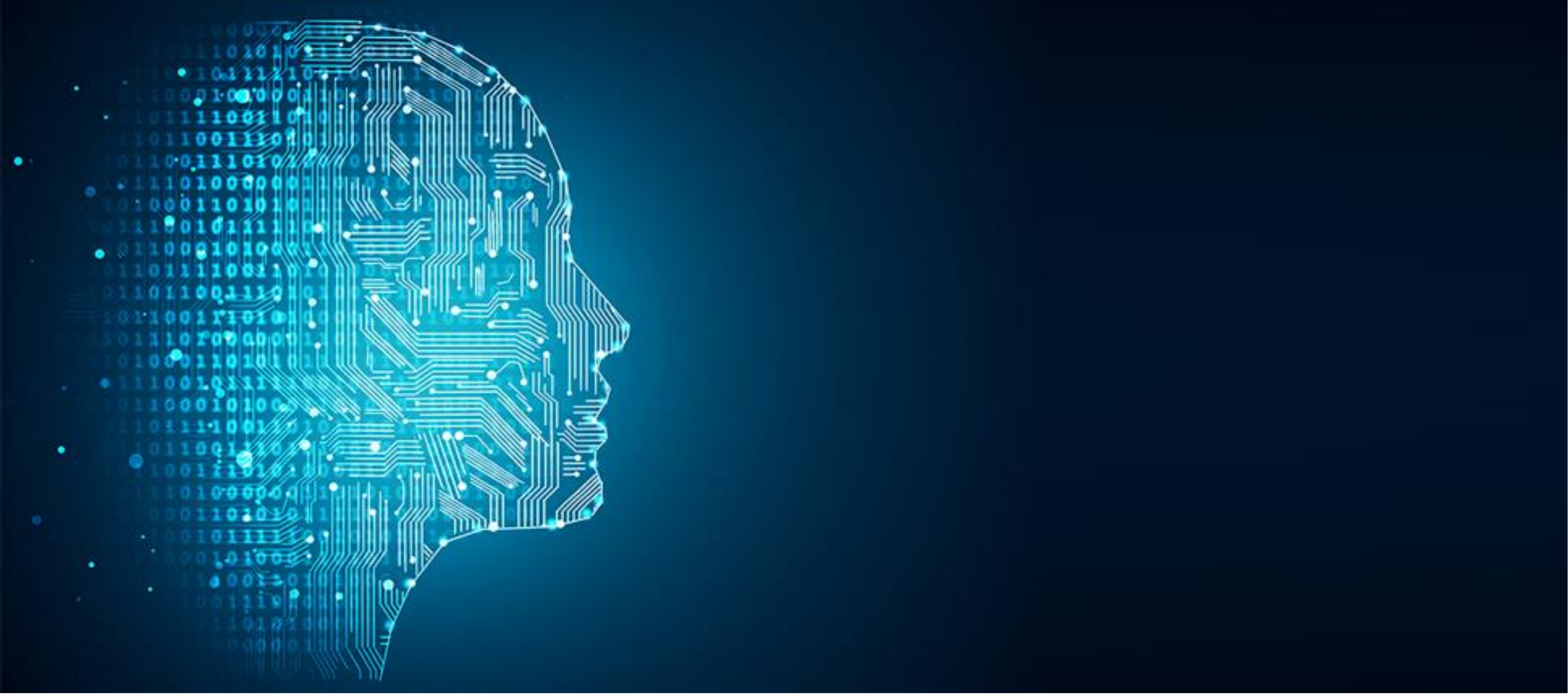
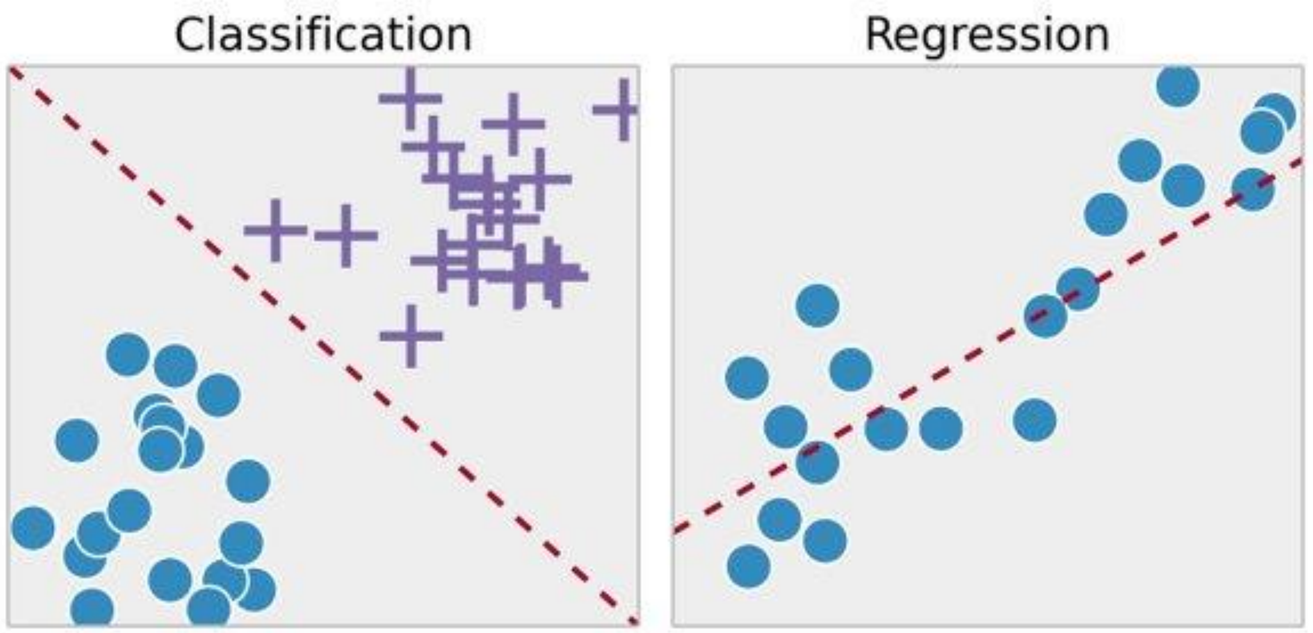


Concepts in Artificial Intelligence & Machine Learning Technologies

Regression and Classification

Designed by Dr Hu Wang
Adjusted by Dr. Wei Zhang

- **Regression V.S. Classification**



Linear Classification

• Regression --- task

1.3. EXAMPLES OF NON-CONVEX OPTIMIZATION PROBLEMS

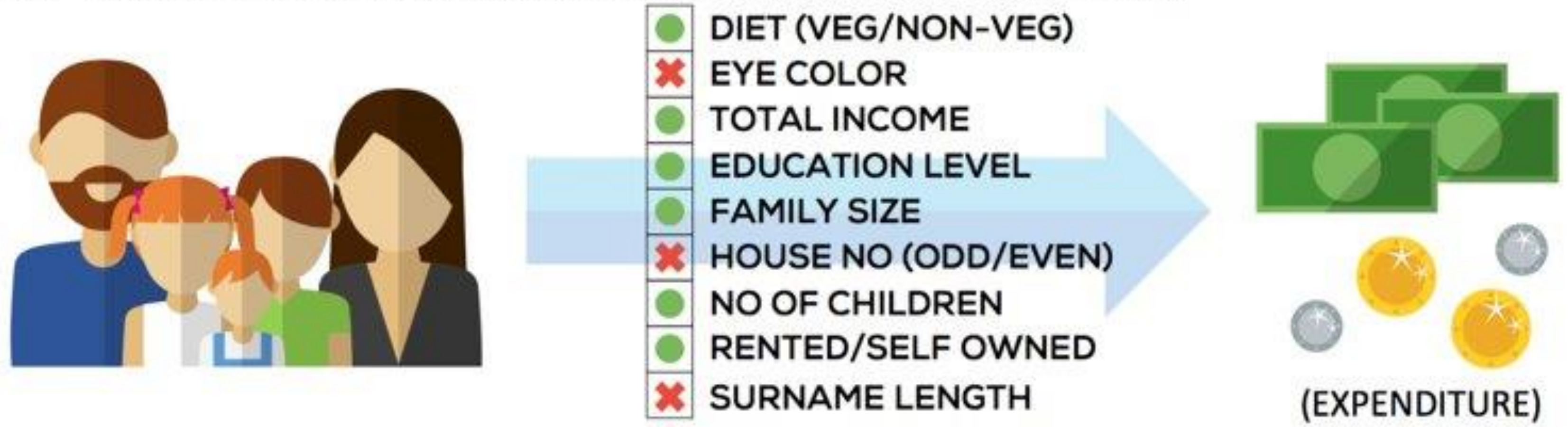
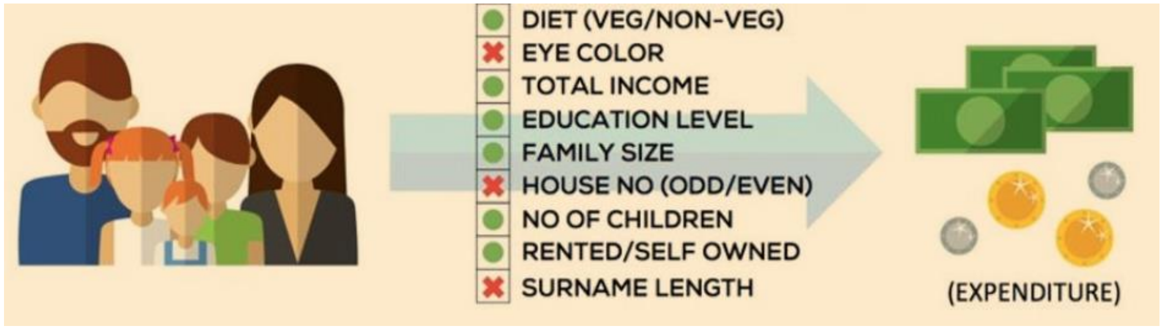
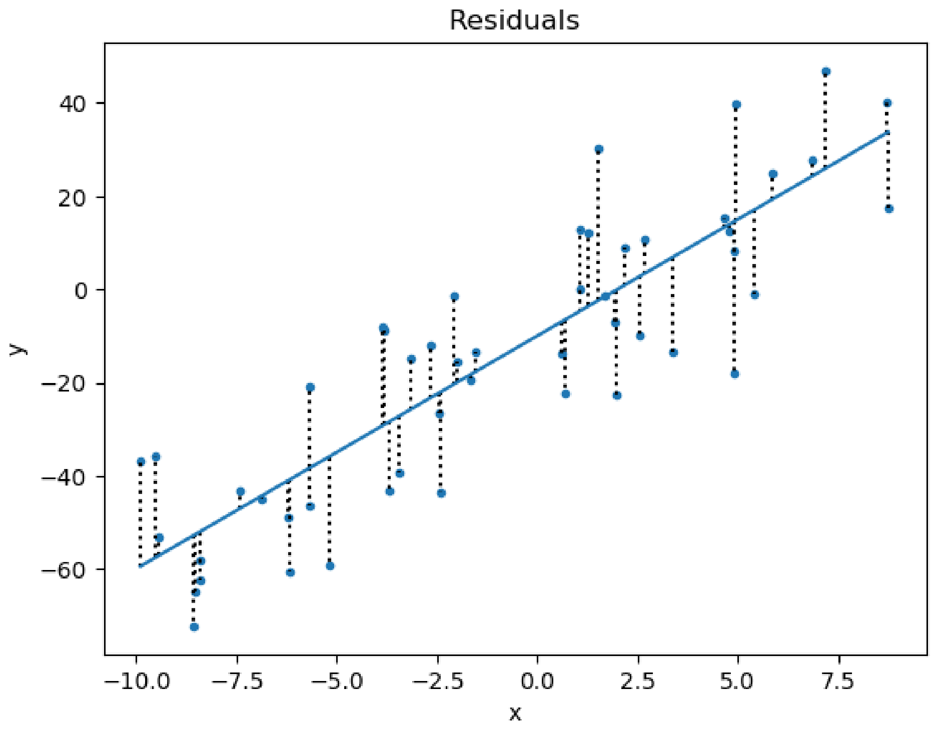


Figure 1.1: Not all available parameters and variables may be required for a prediction or learning task. Whereas the family size may significantly influence family expenditure, the eye color of family members does not directly or significantly influence it. Non-convex optimization techniques, such as sparse recovery, help discard irrelevant parameters and promote compact and accurate models.

https://arxiv.org/abs/1712.07897

- ## Regression


DIET (VEG/NON-VEG)
EYE COLOR
TOTAL INCOME
EDUCATION LEVEL
FAMILY SIZE
HOUSE NO (ODD/EVEN)
NO OF CHILDREN
RENTED/SELF OWNED
SURNAME LENGTH
(EXPENDITURE)


Residuals
y
x
40
20
0
−20
−40
−60
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

Loss Function

Loss Function

x_1 / x_2 / x_3 / x_4

$$\xrightarrow{\quad} \boxed{f_1} \xrightarrow{\quad}$$

Loss = 50%

"Dog"/"Dog"/"Dog"/"Dog"

x_1: y_1:"Cat"

x_2: y_2:"Cat"

x_3: y_3:"Dog"

x_4: y_4:"Dog"

Labelled Data

Loss Function

Machine is going to find the function with lowest loss

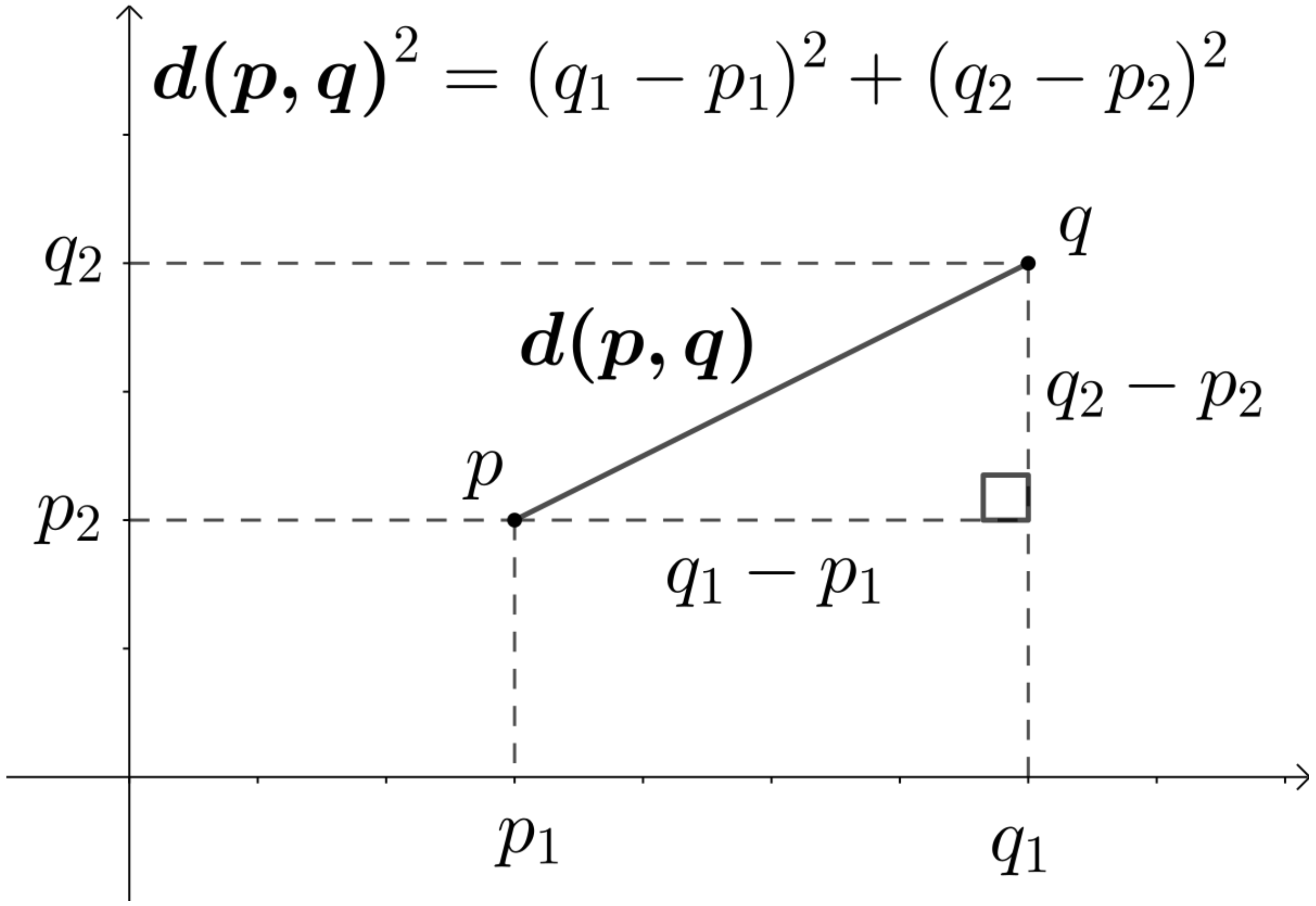
x_1 / x_2 / x_3 / x_4 $\rightarrow$ $\boxed{f_2}$ $\rightarrow$ "Cat"/"Cat"/"Dog"/"Dog"

Loss = 0%

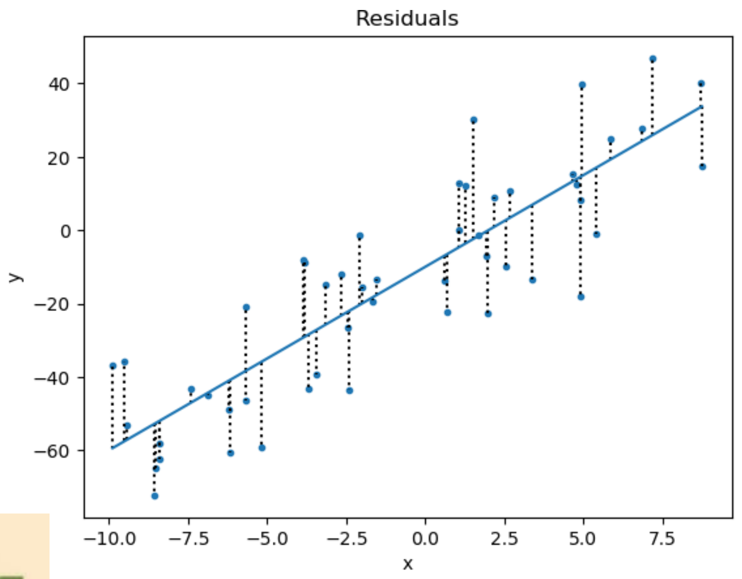
x_1:	y_1:"Cat"	x_2:	y_2:"Cat"
x_3:	y_3:"Dog"	x_4:	y_4:"Dog"

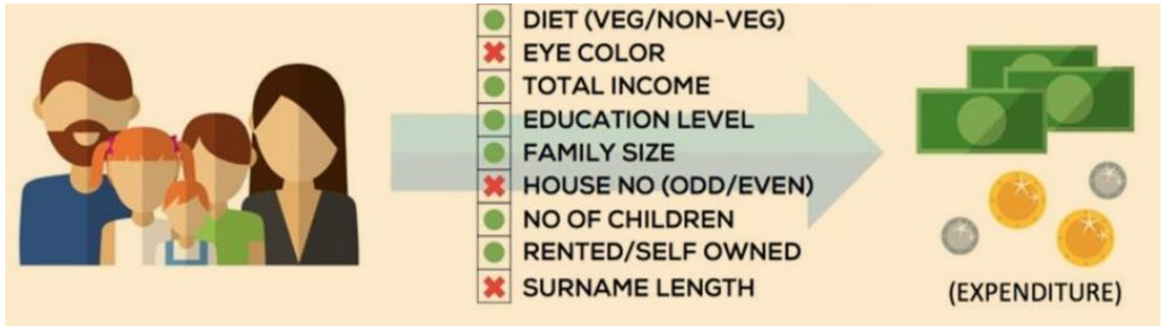
Labeled Data

- **L2 distance**


$\boldsymbol{d(p,q)}^2 = (q_1 - p_1)^2 + (q_2 - p_2)^2$
q
q_2
$\boldsymbol{d(p,q)}$
$q_2 - p_2$
p
p_2
$q_1 - p_1$
p_1
q_1

- ## Regression



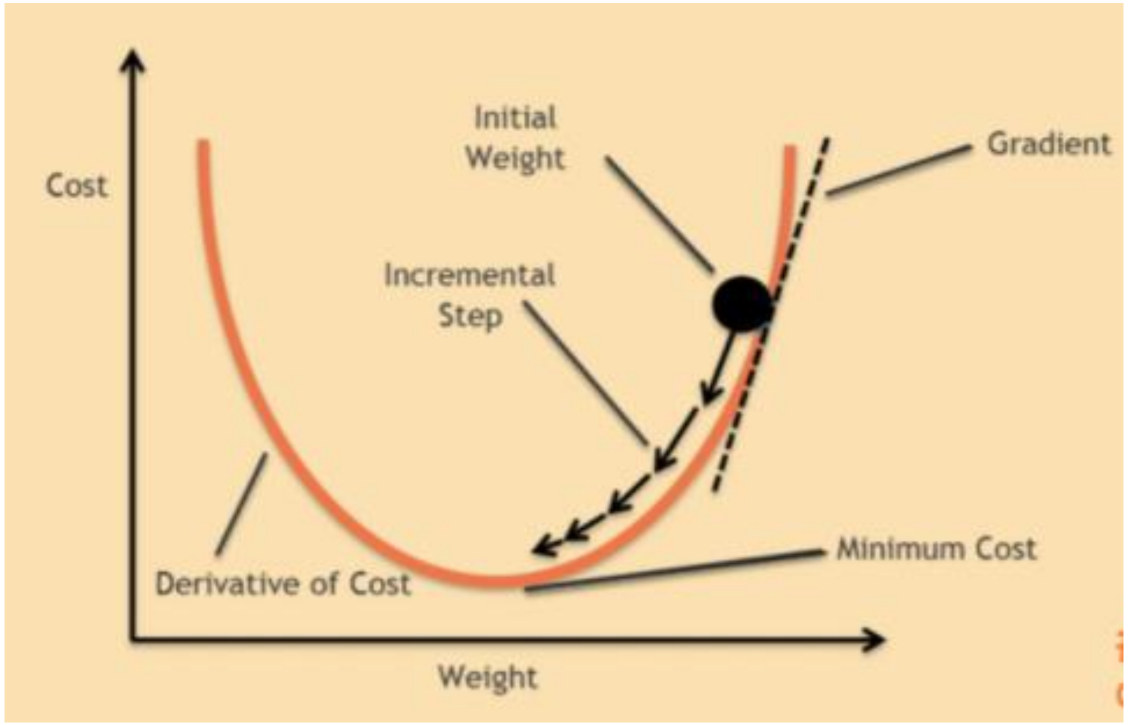


A popular way to recover $\mathbf{w}^*$ is using the *least squares* formulation

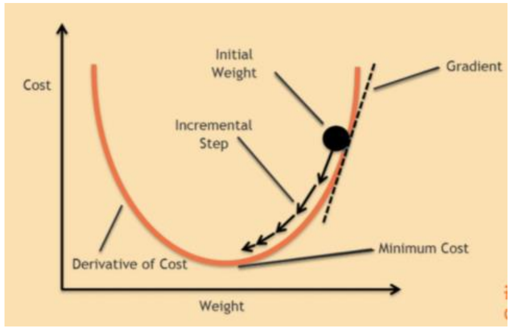
$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^\top \mathbf{w}\right)^2 .$$

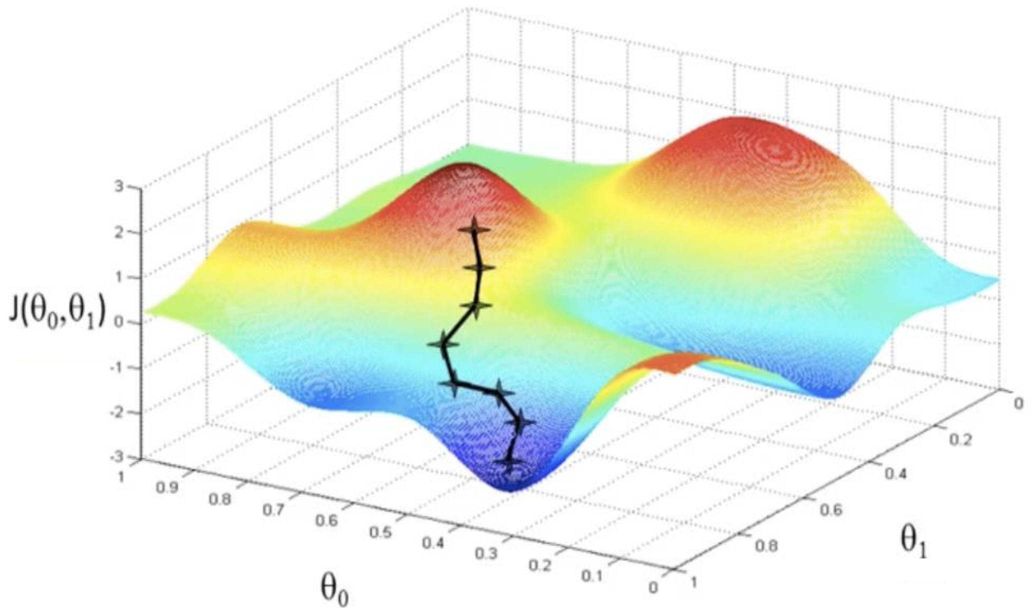
The linear regression problem as well as the least squares estimator, are extremely well studied and their behavior, precisely known.

- ## Optimization: Gradient Descent

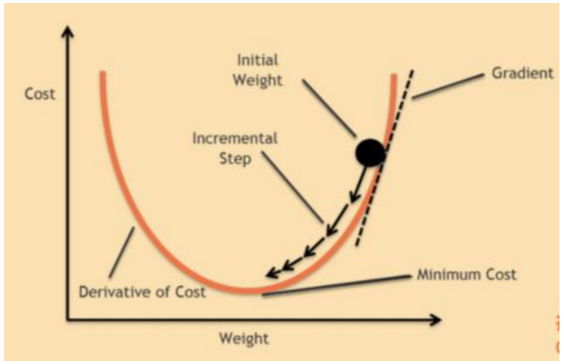


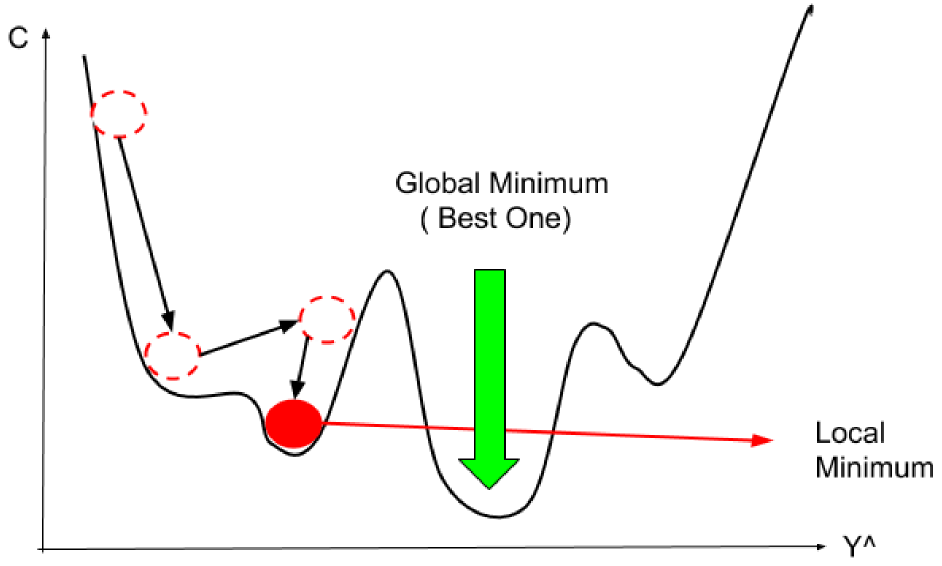
• Optimization: Gradient Descent



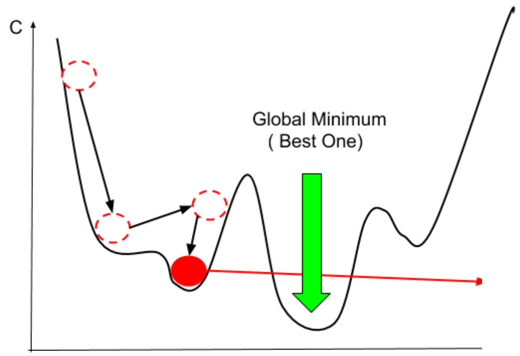


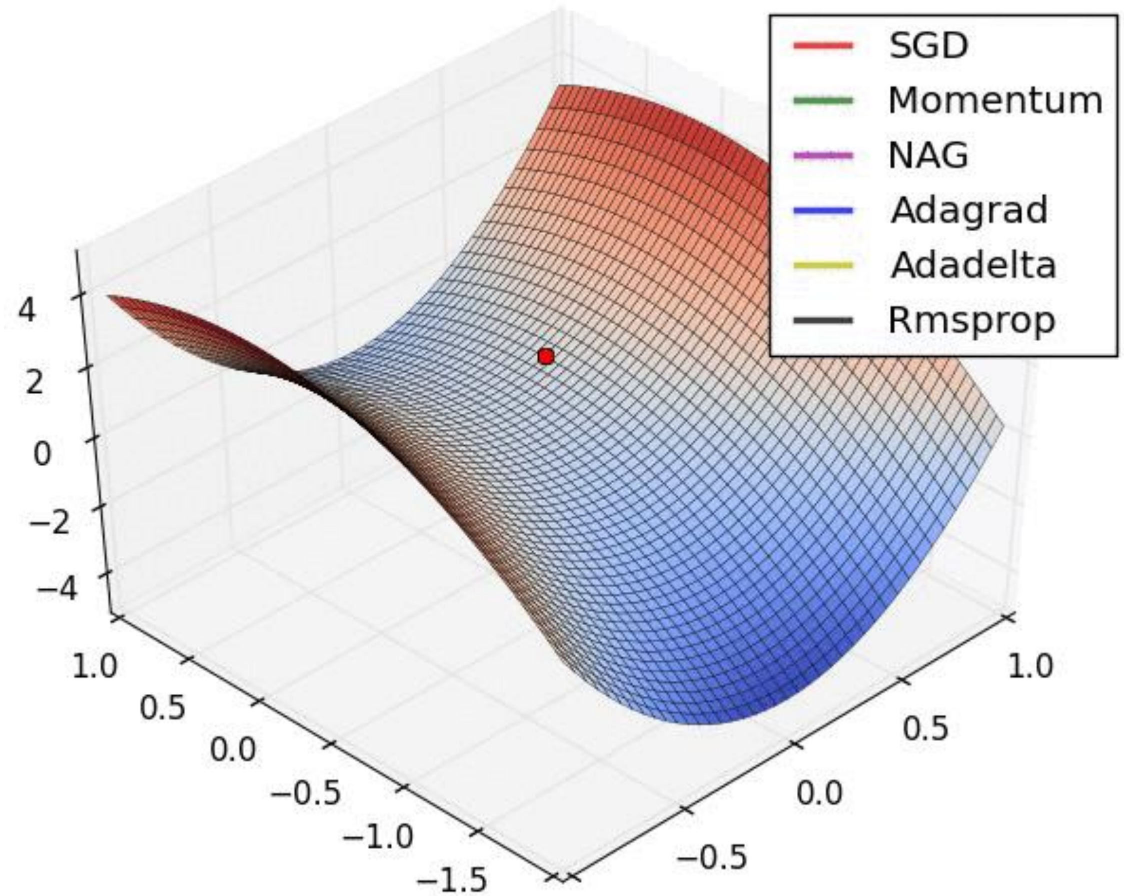
- ## Optimization



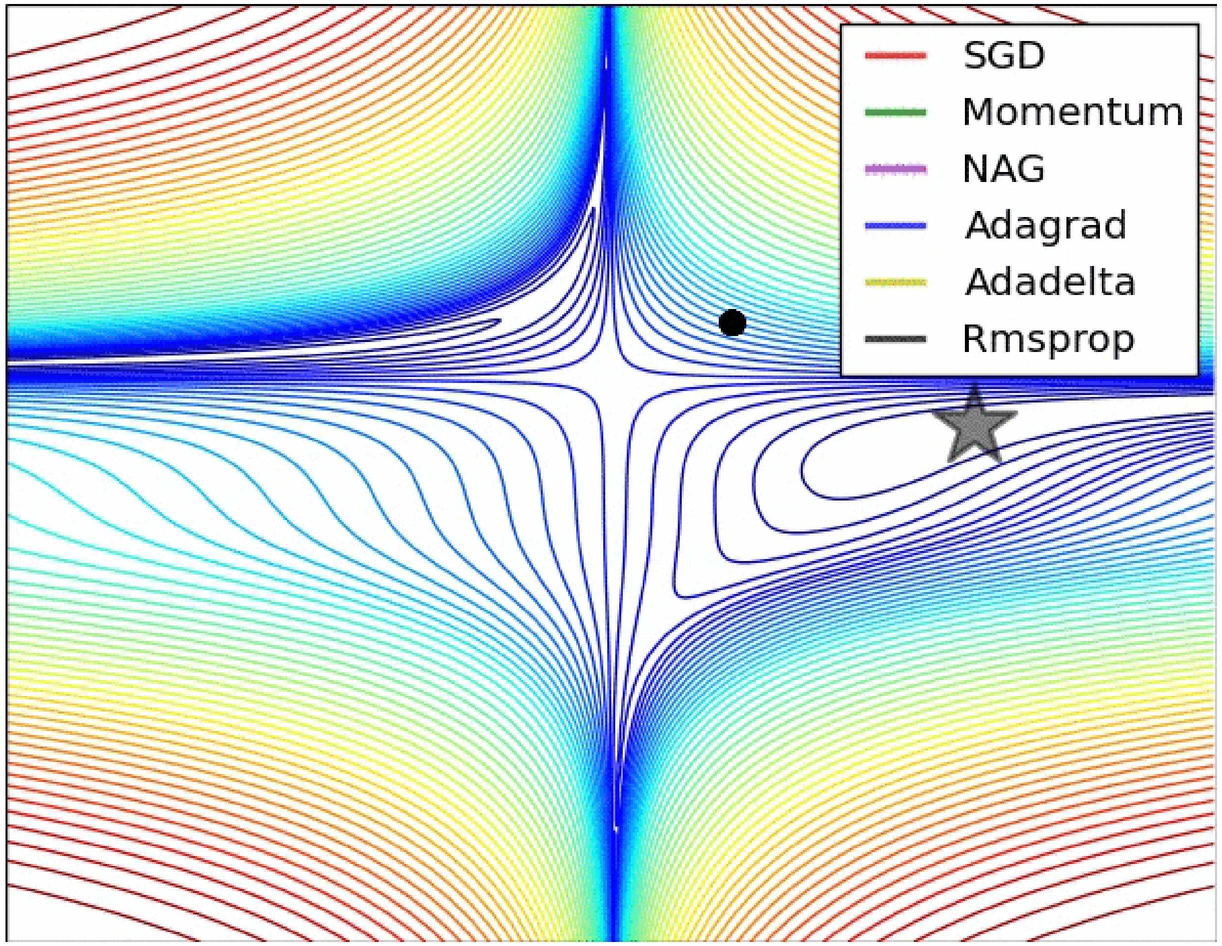


• Optimization



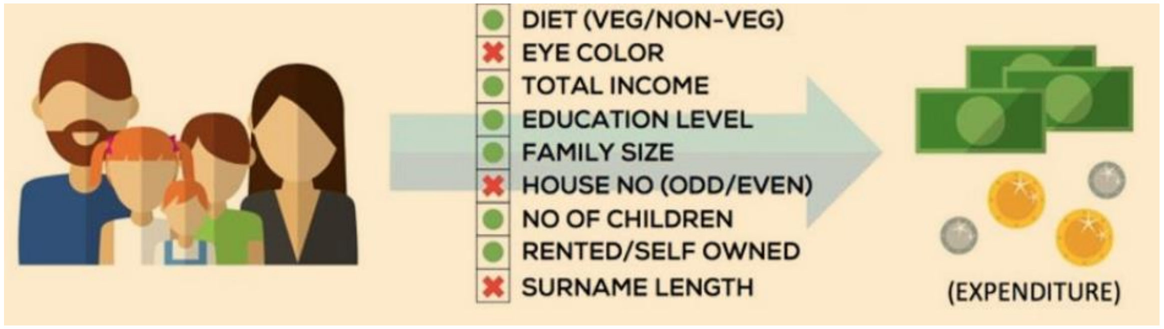


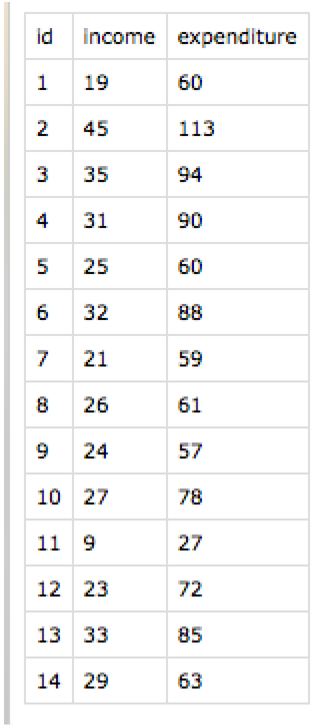
• Gradient Descent



Programming Example

• Regression





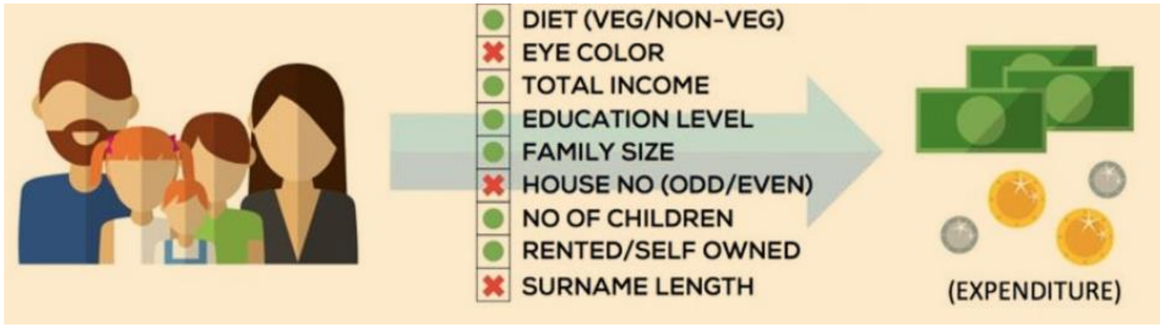
id	income	expenditure
1	19	60
2	45	113
3	35	94
4	31	90
5	25	60
6	32	88
7	21	59
8	26	61
9	24	57
10	27	78
11	9	27
12	23	72
13	33	85
14	29	63

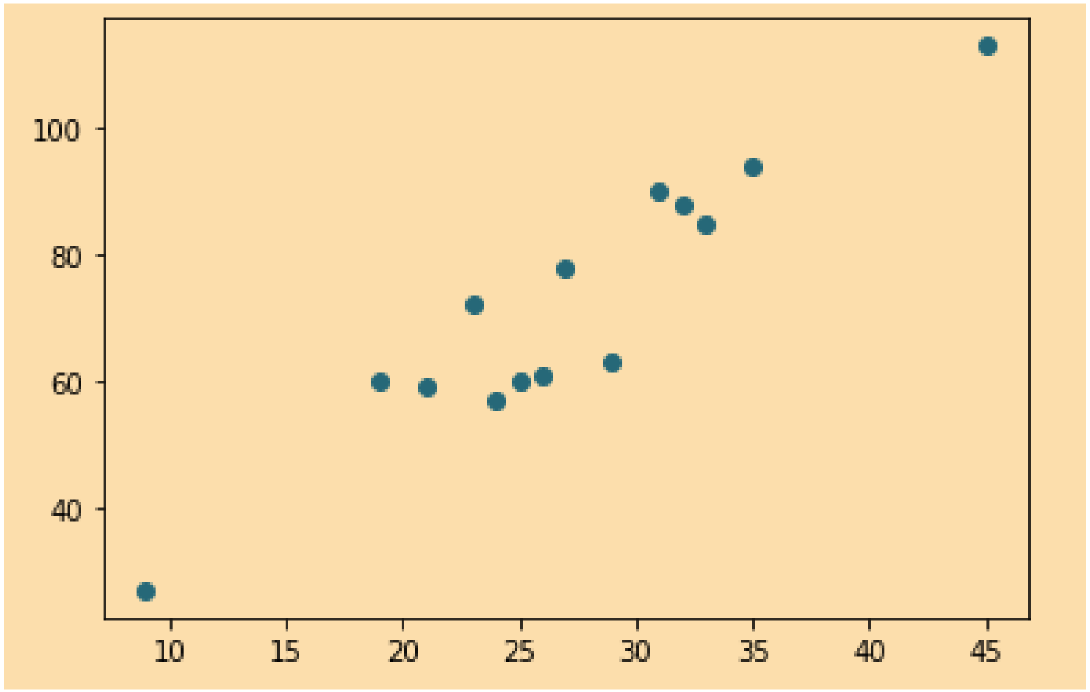
A popular way to recover $\mathbf{w}^*$ is using the *least squares* formulation

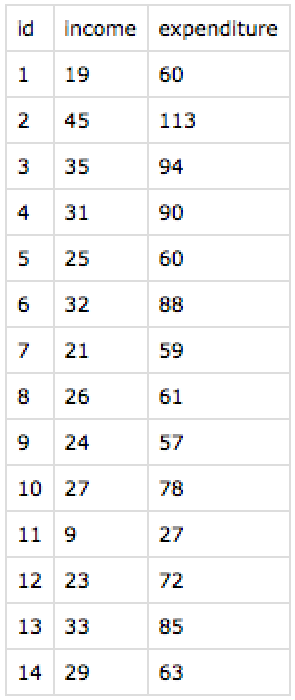
$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^\top \mathbf{w}\right)^2.$$

The linear regression problem as well as the least squares estimator, are extremely well studied and their behavior, precisely known.

- **Regression**

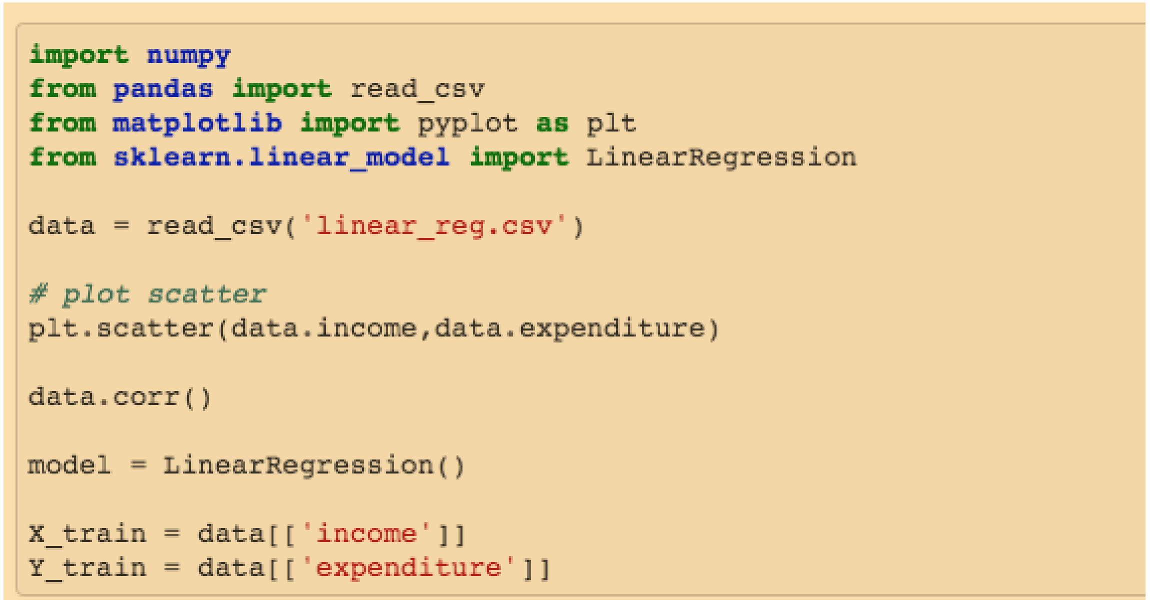




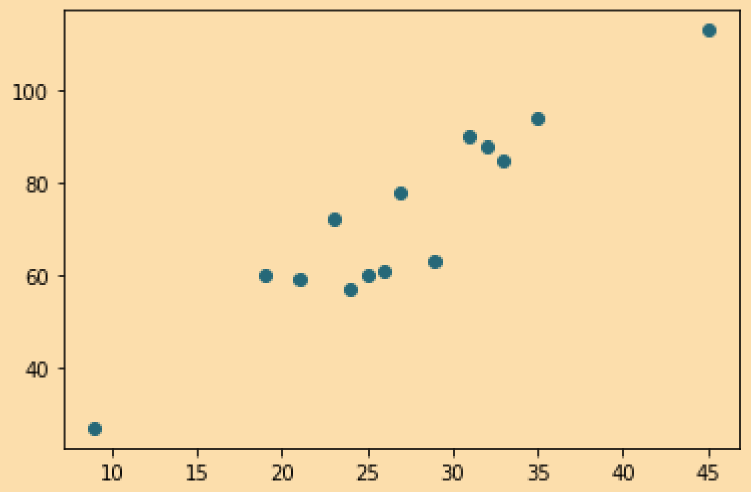


id	income	expenditure
1	19	60
2	45	113
3	35	94
4	31	90
5	25	60
6	32	88
7	21	59
8	26	61
9	24	57
10	27	78
11	9	27
12	23	72
13	33	85
14	29	63

- ## Regression



```
import numpy
from pandas import read_csv
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression

data = read_csv('linear_reg.csv')

# plot scatter
plt.scatter(data.income,data.expenditure)

data.corr()

model = LinearRegression()

X_train = data[['income']]
Y_train = data[['expenditure']]
```

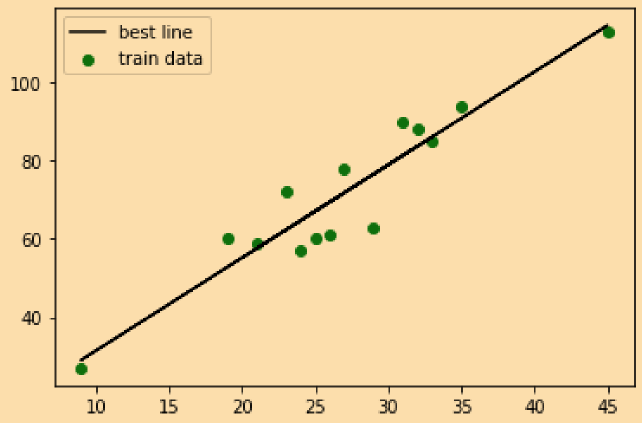


• Regression

```
# Model training
model.fit(X_train,Y_train)
model.score(X_train,Y_train)

# prediction based on trained model
model.predict([[60],[70]])

# check intercept
a = model.intercept_[0]

# get params
b = model.coef_[0][0]
a + b*numpy.array([60,70])
```

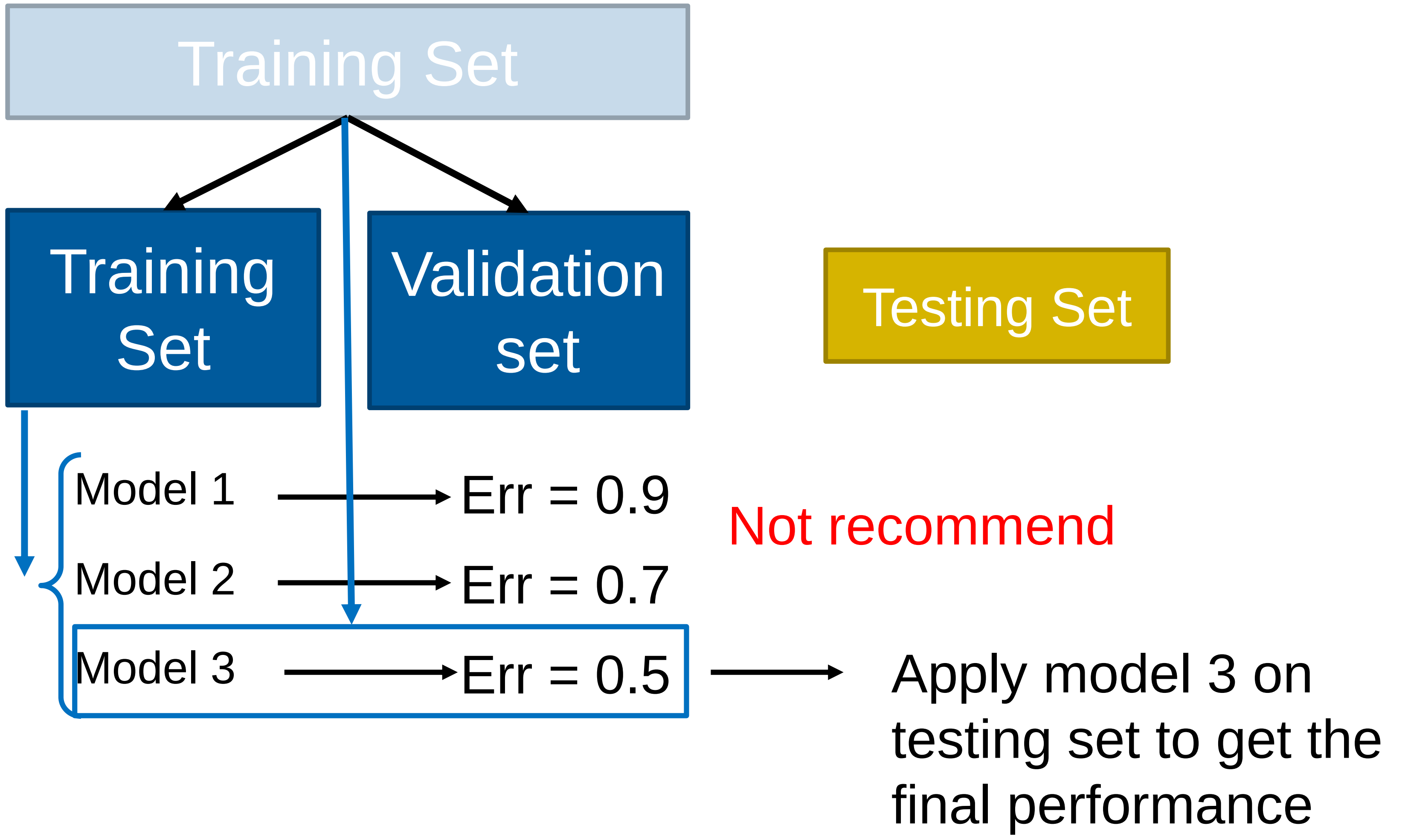
```
array([150.0667131, 173.7963006])
```

```
# plot scatter after training
plt.scatter(data.income,data.expenditure, color = 'green', label = 'train data')

# depict the best fit line
Y_train_pred = model.predict(X_train)
plt.plot(X_train, Y_train_pred, color = 'black', label = 'best line')

# output
plt.legend(loc = 2)
plt.show()
```

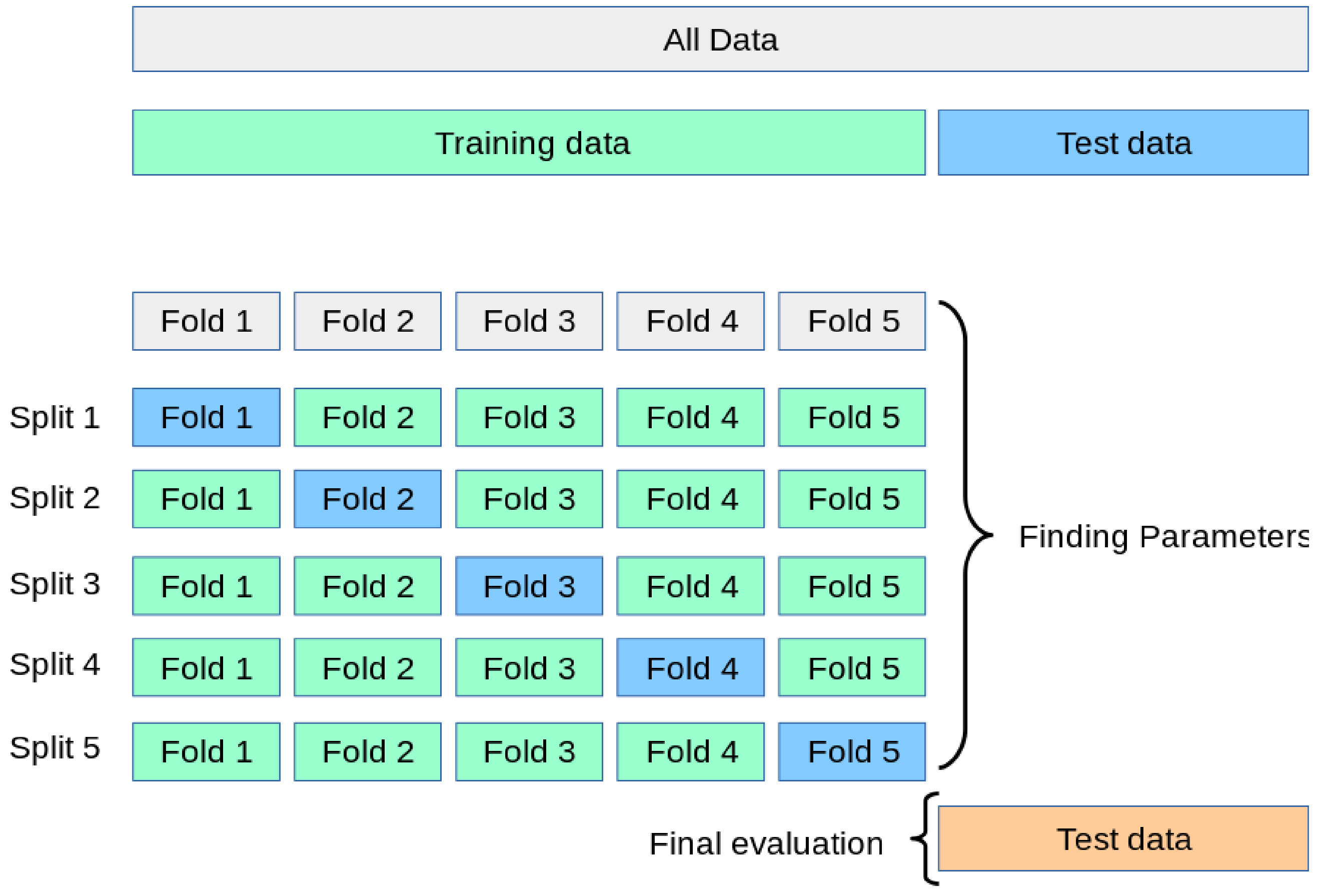


Model Evaluation

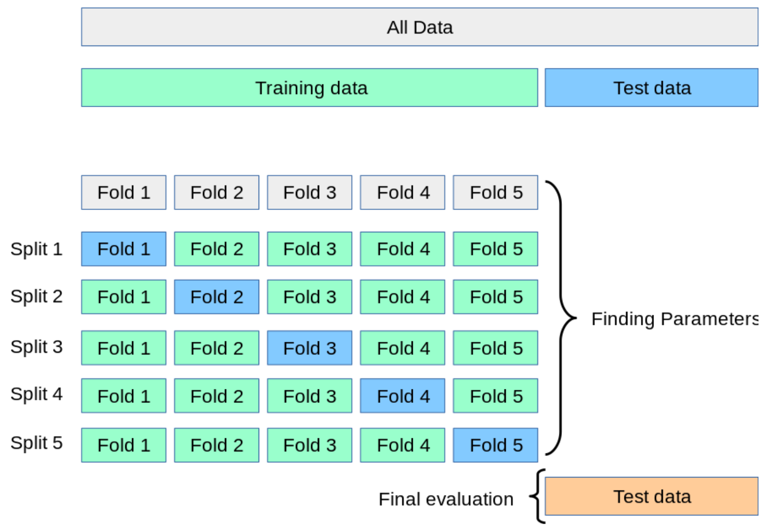
Train-Val-Test Set in DNN model training



- ## K-fold cross validation

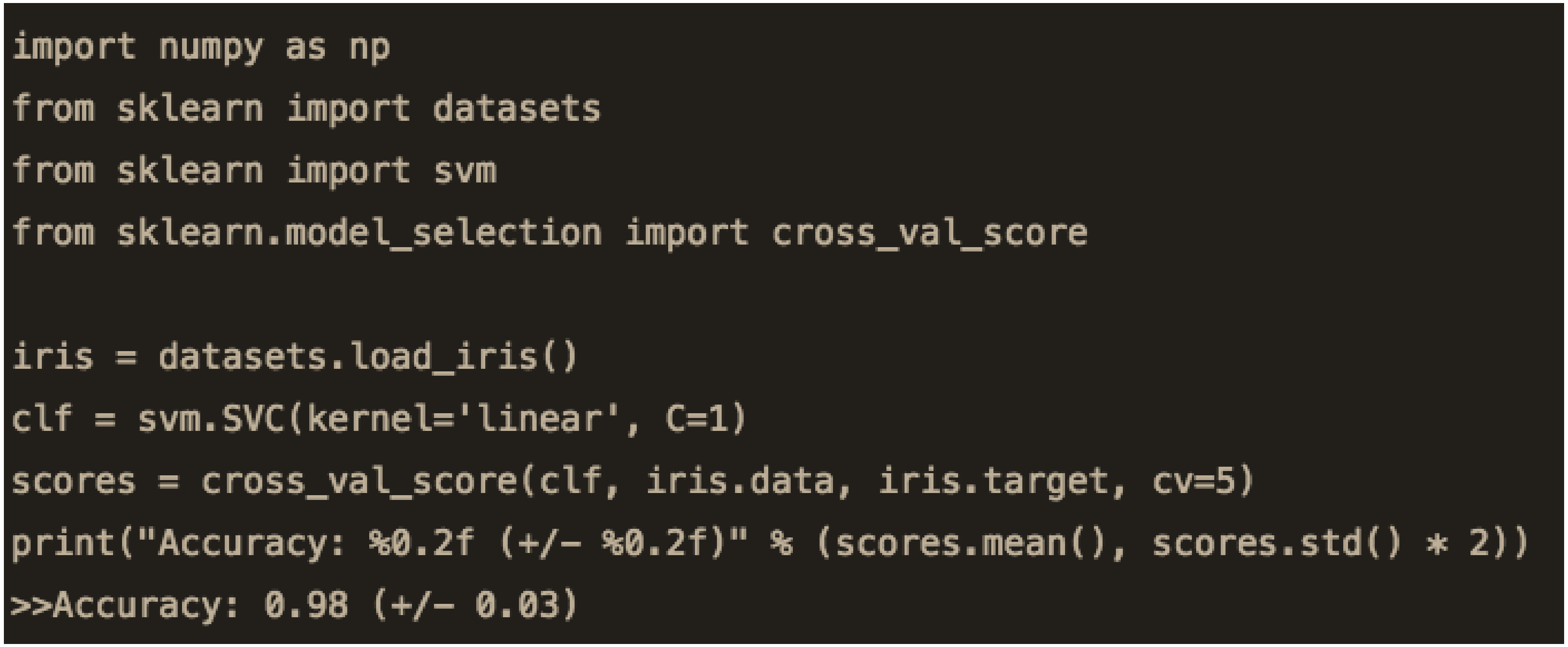


• Why K-fold cross validation?



- Avoid inappropriate random [train, test, split] division. Cross-validation can let each sample appear at least once in the validation set
- Through multiple fold divisions, it will provide us the sensitivity information of our model to the training set (worst/best-case performance)
- Higher data utilization

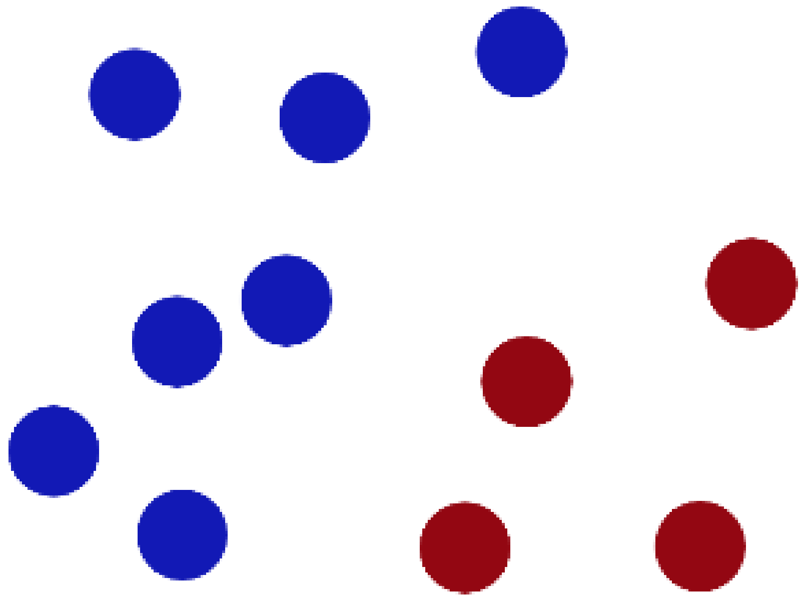
• K-fold cross validation



```
import numpy as np
from sklearn import datasets
from sklearn import svm
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()
clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, iris.data, iris.target, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
>>Accuracy: 0.98 (+/- 0.03)
```

Support Vector Machine

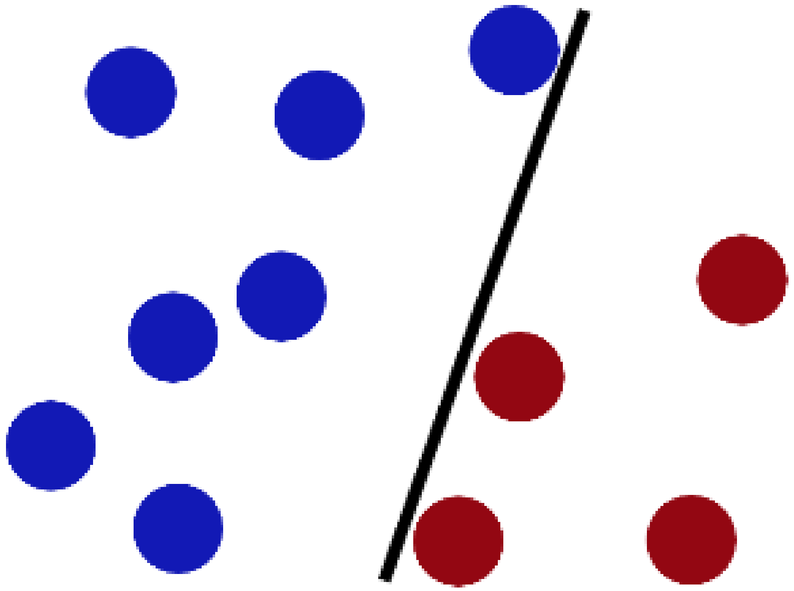
Support Vector Machine --- 2D



On Valentine's Day long ago, the hero was going to save his lover, but the devil played a game with him.

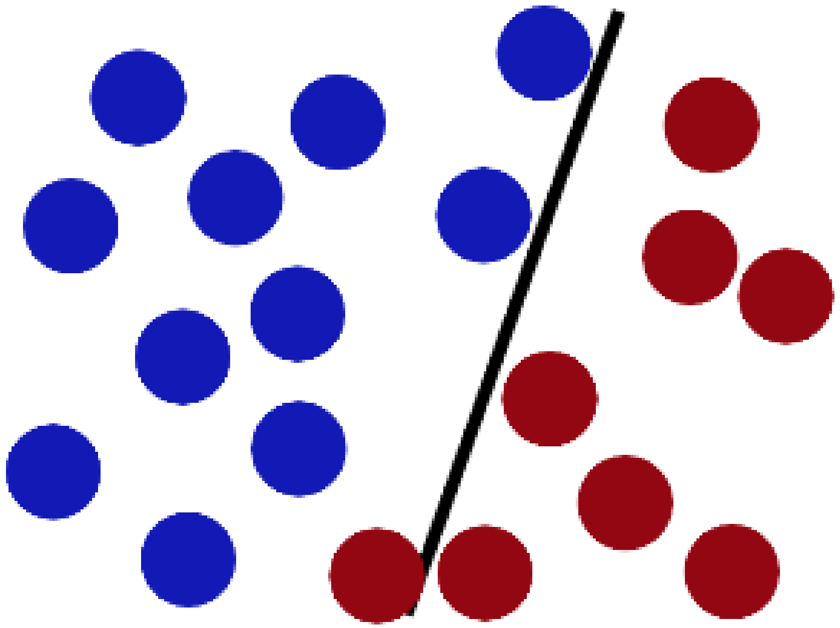
The devil placed balls of two colors on the table, and said: "You should use a stick to separate them. The requirement is try to make the boundry is correct if more balls are put on."

• Support Vector Machine --- 2D



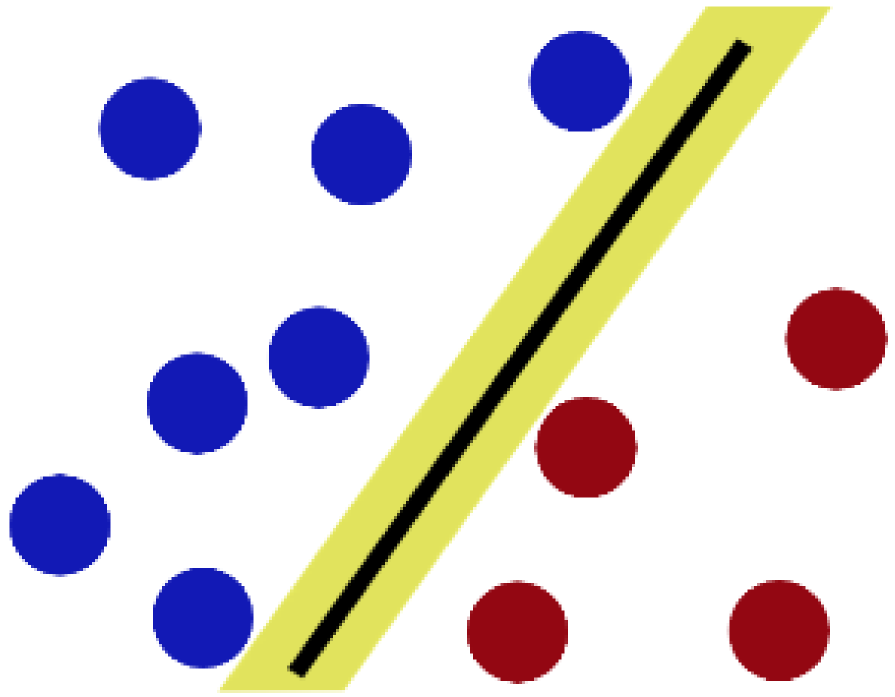
So the hero put it like this, and did a good job ...

• Support Vector Machine --- 2D



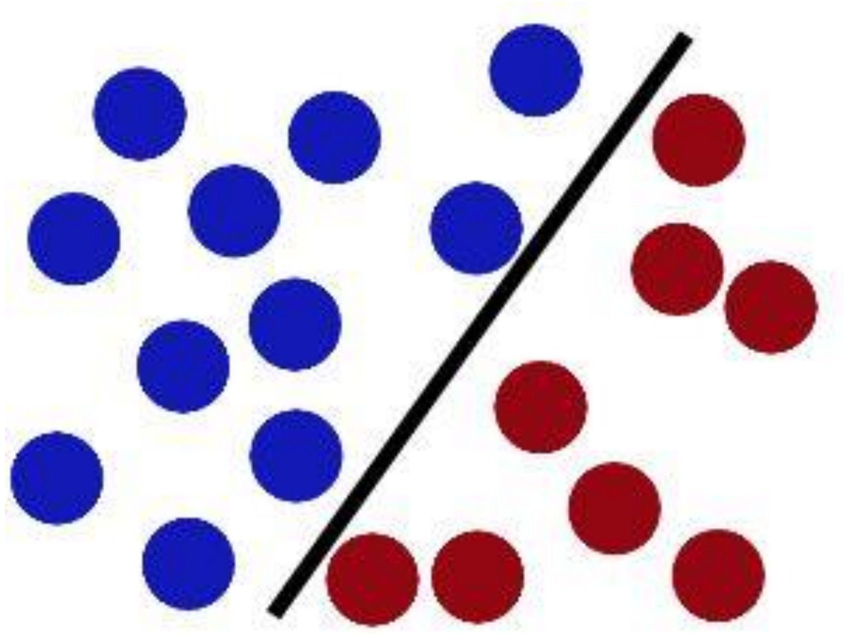
Then the devil put more balls on the table, it seems that one of the balls is in the wrong camp.

• Support Vector Machine --- 2D



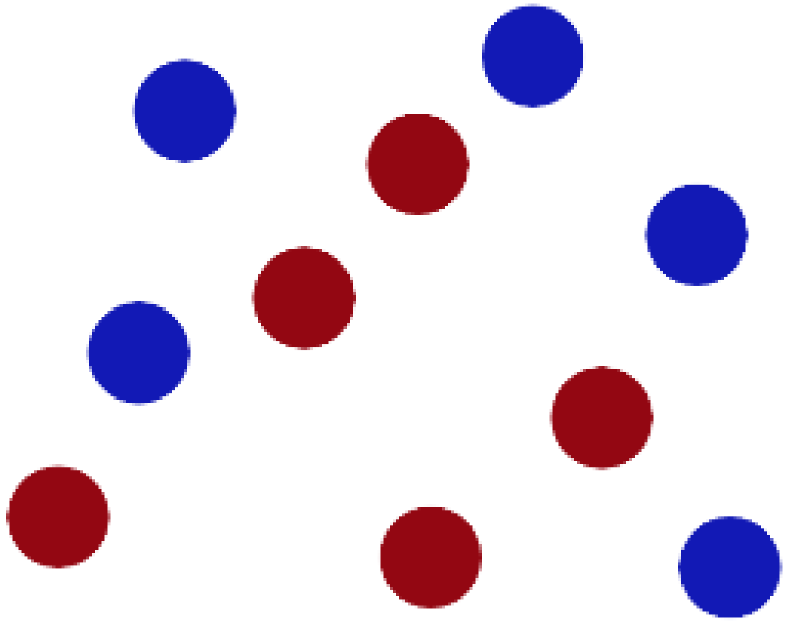
SVC: trying to put the stick in the best position so that there is as large a gap as possible on both sides of the stick.

Support Vectors Machine --- 2D



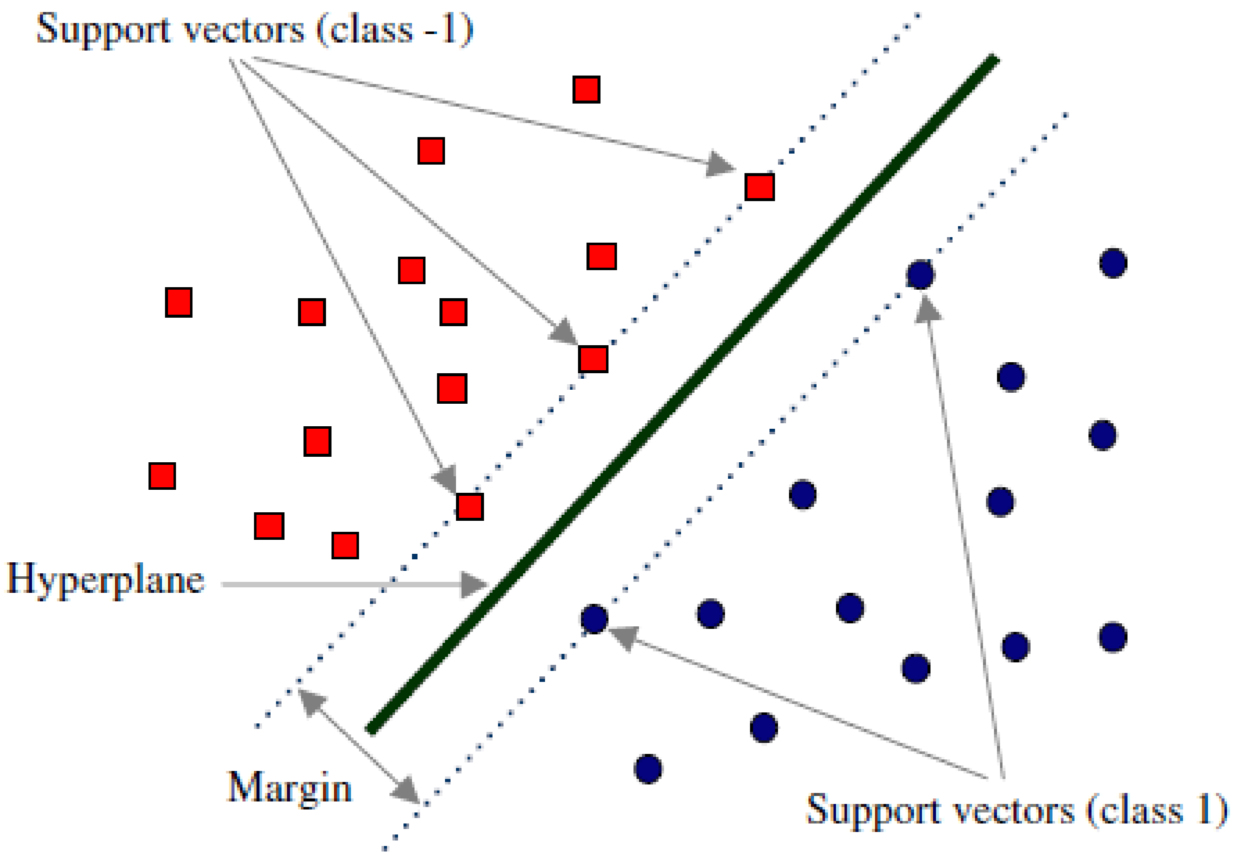
Now even if the devil puts more balls, the stick is still a good dividing boundry.

• Support Vector Machine --- 2D

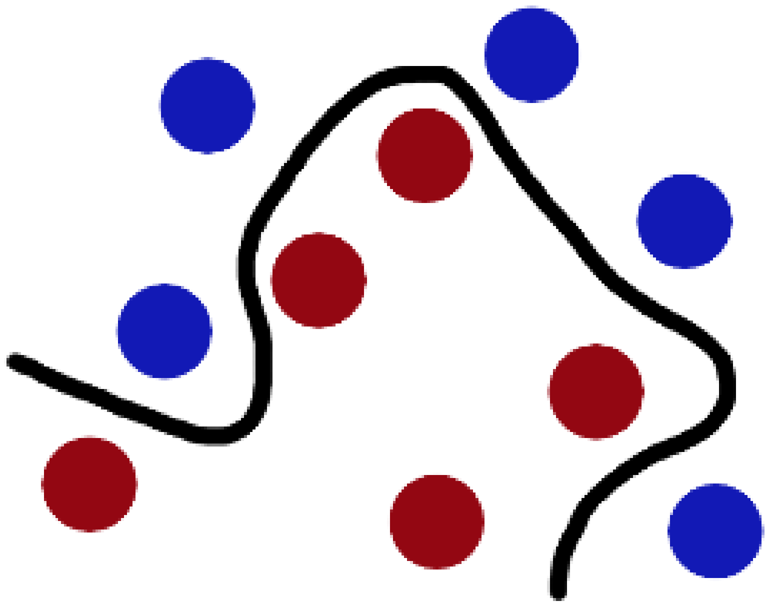


Then, the devil saw that the hero had learned a trick, so the devil gave the hero a new challenge. (There is another more important trick in the SVM toolbox)

- ## Support Vectors Machine

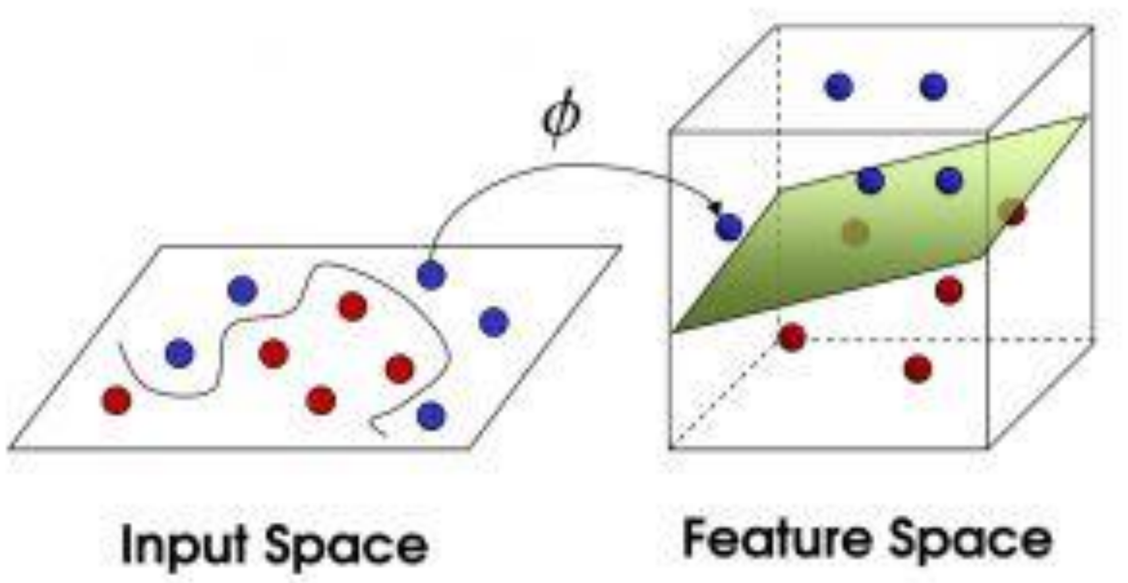


Support Vectors Classifier



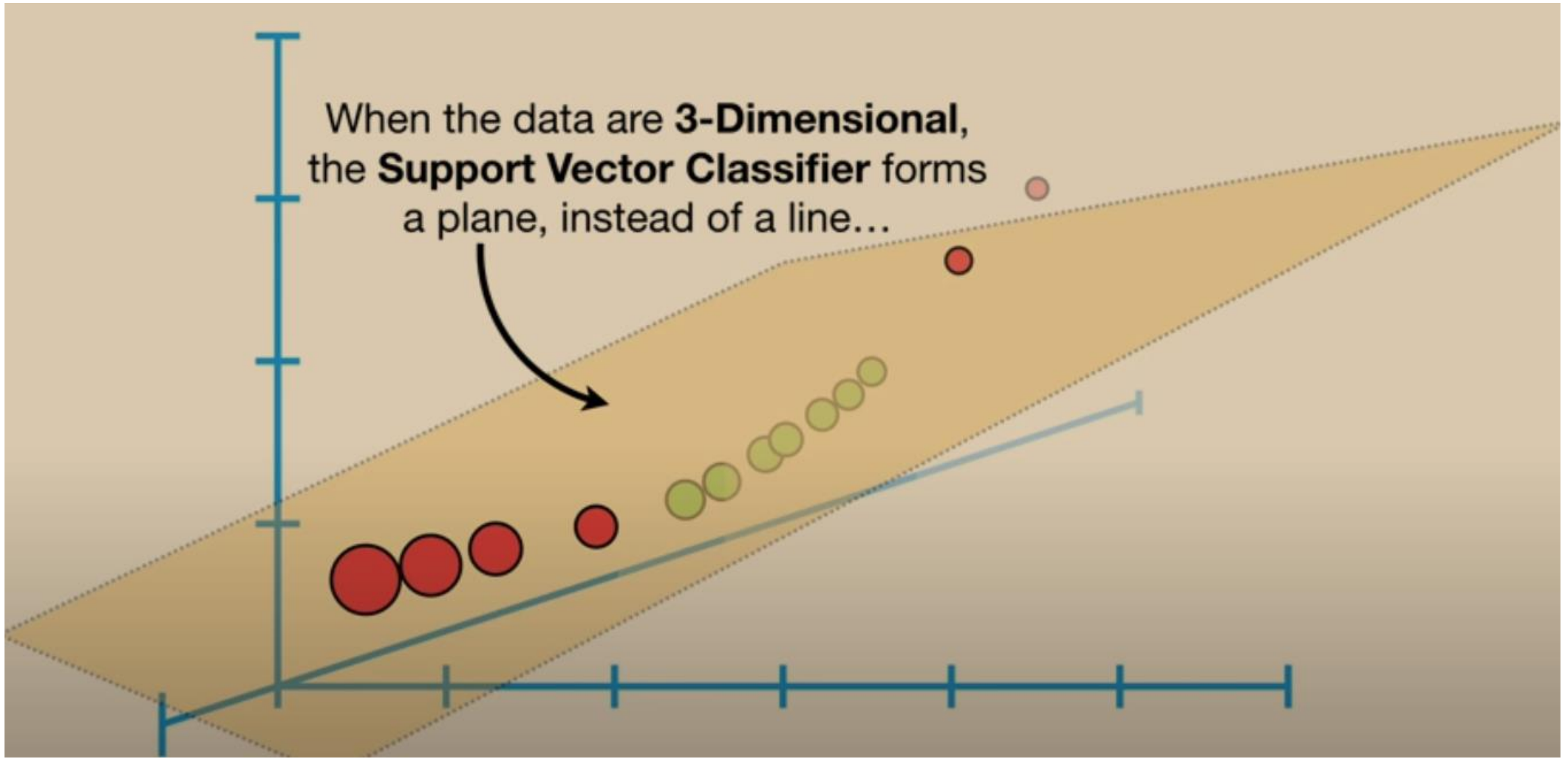
Now, from the perspective of the 2D, the balls look like they are separated by a curve.

• Support Vector Machine



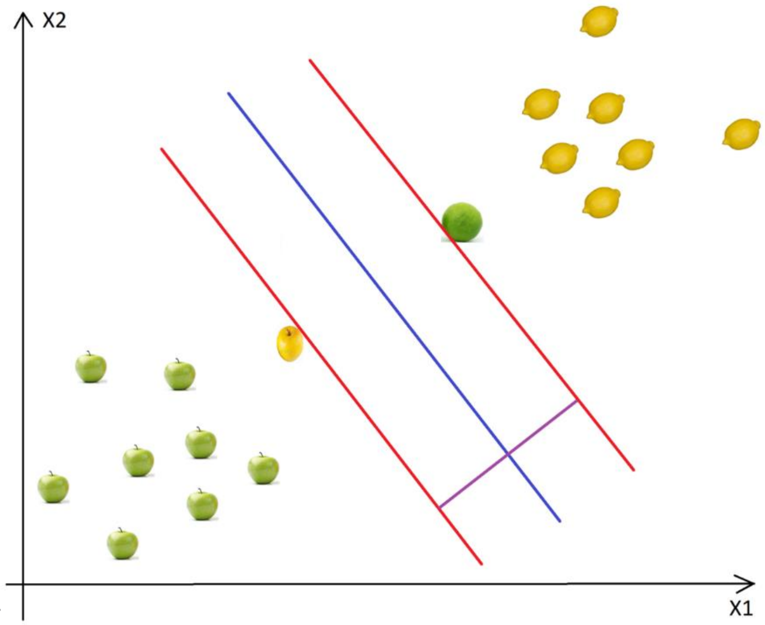
Now, the hero does not have a good boundry to help him separate the balls. But, like in action movies, the hero patted the table and the ball flew into the air. Then, with this move, the hero grabbed a piece of paper and inserted it between the two balls.

• Support Vector Machine--- 3D


When the data are **3-Dimensional**, the **Support Vector Classifier** forms a plane, instead of a line…

Support Vector Machine--- Steps

- Select two hyperplanes (in 2D) which separates the data with no points between them (red lines)
- Maximize their distance (the margin)
- The average line (here the line half way between the two red lines) will be the decision boundary
- If the current dimension cannot be separable, change a kernel that to add dimension

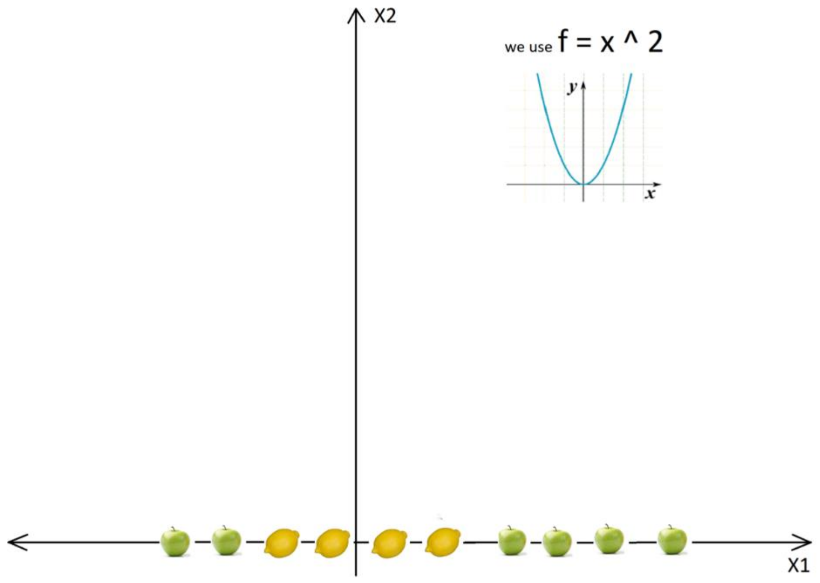


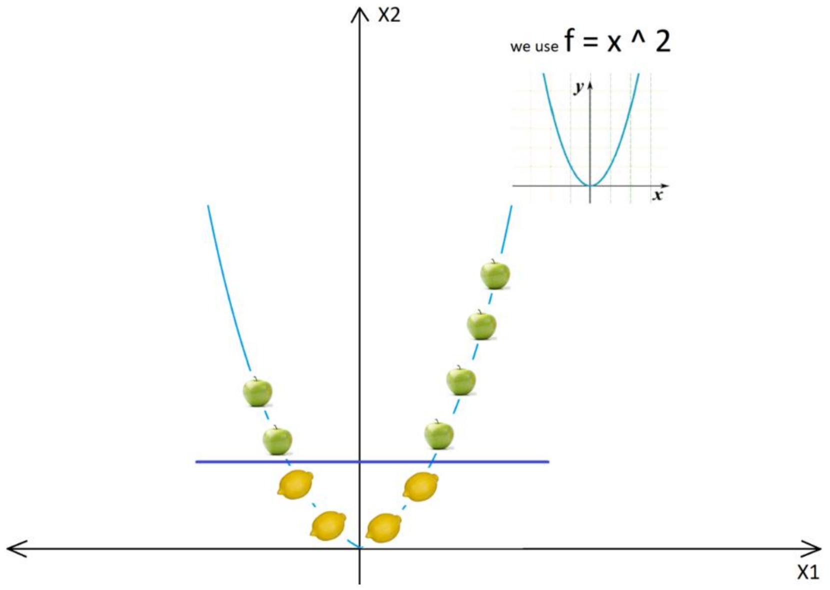
• Support Vector Machine--- Kernels

Popular kernels are:

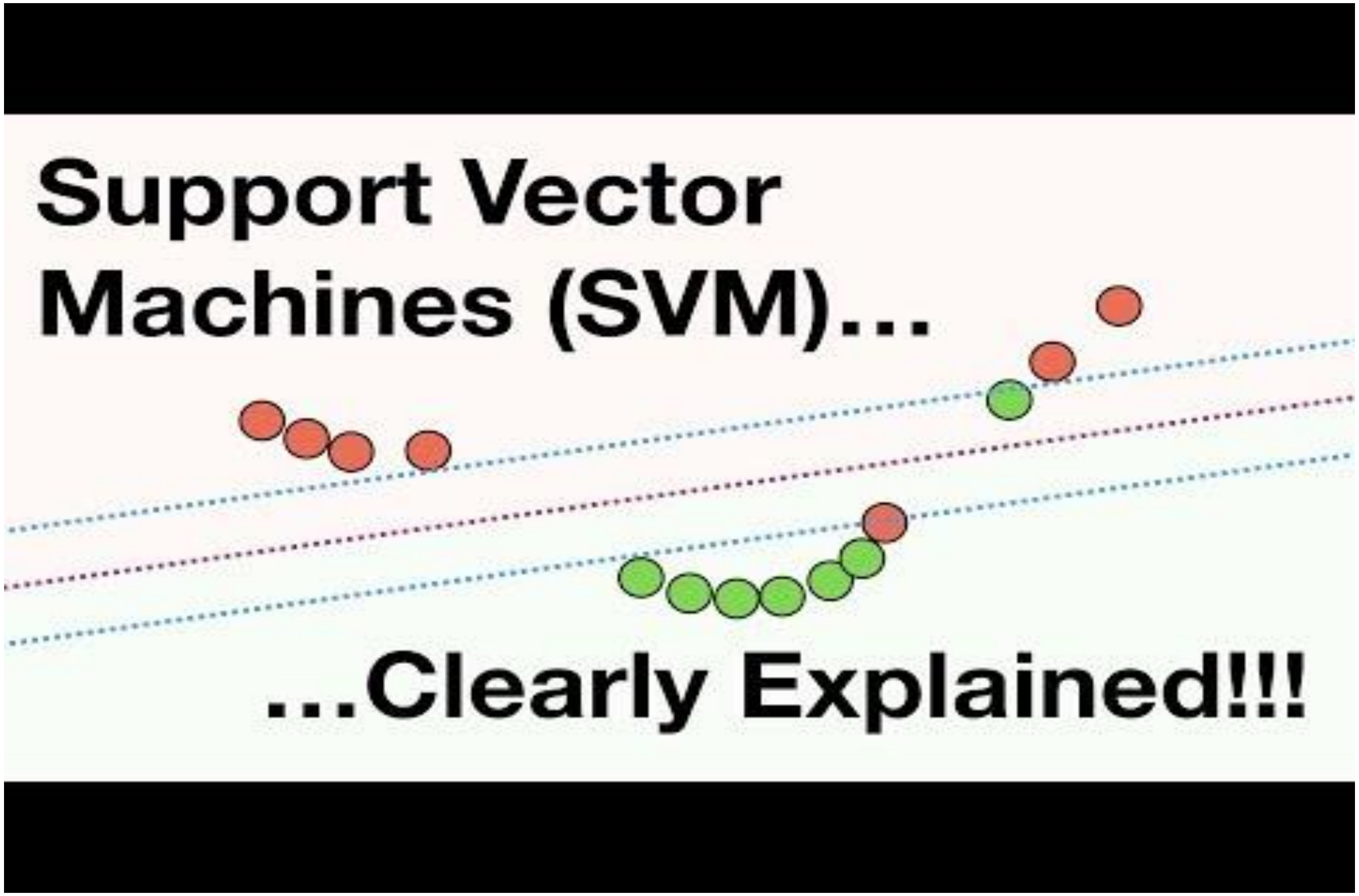
- Polynomial Kernel,
- Gaussian Kernel,
- Radial Basis Function (RBF),
- Laplace RBF Kernel,
- Sigmoid Kernel,
- Anove RBF Kernel

- ## Support Vector Machine--- 1D to 2D example





- SVM



https://www.youtube.com/watch?v=efR1C6CvhmE

Reference

- https://towardsdatascience.com/svm-and-kernel-svm-fed02bef1200