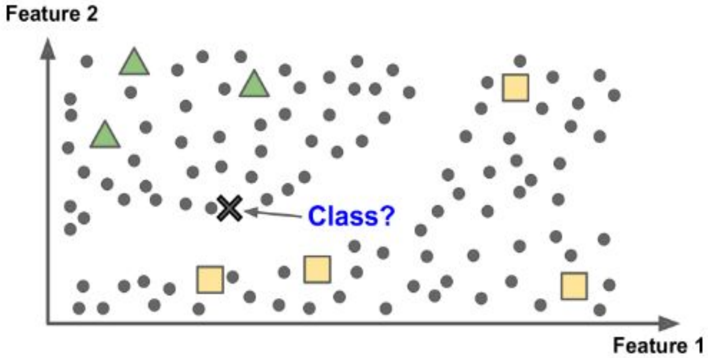
Unsupervised Learning

Data labeling is not required	
An unlabeled training set for unsupervised learning	Clustering



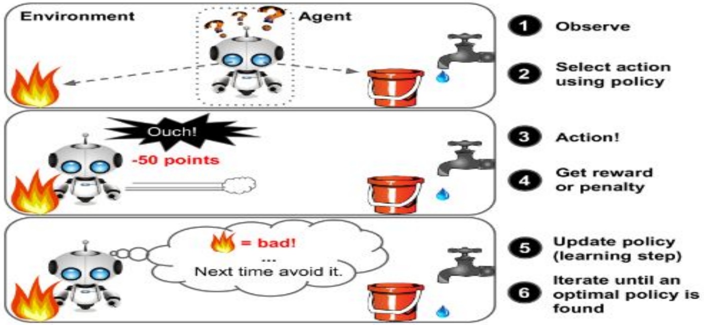


Semisupervised Learning



Reinforcement Learning

No need to have a lot of data



SUPERVISED LEARNING

- CLASSIFICATION
- REGRESSION

LINEAR REGRESSION

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog).

Simple linear regression uses traditional slope-intercept form, where m and b are the variables our algorithm will try to "learn" to produce the most accurate predictions. x represents our input data and y represents our prediction.

$$y=mx+b$$

Design

- Project will be done using Google colab
- Library used: sklearn, numpy, matplotlib, pandas
- Example code
- Modify the example code to make it work

IMPLEMENTATION

STEP 1

Original code:

```
import numpy as np

X = 2 * np.random.rand(100, 1)

y = 4 + 3 * X + np.random.randn(100, 1)
```

IMPLEMENTATION

Step 2

Modify the code:

```python
import numpy as np
import pandas as pd

# X = 2 * np.random.rand(100, 1)
# y = 4 + 3 * X + np.random.randn(100, 1)
from google.colab import files
uploaded  = files.upload()

import io
abalone = pd.read_csv(
    io.BytesIO(uploaded['abalone_train.csv']),
    names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
           "Viscera weight", "Shell weight", "Age"])
# X1 is
#    0       0.435
#    1       0.585
#    2       0.655
#    .....
X1 = abalone["Length"]

# X2 is
#    array([0.435, 0.585, ...., 0.45])
X2 = np.array(X1)

# X is
#    array([[0.435],
#           [0.585],
#           [0.655],
#           ...,A
#           [0.53 ],
#           [0.395],
#           [0.45 ]])
X = X2.reshape(-1, 1)

y1 = abalone["Height"]
y2 = np.array(y1)
y = y2.reshape(len(y2), 1)
```

TEST

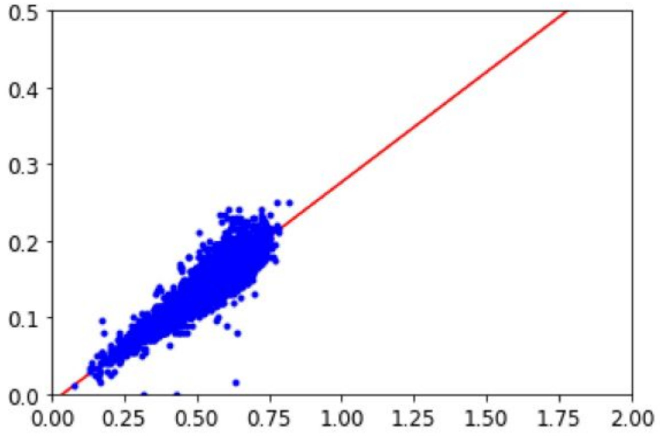
Use the give data

[abalone_train.csv]

```
import io
abalone = pd.read_csv(
    io.BytesIO(uploaded['abalone_train.csv']),
    names=["Length", "Diameter", "Height", "Whole weight", "Shucked weight",
           "Viscera weight", "Shell weight", "Age"])
```

TEST

In [108]:
```
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.axis([0, 2, 0, 0.5])
plt.show()
```

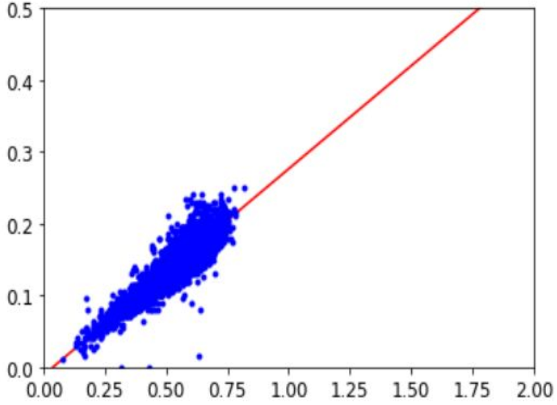


The figure in the book actually corresponds to the following code, with a legend and axis labels:

TEST

2...

```
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.axis([0, 2, 0, 0.5])
plt.show()
```



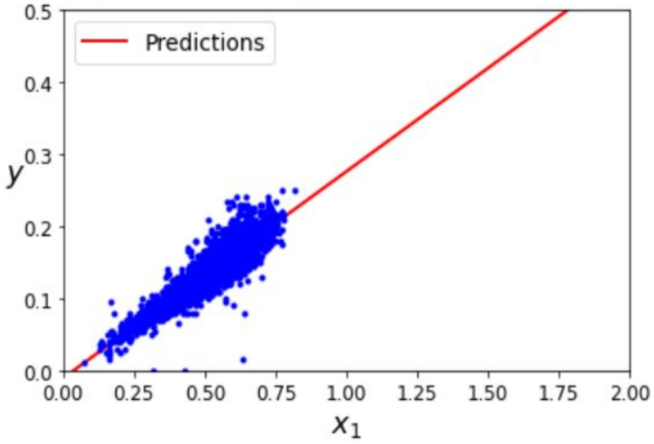
The figure in the book actually corresponds to the following code, with a legend and axis labels:

TEST

3...

```
plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 2, 0, 0.5])
save_fig("linear_model_predictions_plot")
plt.show()
```

```
Saving figure linear_model_predictions_plot
```



RESULT

And now let's measure the final model's accuracy on the test set:

In [180…

```
logits = X_test.dot(Theta)
Y_proba = softmax(logits)
y_predict = np.argmax(Y_proba, axis=1)

accuracy_score = np.mean(y_predict == y_test)
accuracy_score
```

Out[180…

```
0.9333333333333333
```

Our perfect model turns out to have slight imperfections. This variability is likely due to the very small size of the dataset: depending on how you sample the training set, validation set and the test set, you can get quite different results. Try changing the random seed and running the code again a few times, you will see that the results will vary.

In [180…

CONCLUSION

Python is very capable for training large dataset with its libraries. And it is easy to understand and use in practice.

ENHANCEMENT IDEAS:

There are also other ways to link to data, for example upload from the Google Drive or link the data from online resources.

To verify the Normal Equation, we can also use for example the sklearn library to get the linear regression formula and compare the result with the one from Normal Equation.

REFERENCE

LINKS:

SUPERVISED LEARNING

PROF HENRY’S MATERIAL

COLAB


THANKS FOR WATCHING