Introduction to Machine and Deep Learning using python

Loriano Storchi

University of Chieti-Pescara
INFN (Istituto Nazionale Di Fisica Nucleare) sez. Perugia
CNR SCITEC (Istituto di Scienze e Tecnologie Chimiche “Giulio Natta”) sez. Perugia

My activities

- Four Component Dirac-Kohn-Sham Theory (BERTHA code)
  - CNR and UNIPG
- Machine Learning and Chemoinformatics
  - UNIPG and MolDiscovery
- HEP (High Energy Physics) - ML techniques and FPGA (Field-programmable gate array) and Cloud Computing
  - INFN and CERN
- Bio and Chemoinformatics
  - UNICH

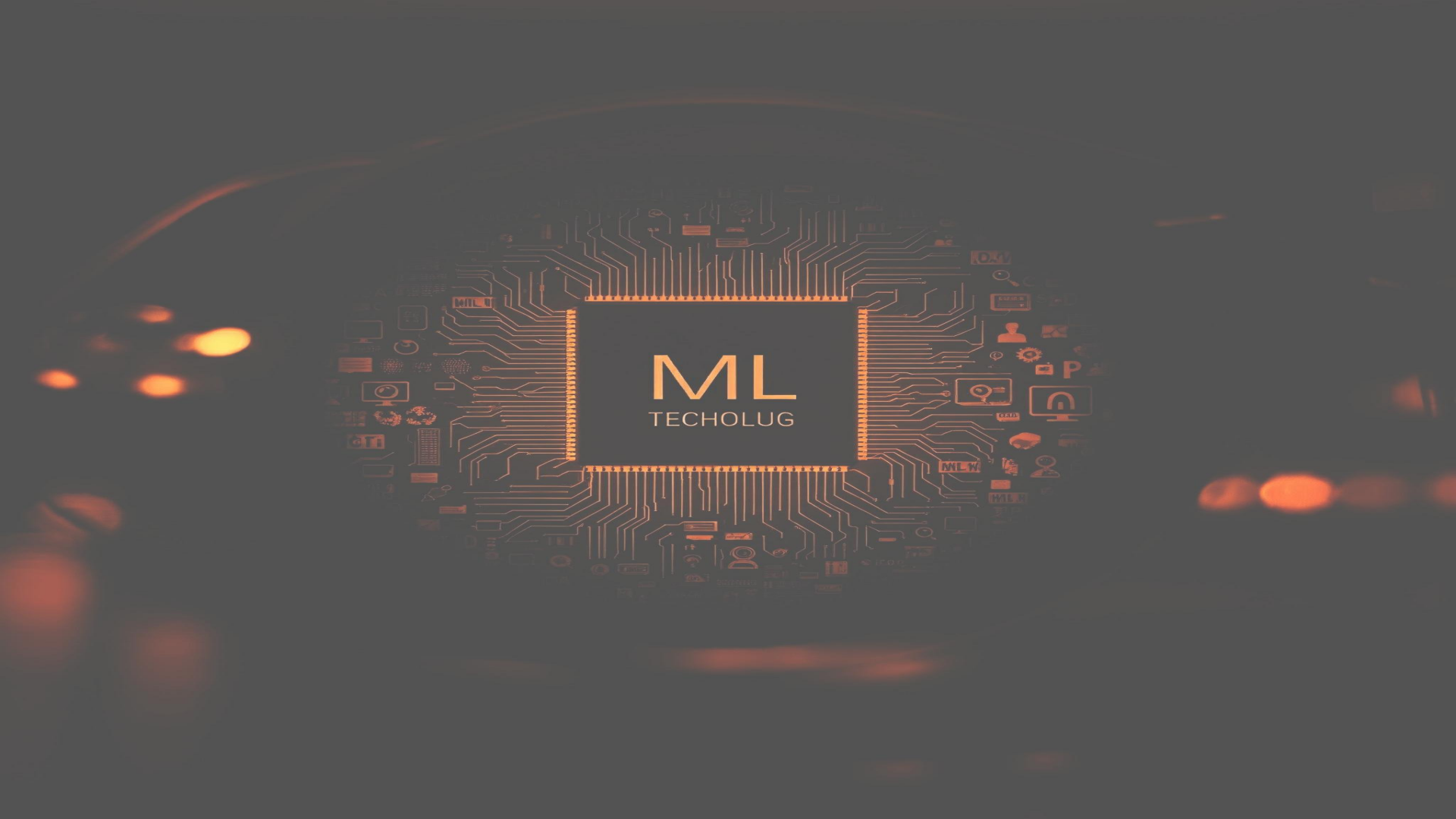
LINK TO THE CODE:

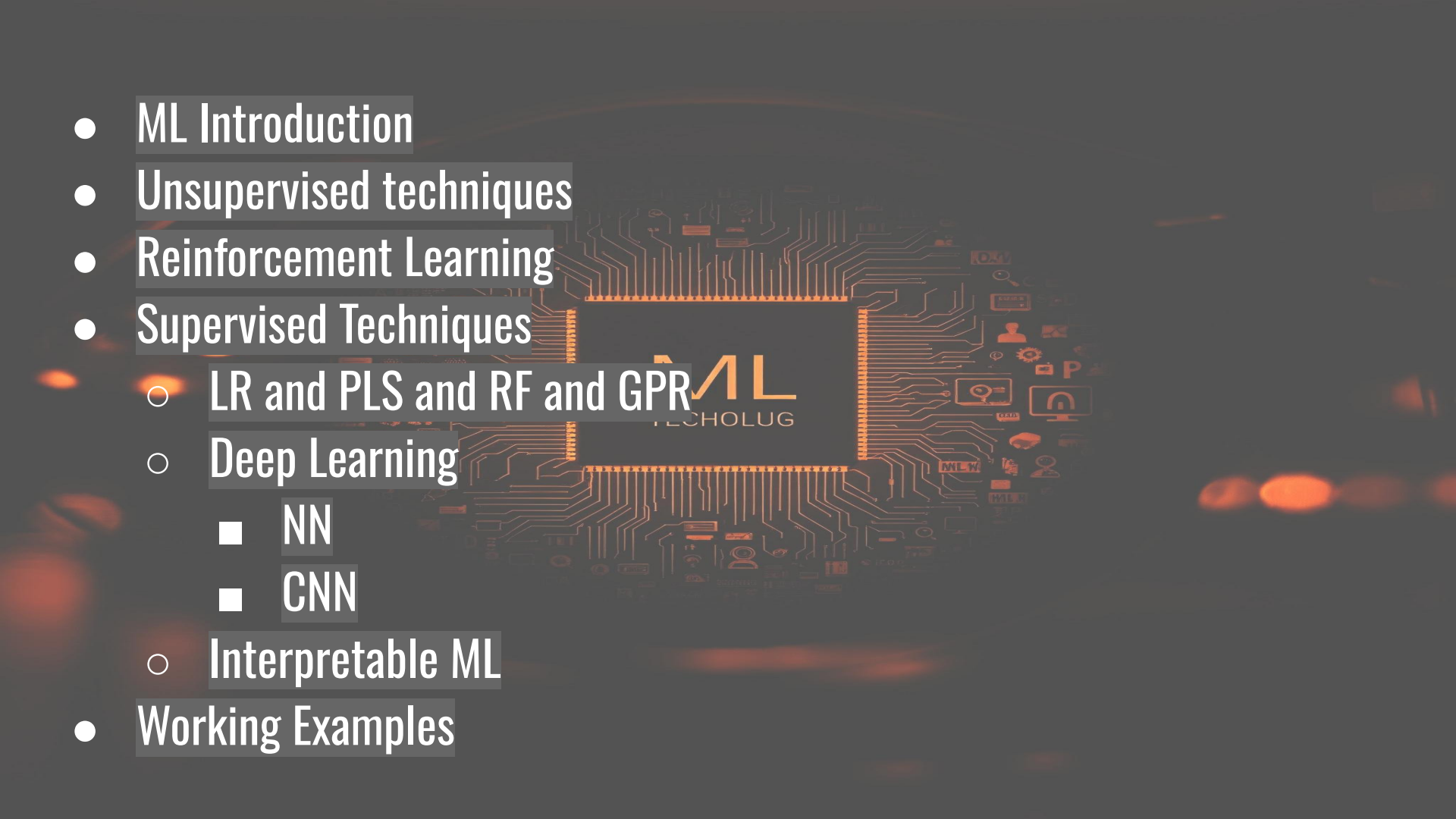
https://github.com/lstorchi

ML_Teaching

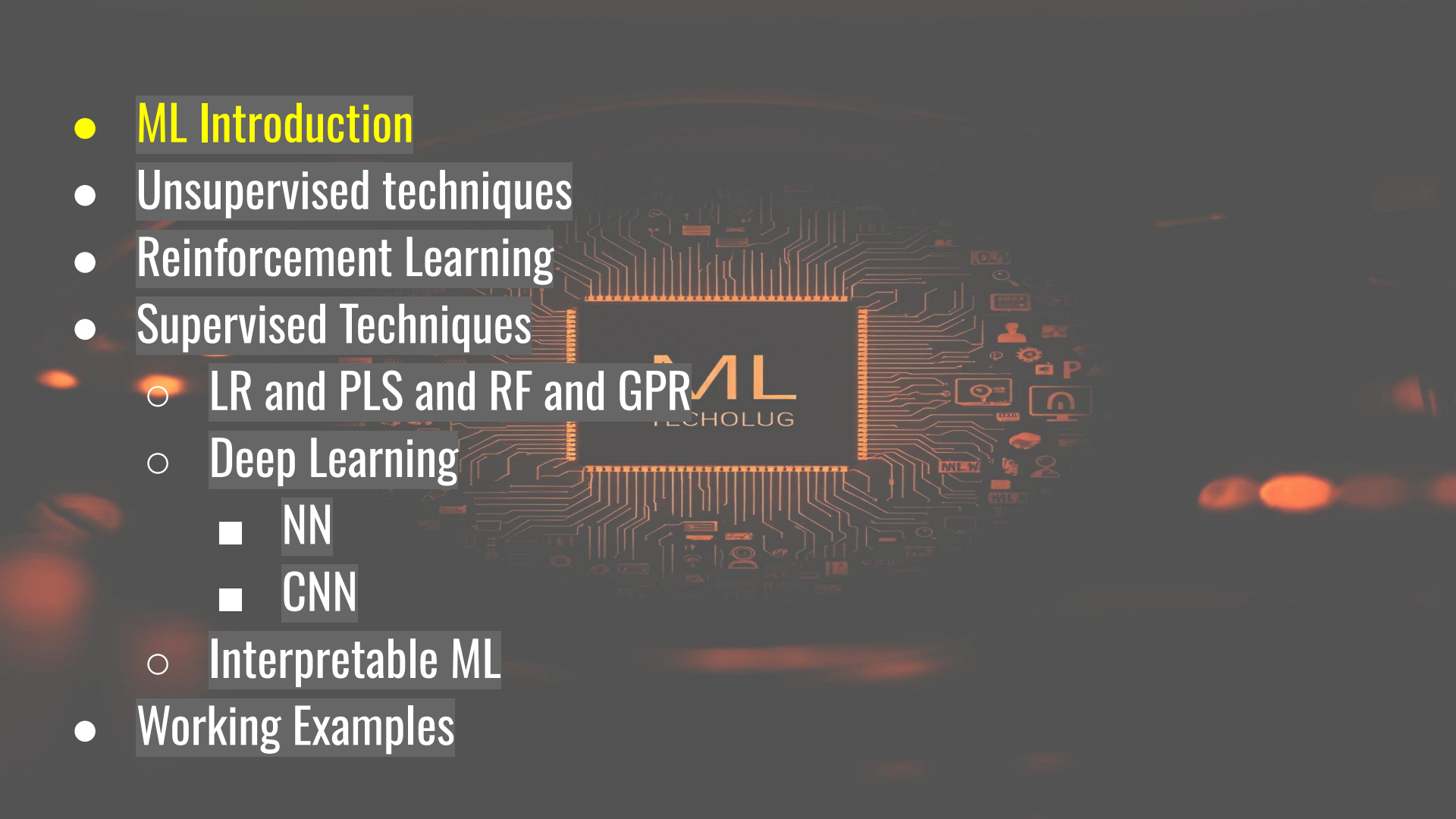
loriano@storchi.org

- **Scikit-learn** (also known as sklearn) is a popular and powerful open-source Python library for machine learning.It provides a wide range of tools.
- **Pandas** is a powerful and versatile open-source Python library for data manipulation and analysis. It provides high-performance, easy-to-use data structures and data analysis tools.
- **TensorFlow** An open-source library developed by Google for numerical computation and large-scale machine learning.. msily for deep neural networks
- **Keras** A high-level API for building and training neural networks
- **Matplotlib** is a comprehensive and widely-used plotting library in Python. It provides a flexible and powerful way to create static, interactive, and animated visualizations in Python


ML
TECHOLUG



- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples



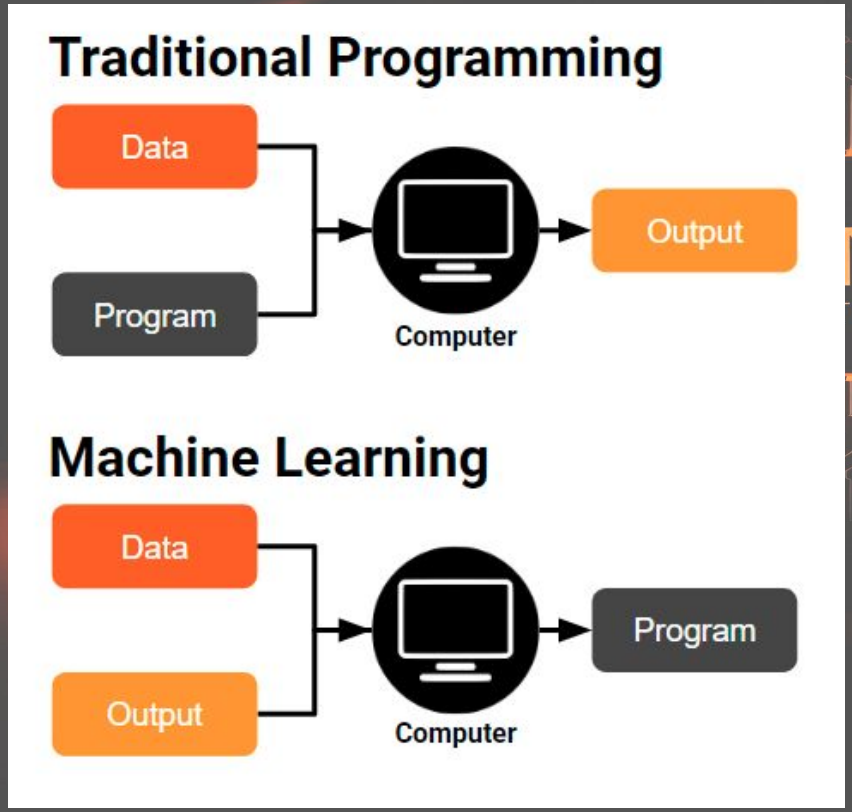
- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

Machine Learning

Machine learning techniques can be divided into two foremost types:

- **Unsupervised**: find hidden patterns or intrinsic structures in data. They are used to draw inferences from data sets consisting of input data without labeled responses (i.e. clustering algorithms)
- **Reinforcement Learning**: the algorithms learn to react to an environment on their own. An agent is in a situation of trial and error, where the consequences of its actions have an impact on the environment and also on the problem's goal. The agent is punished or rewarded on the basis of its behavior, with the idea that, in the future, it will prefer optimal actions (i.e. our intelligent cache system)
- **Supervised**: used when you want to predict or explain the data you possess. A supervised algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions

Machine Learning



A machine learning approach to detecting odd and even numbers, such as using a binary classification model trained on a dataset of numbers and their parity, differs from the standard approach, which involves dividing the number by 2 and checking the remainder.

Machine Learning - Features

Features, also known as descriptors, are the input variables used to make predictions.
In cheminformatics, features often include molecular weight, chemical structure, and physical properties.
They can be calculated or experimentally determined.
Careful selection of features is crucial for model performance.
Feature engineering techniques can improve model accuracy.

Machine Learning

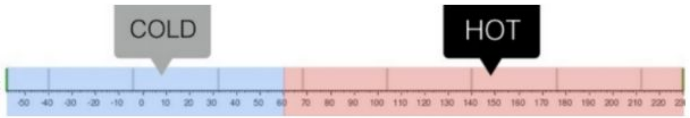
Regression

What will be the temperature tomorrow?

Fahrenheit

Classification

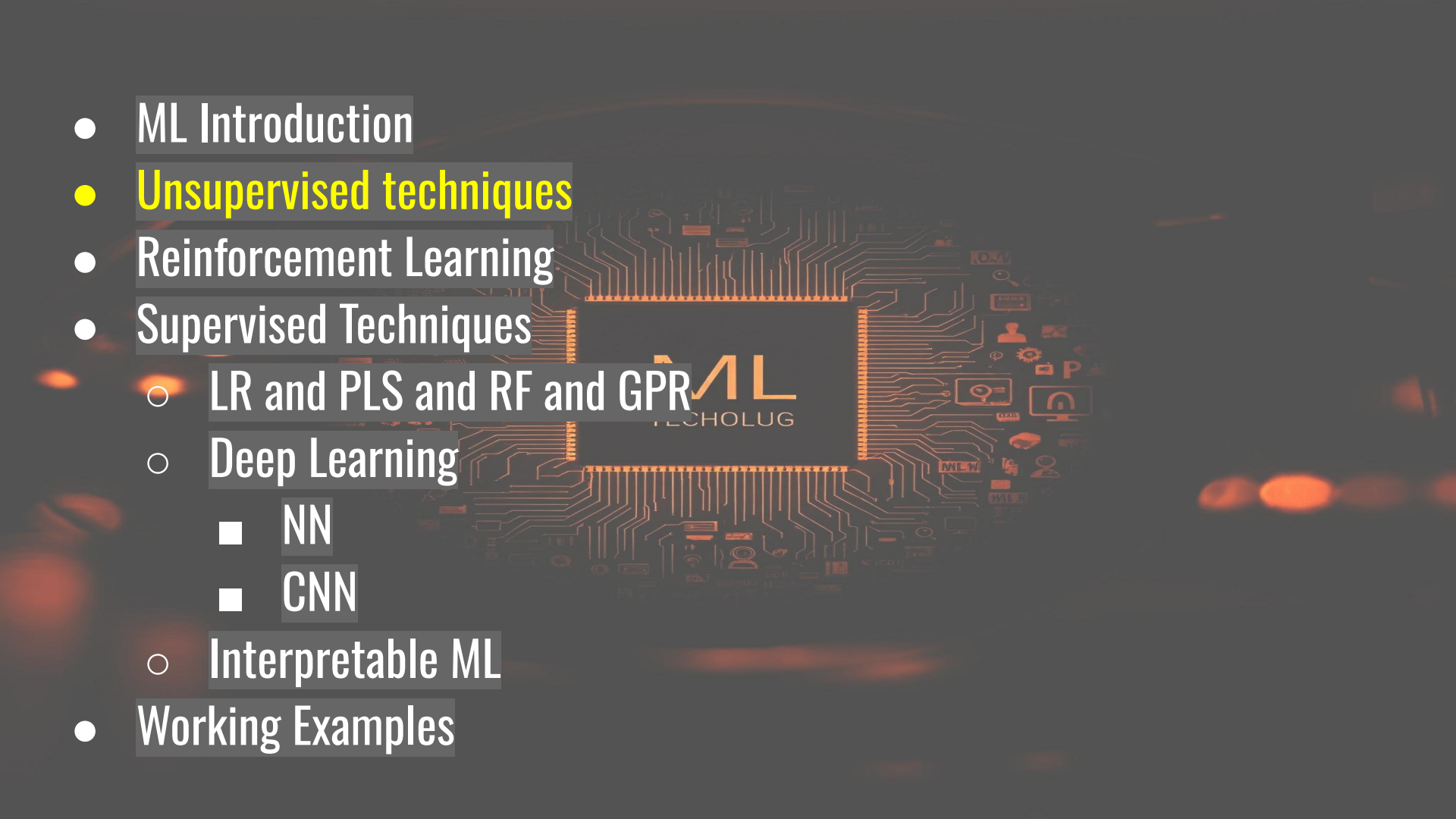
Will it be hot or cold tomorrow?



Fahrenheit

Features could be: the day of the year and the today temperature

Label: is the temperature for the regression and



- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

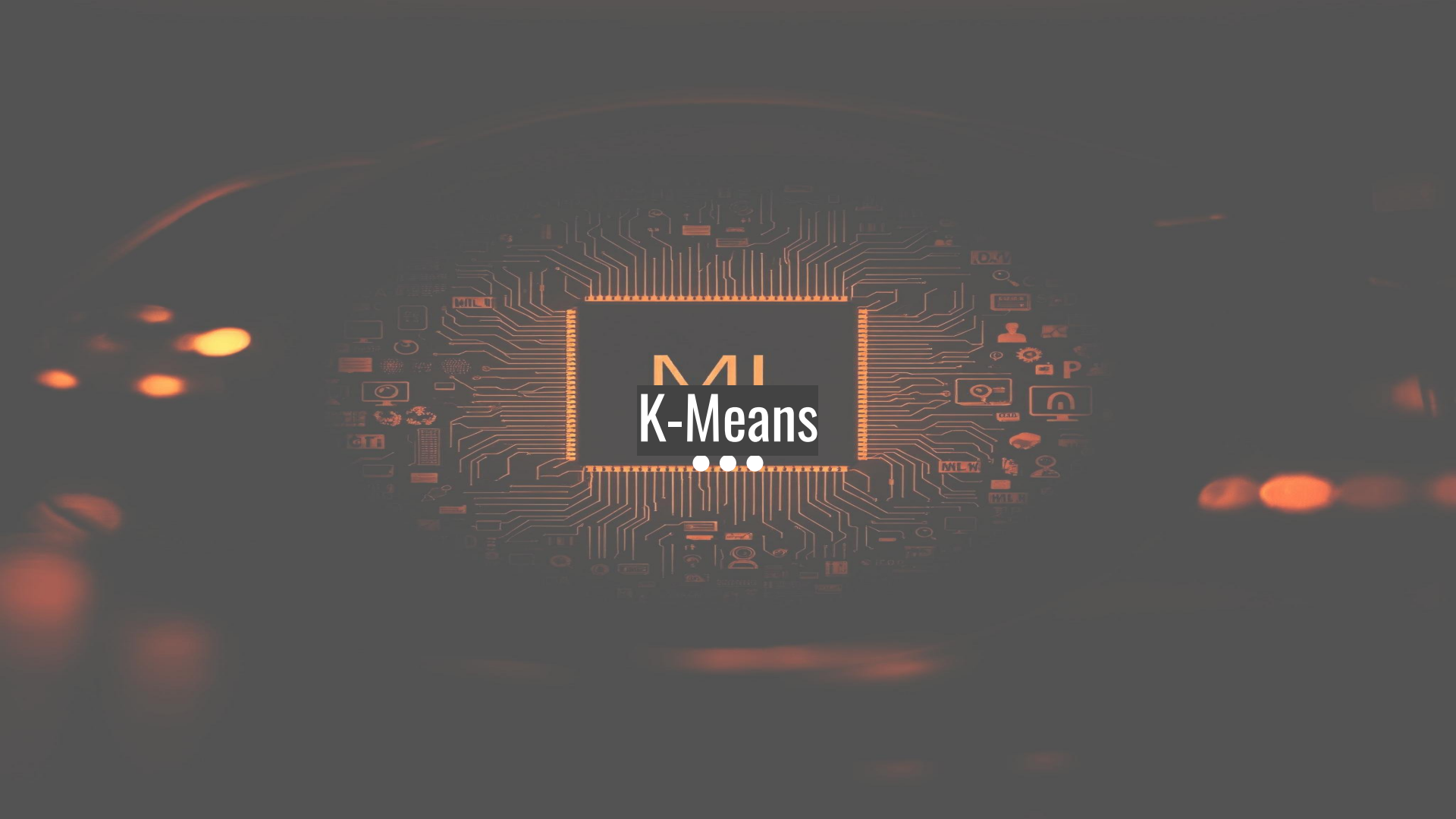
Machine Learning

Machine learning techniques can be divided into two foremost types:

- **Unsupervised**: find hidden patterns or intrinsic structures in data. They are used to draw inferences from data sets consisting of input data without labeled responses (i.e. clustering algorithms)
- **Reinforcement Learning**: the algorithms learn to react to an environment on their own. An agent is in a situation of trial and error, where the consequences of its actions have an impact on the environment and also on the problem's goal. The agent is punished or rewarded on the basis of its behavior, with the idea that, in the future, it will prefer optimal actions (i.e. our intelligent cache system)
- **Supervised**: used when you want to predict or explain the data you possess. A supervised algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions

Unsupervised Machine Learning

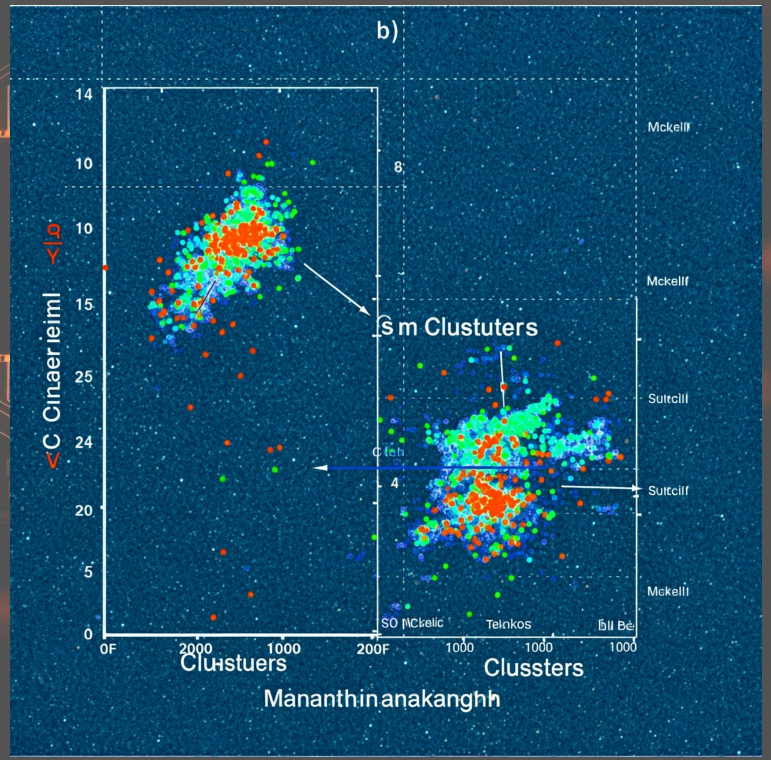
- Clustering: Clustering algorithms group similar data points together based on their inherent structure or features. Some popular clustering methods include:
  - K-Means Clustering: Partitions data into 'k' clusters, where each data point belongs to the cluster with the nearest mean.
- Dimensionality Reduction: Techniques for reducing the number of features (dimensions) in a dataset while retaining as much information as possible. This can help with visualization, noise removal, and improving the performance of other ML algorithms. Some widely used methods are:
  - Principal Component Analysis (PCA): Transforms data into a new set of uncorrelated variables (principal components) that capture the maximum variance in the data.


K-Means

Clustering K-means

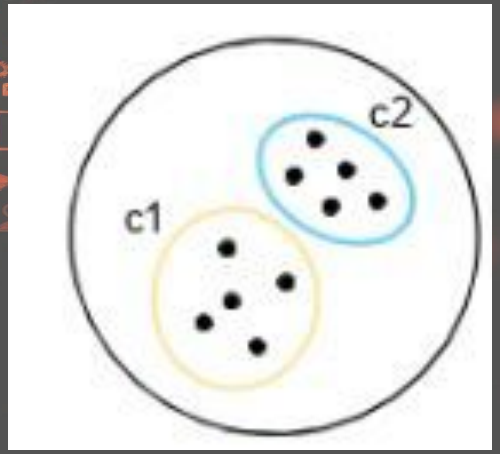
Two-dimensional data can be easily visualized for intuitive understanding..

K-means aims to minimize within-cluster distance and maximize between-cluster distance when we are considering a N-dimensional space.



Clustering K-means

In k-means clustering, the objects are divided into several clusters mentioned by the number ‘K.’ So if we say K = 2, the objects are divided into two clusters, c1 and c2



- The features or characteristics are compared, and all objects having similar characteristics are clustered together.
- The algorithm works by first randomly picking some central points (called centroids) and then assigning every data point to the nearest centroid.
- Once that’s done, it recalculates the centroids based on the new groupings and repeats the process until the clusters make sense

Clustering K-means

Grouping Similar Data Points

K-Means is designed to cluster data points that share common traits, allowing patterns or trends to emerge.

Minimizing Within-Cluster Distance

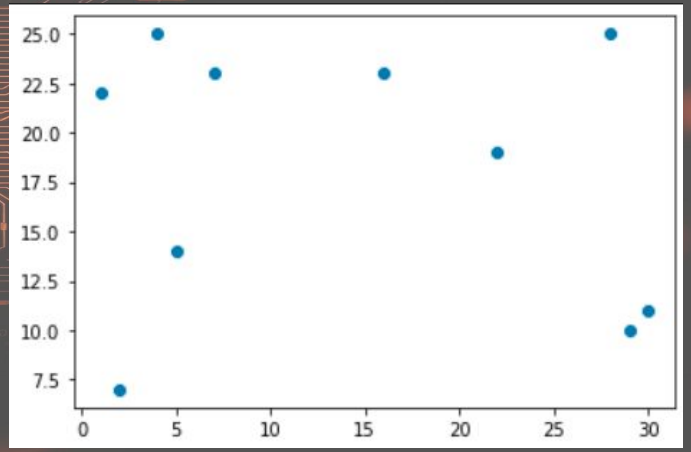
Keep data points in each group as close to the cluster's centroid as possible

Maximizing Between-Cluster Distance

K-Means also aims to maintain clear separation between different clusters.

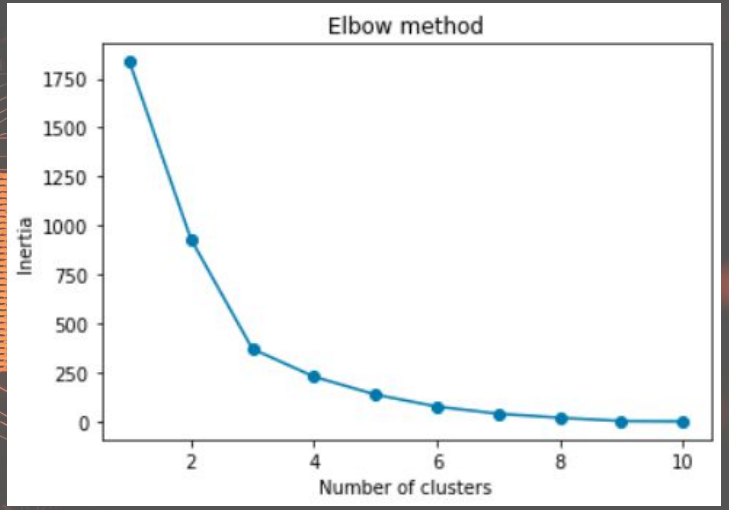
Clustering K-means: Example and Code

```
import matplotlib.pyplot as plt
import random

x = []
y = []

for i in range(10):
    x.append(random.randint(1, 30))
    y.append(random.randint(1, 30))

plt.scatter(x, y)
plt.show()
```



Clustering K-means: Example and Code

```python
from sklearn.cluster import KMeans

data = list(zip(x, y))
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



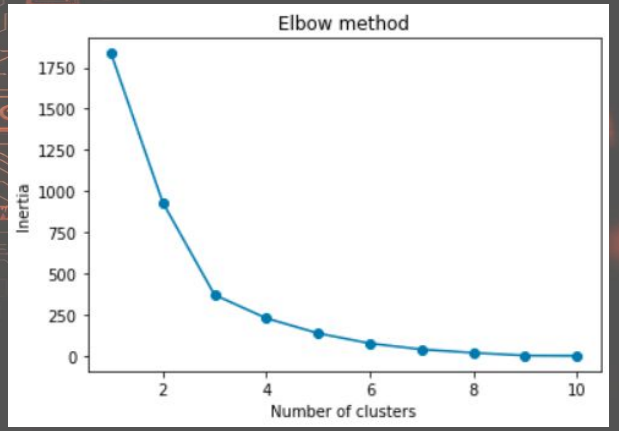
The elbow method shows that 3 is a good value for K, so we retrain and visualize the result:

Clustering K-means: Example and Code

inertia is a key concept that measures the compactness of your clusters. Think of it as a way to quantify how tightly grouped the data points are within each cluster

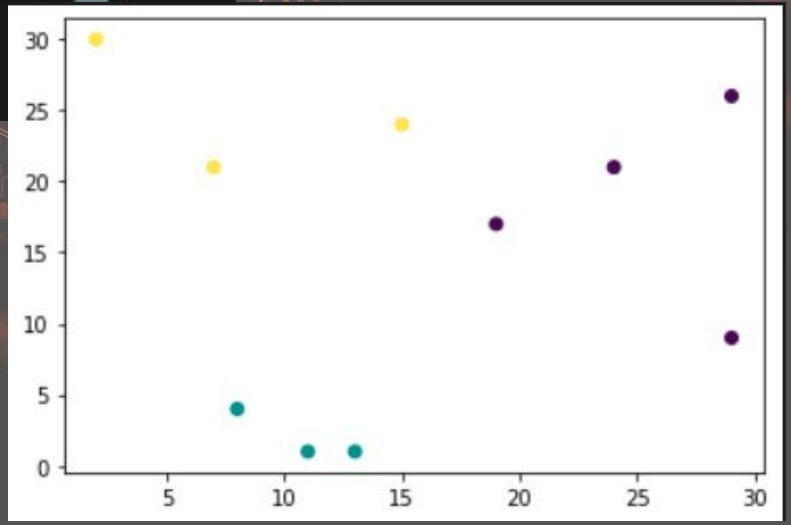
Inertia is calculated by summing the squared distances between each data point and its assigned cluster center (centroid).

Find the Elbow: Look for the point on the curve where the rate of decrease in inertia starts to slow down significantly. This point resembles an elbow, hence the name "elbow method.



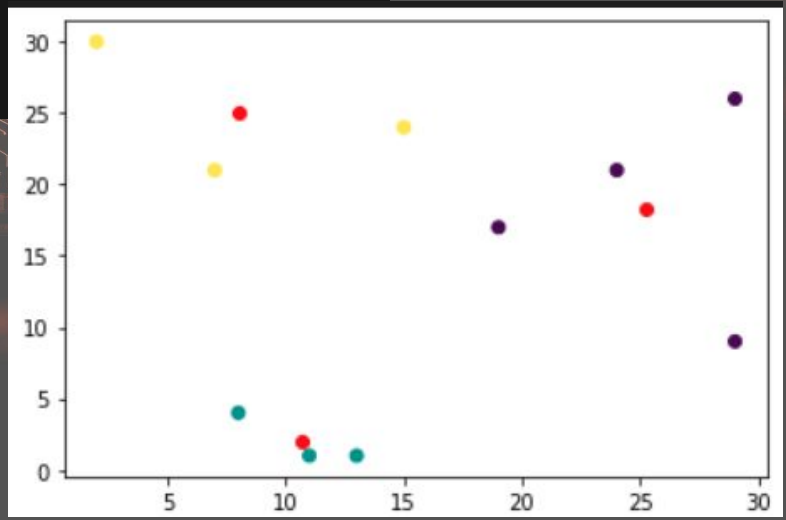
Clustering K-means: Example and Code

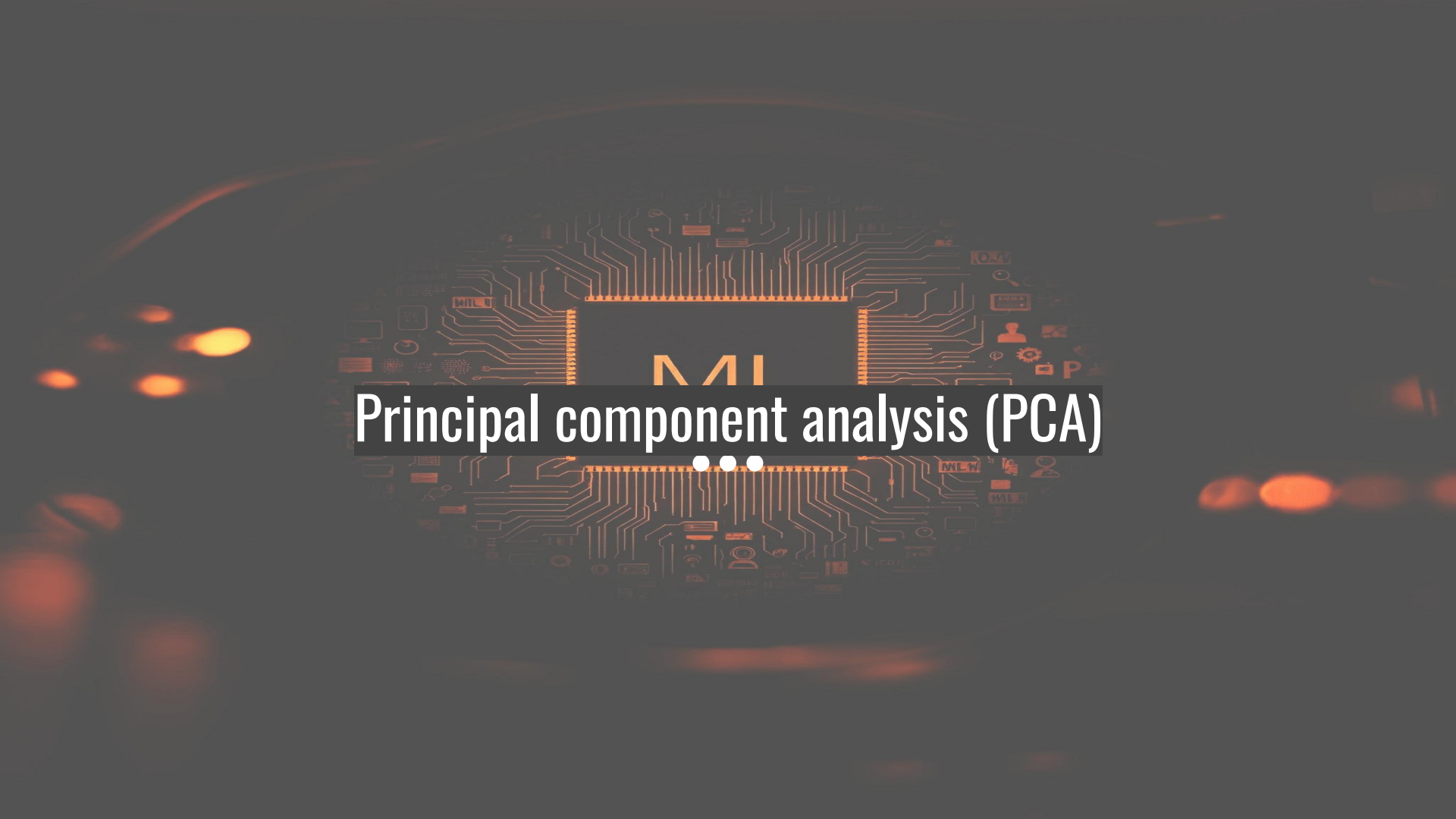
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```



Clustering K-means: Example and Code

```
# plots also centroids of clusters
plt.scatter(x, y, c=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:, 0], \
            kmeans.cluster_centers_[:, 1], \
            c='red')
plt.show()
```





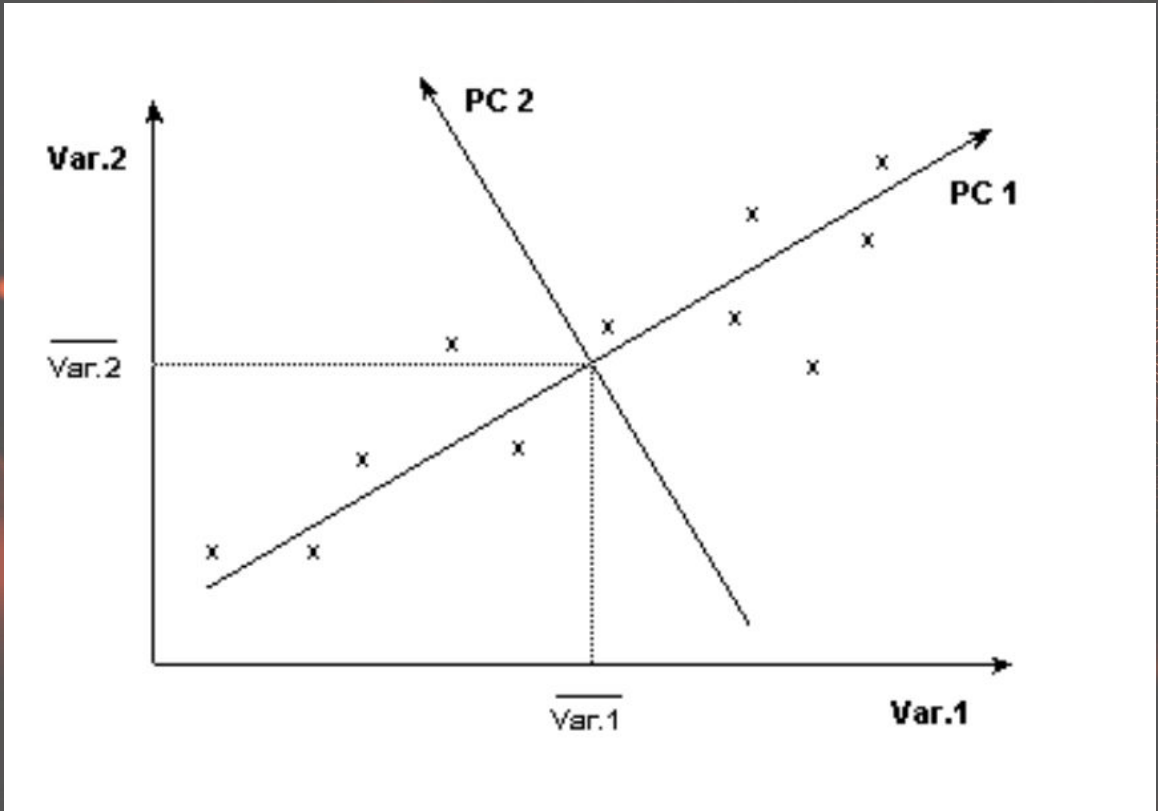
Principal component analysis (PCA)

Dimensionality Reduction:: PCA

Principal component analysis, or PCA, is a statistical procedure that allows you to summarize the information content in large data tables by means of a smaller set of “summary indices”

- Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables.
- **These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components**
- **Practically it consists on a diagonalization of the covariance matrix**

Dimensionality Reduction:: PCA



How can you observe from the figure, the first principal component (PC 1) is in the direction of maximum variance and its origin is located in the average value of the variable. The residual variance is represented by the second principal component (PC 2), in the direction perpendicular to the first component.

Dimensionality Reduction:: PCA

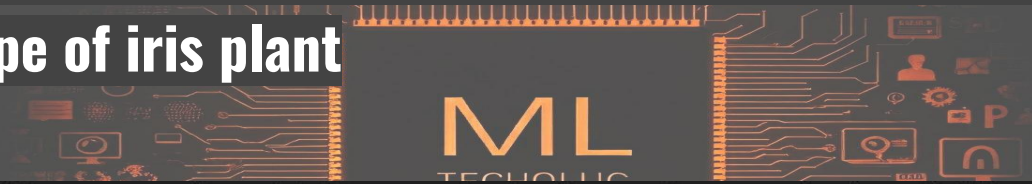
UC Irvine Machine Learning Repository

```
import pandas as pd

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

features = ['sepal length','sepal width','petal length','petal width']
label = ['target']

allnames = []
allnames.extend(features)
allnames.extend(label)

df = pd.read_csv(url, names=allnames)
print(df.head())
print(df.columns)
```

Dimensionality Reduction:: PCA

UC Irvine Machine Learning Repository

The data set contains 3 classes of N instances each, where each class refers to a type of iris plant




```
   sepal length  sepal width  petal length  petal width        target
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
Index(['sepal length', 'sepal width', 'petal length', 'petal width', 'targ
```

SCALE the Data

Dimensionality Reduction:: PCA

Scale the data so: mean = 0 and variance = 1

```
from sklearn.preprocessing import StandardScaler

x = df.loc[:, features].values
y = df.loc[:,label].values
print(x[0:5])
print()
x = StandardScaler().fit_transform(x)
print(x[0:5])
```

Dimensionality Reduction:: PCA

Scale the data so: mean = 0 and variance = 1

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]

[[-0.90068117  1.03205722 -1.3412724  -1.31297673]
 [-1.14301691 -0.1249576  -1.3412724  -1.31297673]
 [-1.38535265  0.33784833 -1.39813811 -1.31297673]
 [-1.50652052  0.10644536 -1.2844067  -1.31297673]
 [-1.02184904  1.26346019 -1.3412724  -1.31297673]]
```

RUN the PCA

Dimensionality Reduction:: PCA

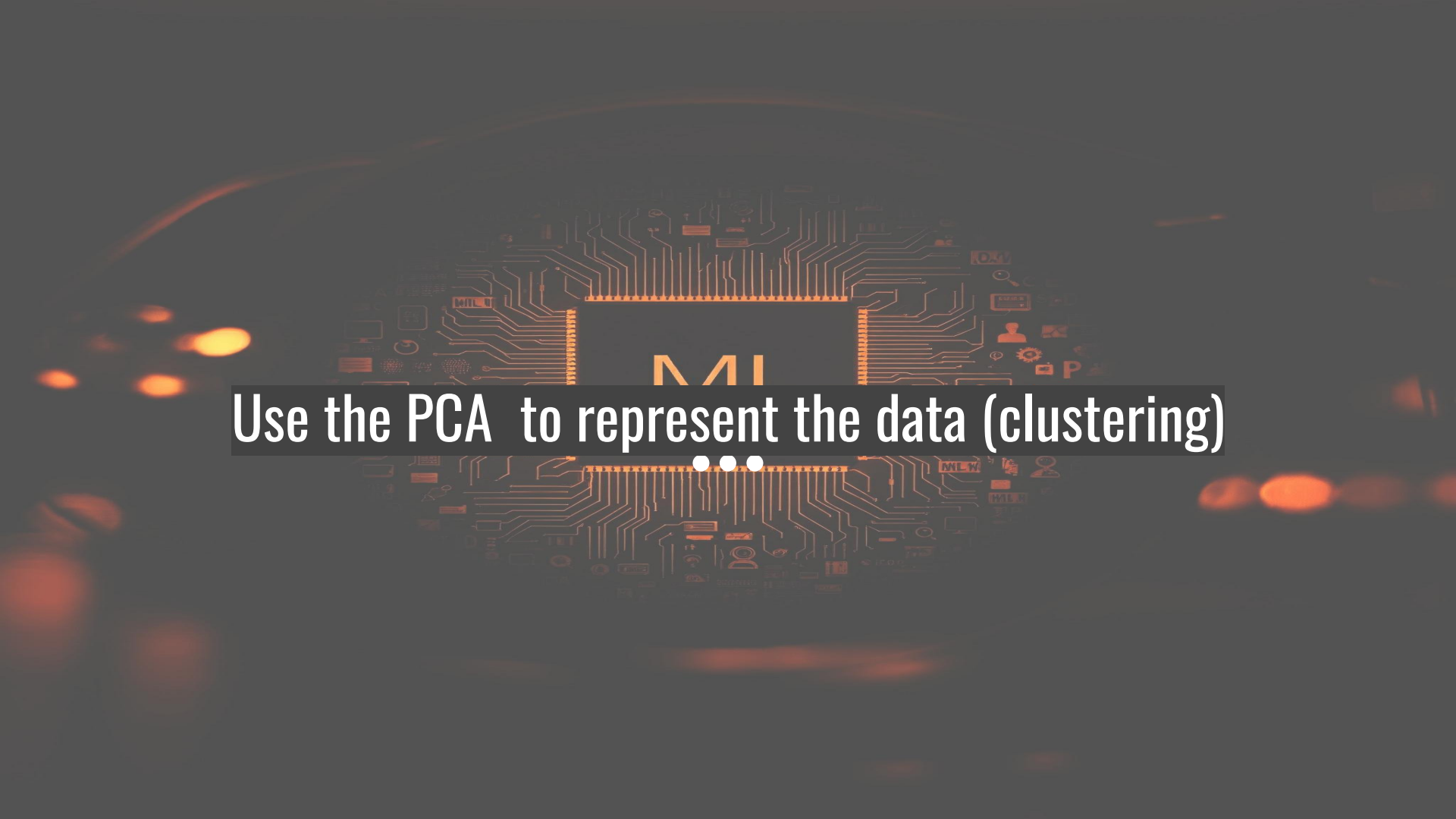
Run the PCA

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pcs = pca.fit_transform(x)
pcsdf2d = pd.DataFrame(data = pcs
                       , columns = ['PC 1', 'PC 2'])

pca = PCA(n_components=3)
pcs = pca.fit_transform(x)
pcsdf3d = pd.DataFrame(data = pcs
                       , columns = ['PC 1', 'PC 2', 'PC 3'])

finaldf2d = pd.concat([pcsdf2d, df[label]], axis = 1)
finaldf3d = pd.concat([pcsdf3d, df[label]], axis = 1)
print(finaldf2d.head())
print(finaldf3d.head())
```

Dimensionality Reduction:: PCA

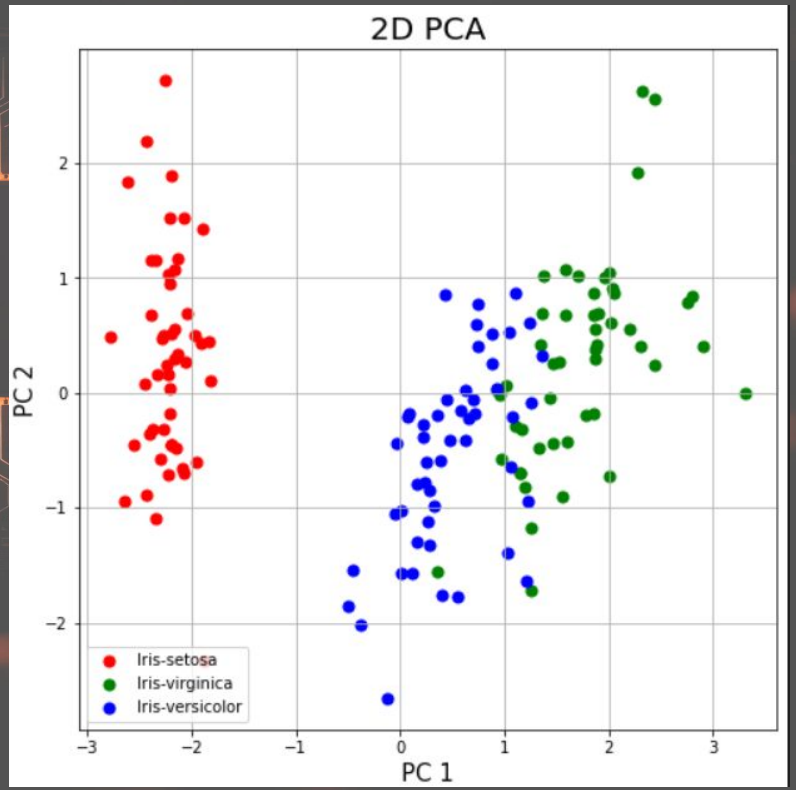
Run the PCA

```
        PC 1      PC 2       target
0 -2.264542  0.505704  Iris-setosa
1 -2.086426 -0.655405  Iris-setosa
2 -2.367950 -0.318477  Iris-setosa
3 -2.304197 -0.575368  Iris-setosa
4 -2.388777  0.674767  Iris-setosa
        PC 1      PC 2      PC 3       target
0 -2.264542  0.505704 -0.121943  Iris-setosa
1 -2.086426 -0.655405 -0.227251  Iris-setosa
2 -2.367950 -0.318477  0.051480  Iris-setosa
3 -2.304197 -0.575368  0.098860  Iris-setosa
4 -2.388777  0.674767  0.021428  Iris-setosa
```


Use the PCA to represent the data (clustering)
...

Dimensionality Reduction:: PCA

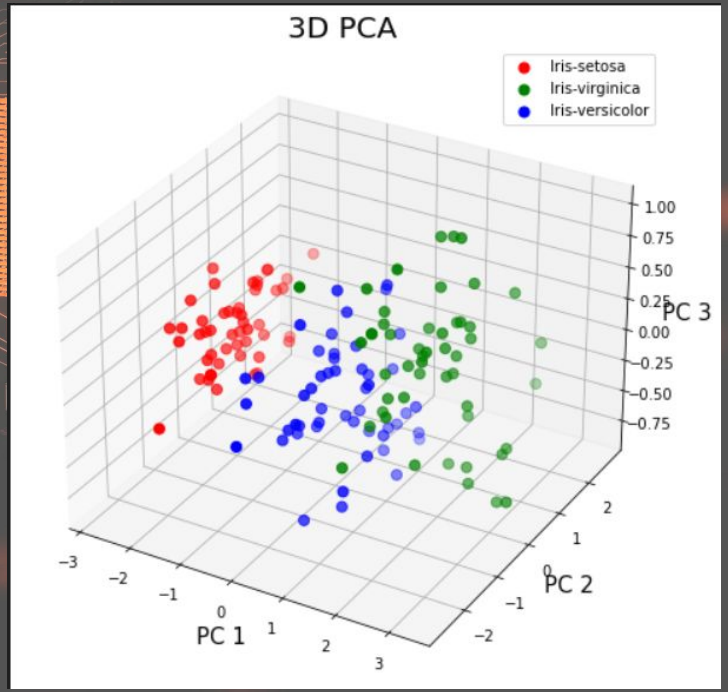
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_title('2D PCA', fontsize = 20)

stargets = set(df[label].values.flatten())
targets = list(stargets)
assert len(targets) == 3
colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):
    indices = finaldf2d[label[0]] == target
    ax.scatter(finaldf2d.loc[indices, 'PC 1']
               , finaldf2d.loc[indices, 'PC 2']
               , c = color
               , s = 50)
plt.legend(targets, loc='lower left')
ax.grid()
```



Dimensionality Reduction:: PCA

```
# 3D plot of the PCA
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_zlabel('PC 3', fontsize = 15)
ax.set_title('3D PCA', fontsize = 20)

for target, color in zip(targets,colors):
    indices = finaldf3d[label[0]] == target
    ax.scatter(finaldf3d.loc[indices, 'PC 1']
               , finaldf3d.loc[indices, 'PC 2']
               , finaldf3d.loc[indices, 'PC 3']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```

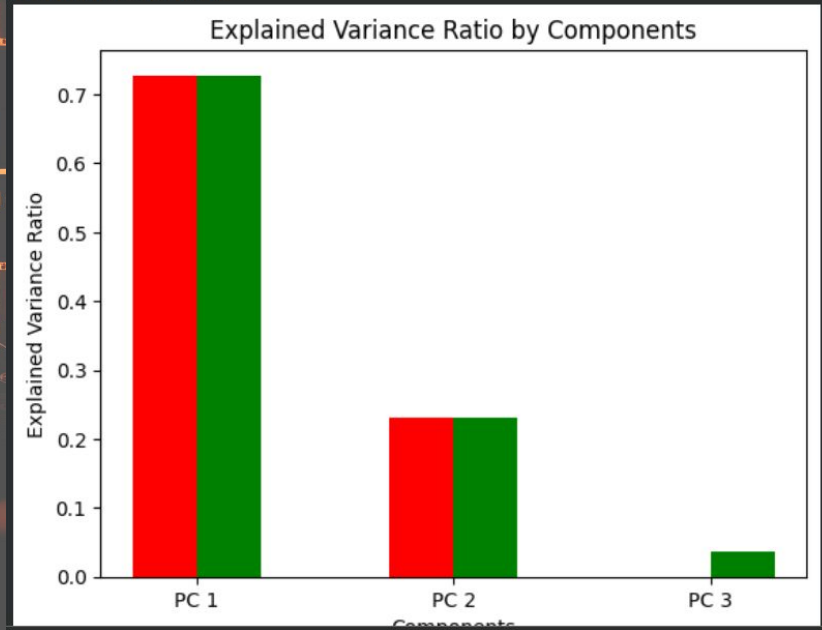
✓ 0.0s

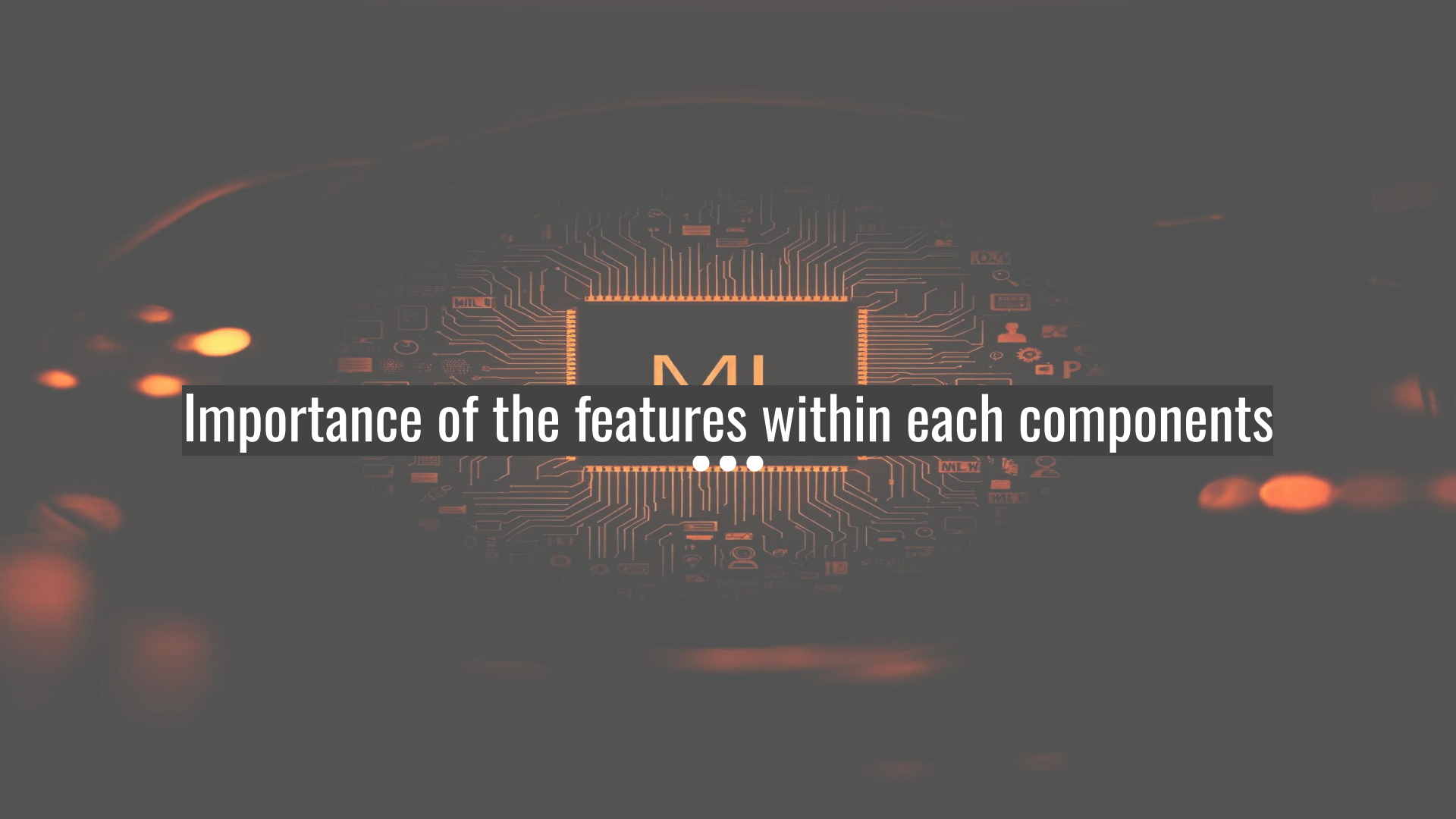


Dimensionality Reduction:: PCA

```
print(pca2d.explained_variance_ratio_)
print(pca2d.explained_variance_ratio_.sum())

print(pca3d.explained_variance_ratio_)
print(pca3d.explained_variance_ratio_.sum())

N = 3
ind = np.arange(1,N+1)
width = 0.25

d2vals = list(pca2d.explained_variance_ratio_)
d2vals.extend([0])
plt.bar(ind, d2vals, width, color = 'r')
d3vals = pca3d.explained_variance_ratio_
plt.bar(ind+width, d3vals, width, color='g')

plt.xlabel("Components")
plt.ylabel('Explained Variance Ratio')
plt.title("Explained Variance Ratio by Components")

plt.xticks(ind+(width/2),['PC 1', "PC 2", "PC 3"])
plt.show()
```

```
[0.72770452 0.23030523]
0.9580097536148199
[0.72770452 0.23030523 0.03683832]
0.9948480731910938
```




Importance of the features within each components

Dimensionality Reduction:: PCA

Component-specific importance: This method provides the importance of each feature within a specific principal component. A feature might be important in one PC but less important in another

```
# print the coefficients of the first two PC
print(pca2d.components_)
for i in range(pca2d.components_.shape[0]):
    print("PC " , i+1, "=  ")
    for j in range(pca2d.components_.shape[1]):
        print( "  %6.2f"%(pca2d.components_[i,j]**2) ,\
          " * ", features[j])
```

Dimensionality Reduction:: PCA

Component-specific importance: This method provides the importance of each fe… might be importa…

```
# print
print(p
for i i
    pri
    for
```

```
[[ 0.52237162 -0.26335492  0.58125401  0.56561105]
 [ 0.37231836  0.92555649  0.02109478  0.06541577]]
PC  1 =
    0.27  *  sepal length
    0.07  *  sepal width
    0.34  *  petal length
    0.32  *  petal width
PC  2 =
    0.14  *  sepal length
    0.86  *  sepal width
    0.00  *  petal length
    0.00  *  petal width
```

Dimensionality reduction when using a Supervised technique
...

Dimensionality Reduction:: PCA

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

x_train, x_test, y_train, y_test = train_test_split(\
    x, y, test_size=0.2, random_state=0)
trainsamples = x_train.shape[0]
testsamples = x_test.shape[0]
logreg = LogisticRegression()
print(x_train.shape)
print(y_train.shape)
logreg.fit(x_train, y_train.reshape(trainsamples))
y_pred = logreg.predict(x_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(\
    logreg.score(x_test, y_test.reshape(testsamples))))
```

Dimensionality Reduction:: PCA

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

x_train, x_test, y_train, y_test = train_test_split(\
    x, y, test_size=0.2, random_state=0)
trainsamples = x_train.shape[0]
testsamples = x_test.shape[0]
logreg = LogisticRegression()
print(x_train.shape)
print(y_train.shape)
logreg.fit(x_train, y_train.reshape(trainsamples))
y_p
pri
```

```
(120, 4)
(120, 1)
Accuracy of logistic regression classifier on test set: 1.00
```

Dimensionality Reduction:: PCA

```
for n in range(1, x.shape[1]+1):
    x = df.loc[:, features].values
    y = df.loc[:,label].values
    pca = PCA(n_components=n)
    x_t = pca.fit_transform(StandardScaler().fit_transform(x))
    x_train, x_test, y_train, y_test = train_test_split(\
        x_t, y, test_size=0.2, random_state=0)
    trainsamples = x_train.shape[0]
    testsamples = x_test.shape[0]
    print(x_train.shape)
    print(y_train.shape)
    logreg = LogisticRegression()
    logreg.fit(x_train, y_train.reshape(trainsamples))
    y_pred = logreg.predict(x_test)
    print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(
        logreg.score(x_test, y_test.reshape(testsamples))))
```

Dimensionality Reduction:: PCA

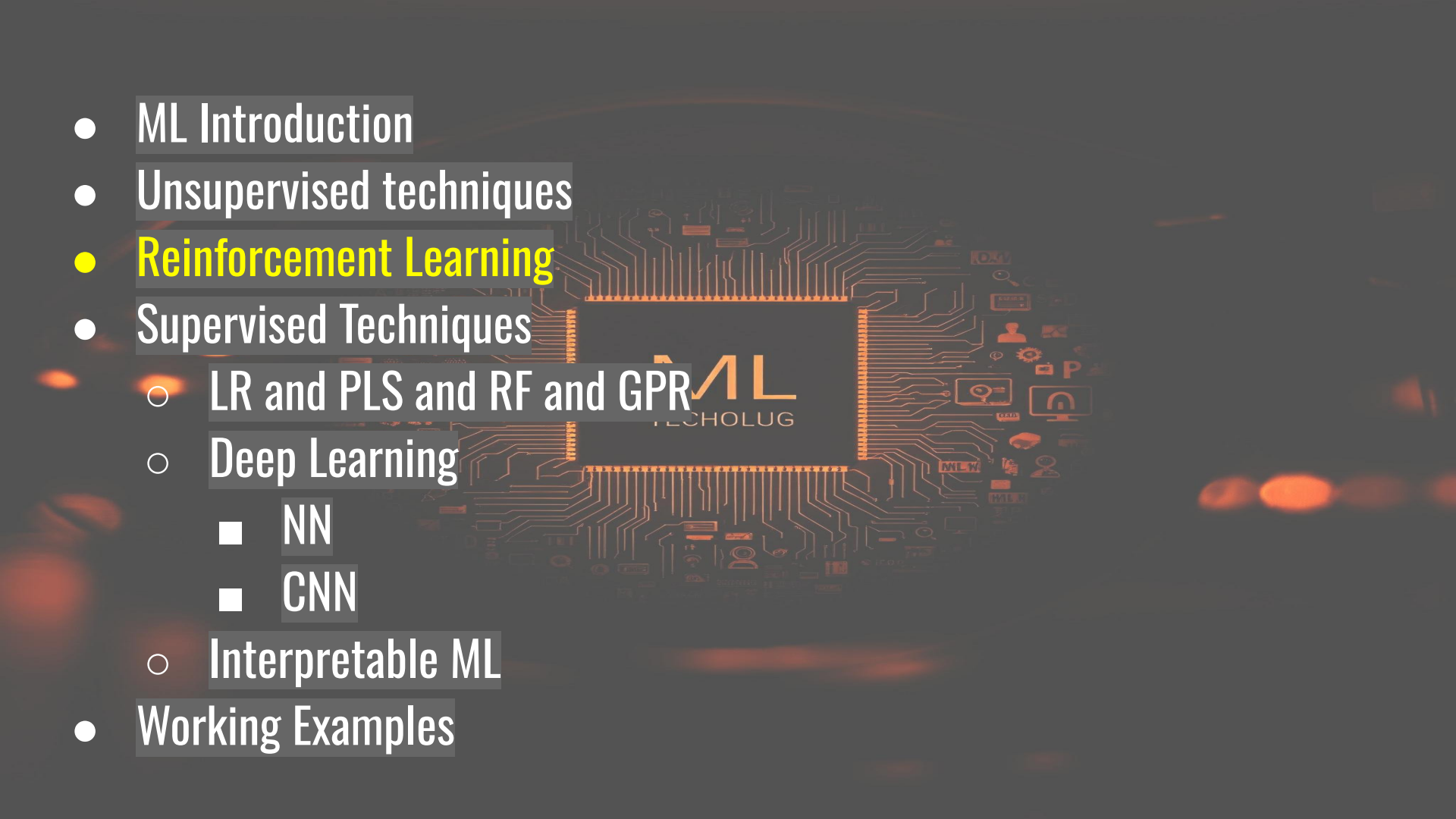
```
for n in range(1, x.shape[1]+1):
    x = df.loc[:, features].values
    y = df.loc[:,label].values
    pca = PCA(n_components=n)
    x_t = pca.fit_transform(StandardScaler().fit_transform(x))
    x_train, x_test, y_train, y_test = train_test_split(\
        x_t, y, test_size=0.2, random_state=0)
    trainsamples = x_train.shape[0]
    testsamples = x_test.shape[0]
    print(x_train.shape)
    print(y_train.shape)
    logreg = LogisticRegression()
    logreg.fit(x_train, y_train.reshape(trainsamples))
    y_pred = logreg.predict(x_test)
    print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(
        logreg.score(x_test, y_test.reshape(testsamples))))
```

Not the best option as we should scale, train the PCA only on the training set . See the afternoon exercise

Dimensionality Reduction:: PCA

```
for n in range(1, x.shape[1]+1):
    x = df.loc[:, features].values
    y = df.loc[:,label].values
    pca = P
    x_t = p
    x_train
        x_t
    trainsa
    testsam
    print(x
    print(y
    logreg
    logreg.
    y_pred
    print('
        log
```

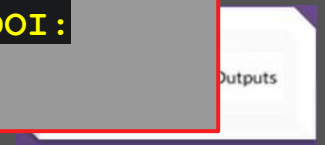
```
(120, 1)
(120, 1)
Accuracy of logistic regression classifier on test set: 0.87
(120, 2)
(120, 1)
Accuracy of logistic regression classifier on test set: 0.87
(120, 3)
(120, 1)
Accuracy of logistic regression classifier on test set: 1.00
(120, 4)
(120, 1)
Accuracy of logistic regression classifier on test set: 1.00
```



- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

Machine Learning

Tommaso Tedeschi, Marco Baioletti, Diego Ciangottini, Valentina Poggioni, Daniele Spiga, Loriano Storchi, Mirco Tracolli, "Smart Caching in a Data Lake for High Energy Physics Analysis", Journal of Grid Computing, DOI: 10.1007/s10723-023-09664-z (2023)



algorithms)

- **Reinforcement Learning**: the algorithms learn to react to an environment on their own. An agent is in a situation of trial and error, where the consequences of its actions have an impact on the environment and also on the problem's goal. The agent is punished or rewarded on the basis of its behavior, with the idea that, in the future, it will prefer optimal actions (i.e. our intelligent cache system)

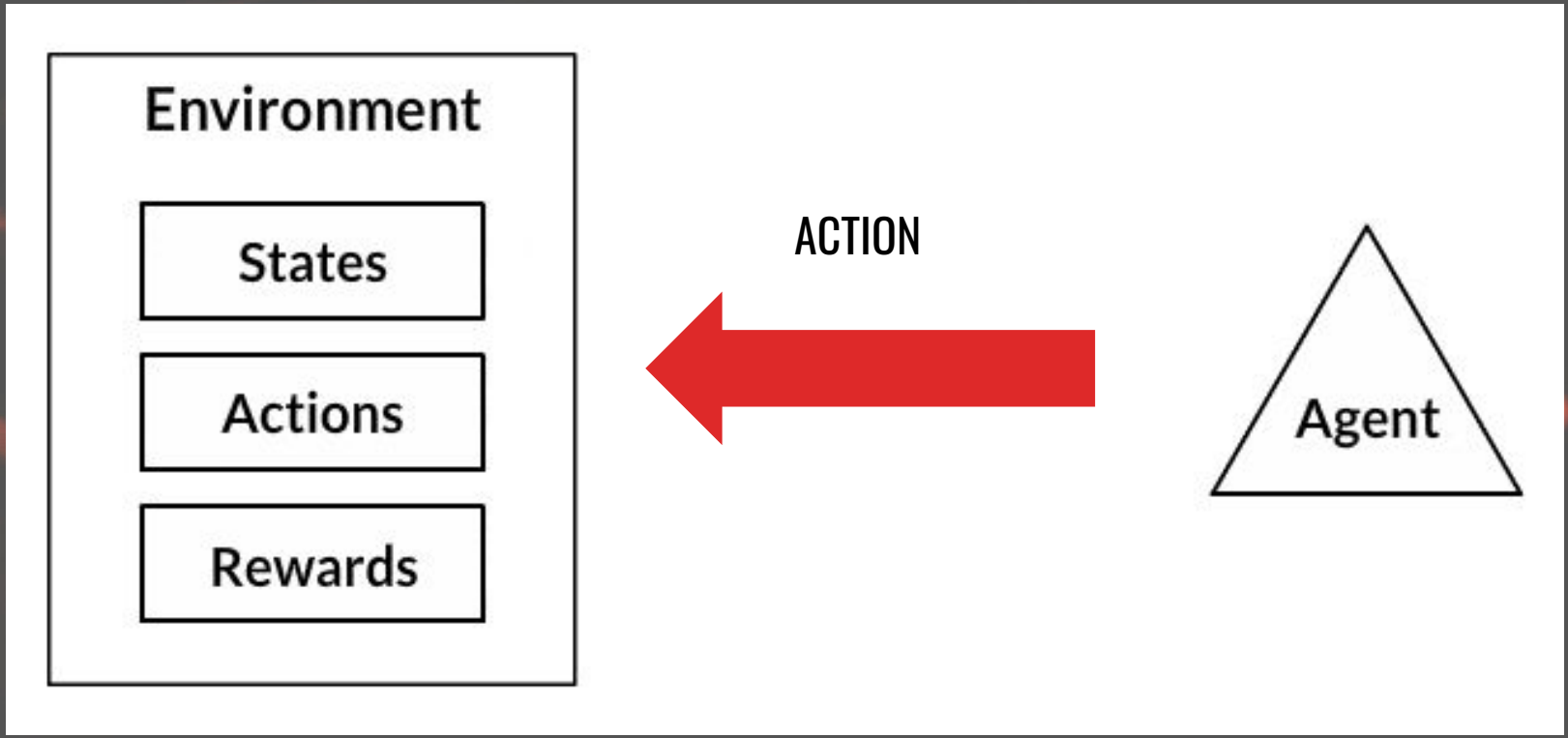
- **Supervised**: used when you want to predict or explain the data you possess. A supervised algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions

Reinforcement Learning: Q-Learning

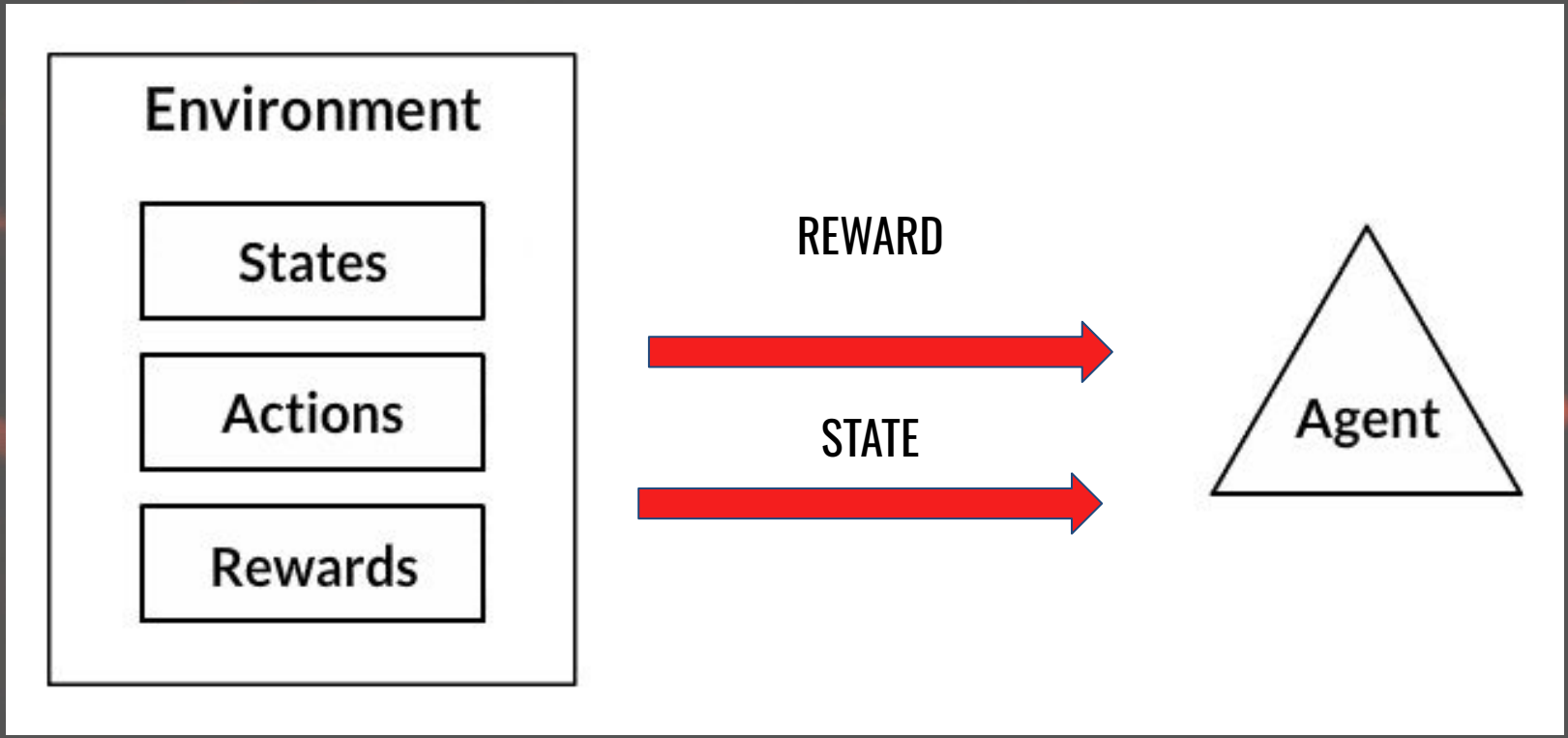
Reinforcement Learning lies between the spectrum of Supervised Learning and Unsupervised Learning,. How does Reinforcement Learning work in a broader sense ?

- An "**agent**" is exposed to the environment
- The situations they encounter are **states**
- Our agents react by performing an **action** to transition from one "**state**" to another "**state**,"
- After the transition, they may receive a **reward** or **penalty** in return
- The **policy** is the strategy of choosing an action given a state in expectation of better outcomes.

Reinforcement Learning: Q-Learning



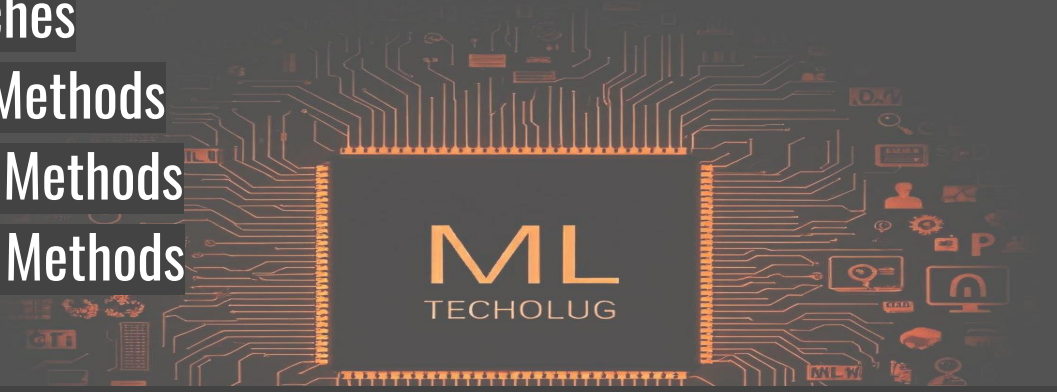
Reinforcement Learning: Q-Learning



Reinforcement Learning: Q-Learning

Several approaches

1. Value-Based Methods
2. Policy-Based Methods
3. Model-Based Methods

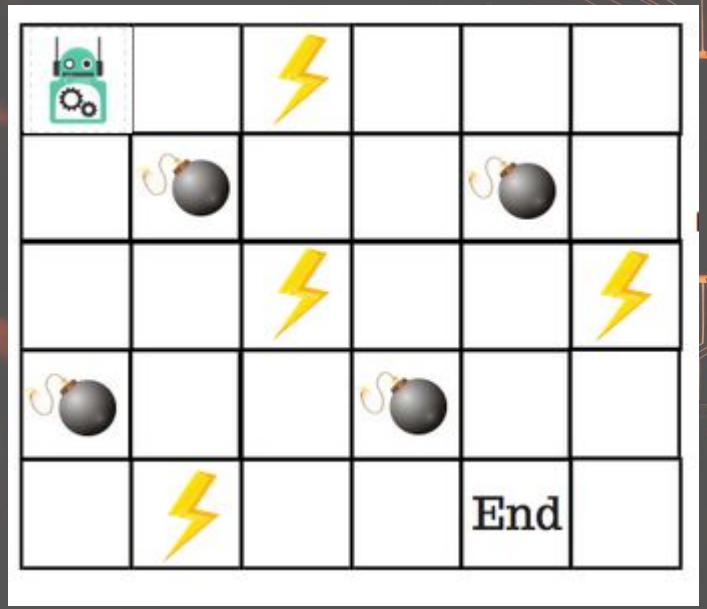


I will give you a quick overview about one of the Value-Based Methods that is the:

Q-Learning: A model-free approach where an agent learns an action-value function (Q-function) that estimates the expected reward for taking a given action in a given state.

Reinforcement Learning: Q-Learning

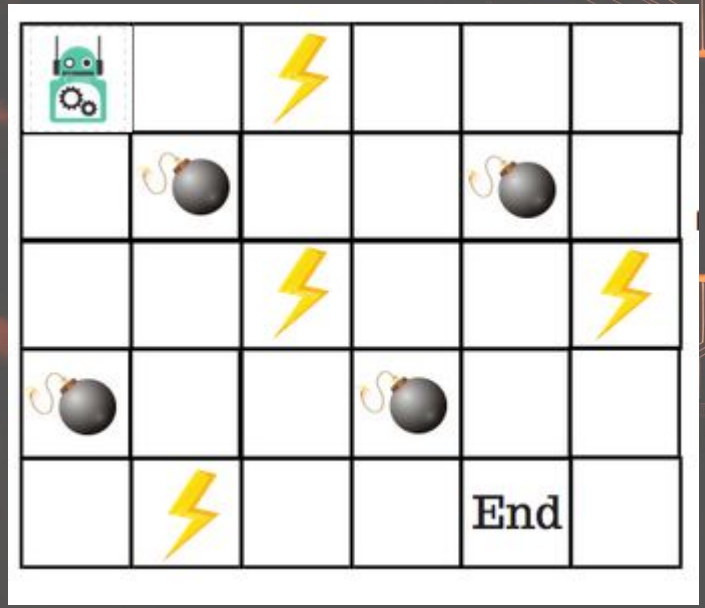
Simple example (no code)



A robot has to cross a maze and reach the end point. There are mines, and the robot can only move one tile at a time. If the robot steps onto a mine, the robot is dead. The robot has to reach the end point in the shortest time possible.

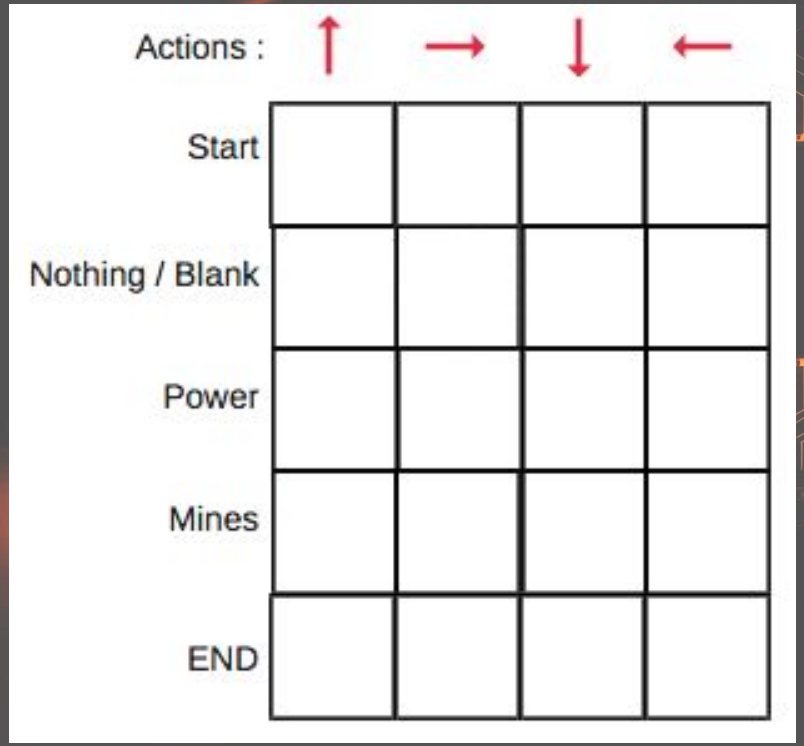
Reinforcement Learning: Q-Learning

Simple example (no code)



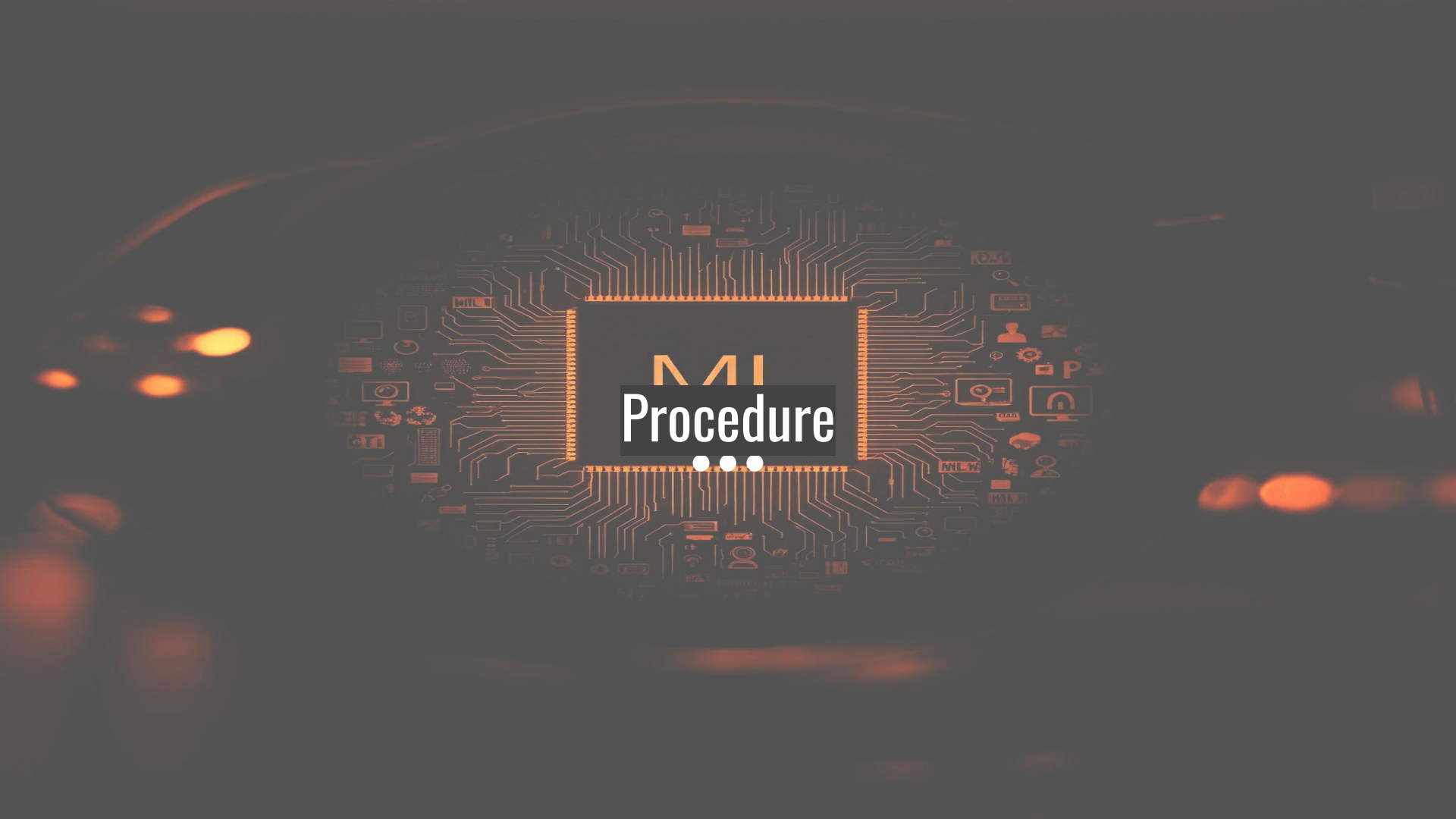
- The robot loses 1 point at each step. (force the robot to take the shortest path).
- Mine, the point loss is 100 and the game ends.
- If the robot gets power it gains 1 point.
- If the robot reaches the end goal, the robot gets 100 points.

Reinforcement Learning: Q-Learning

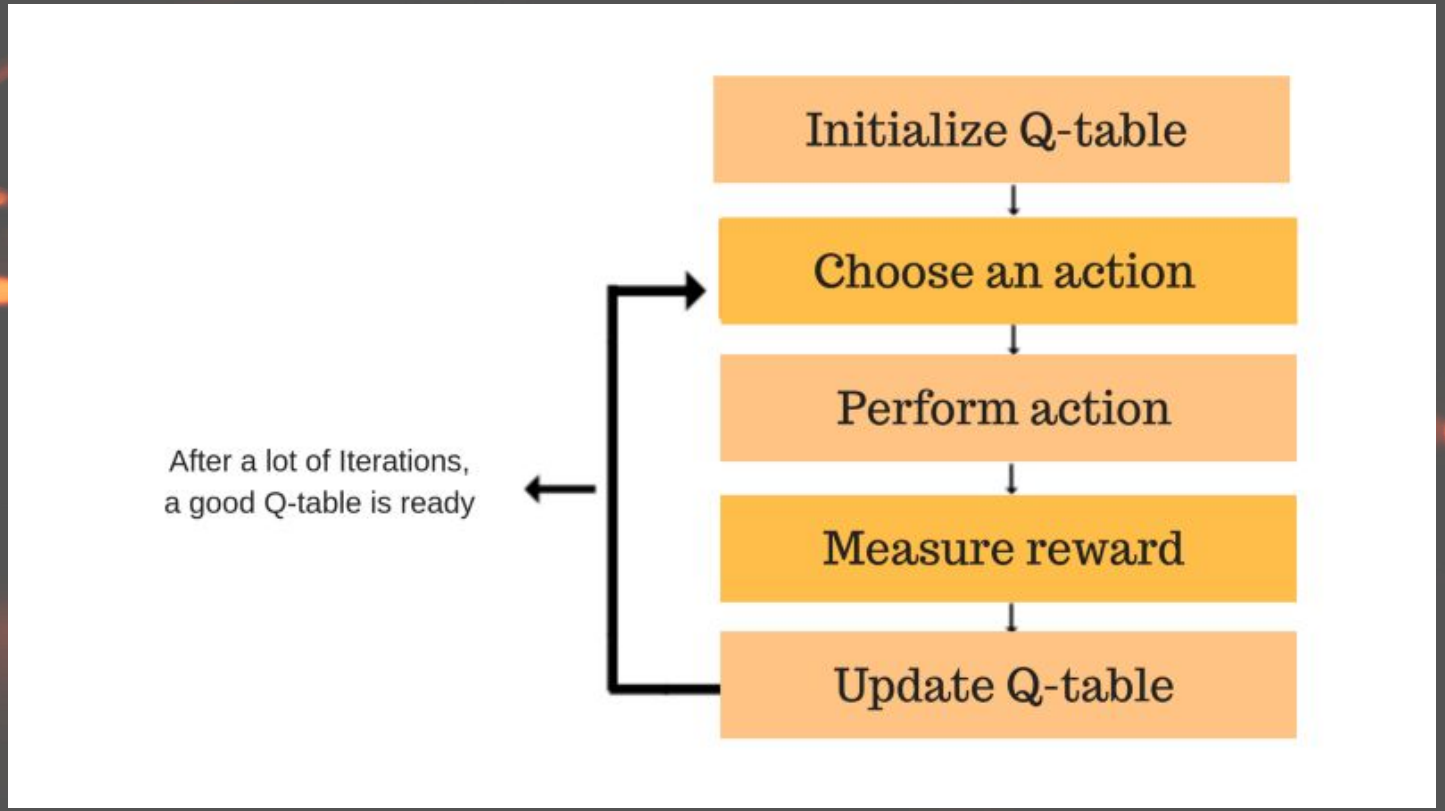


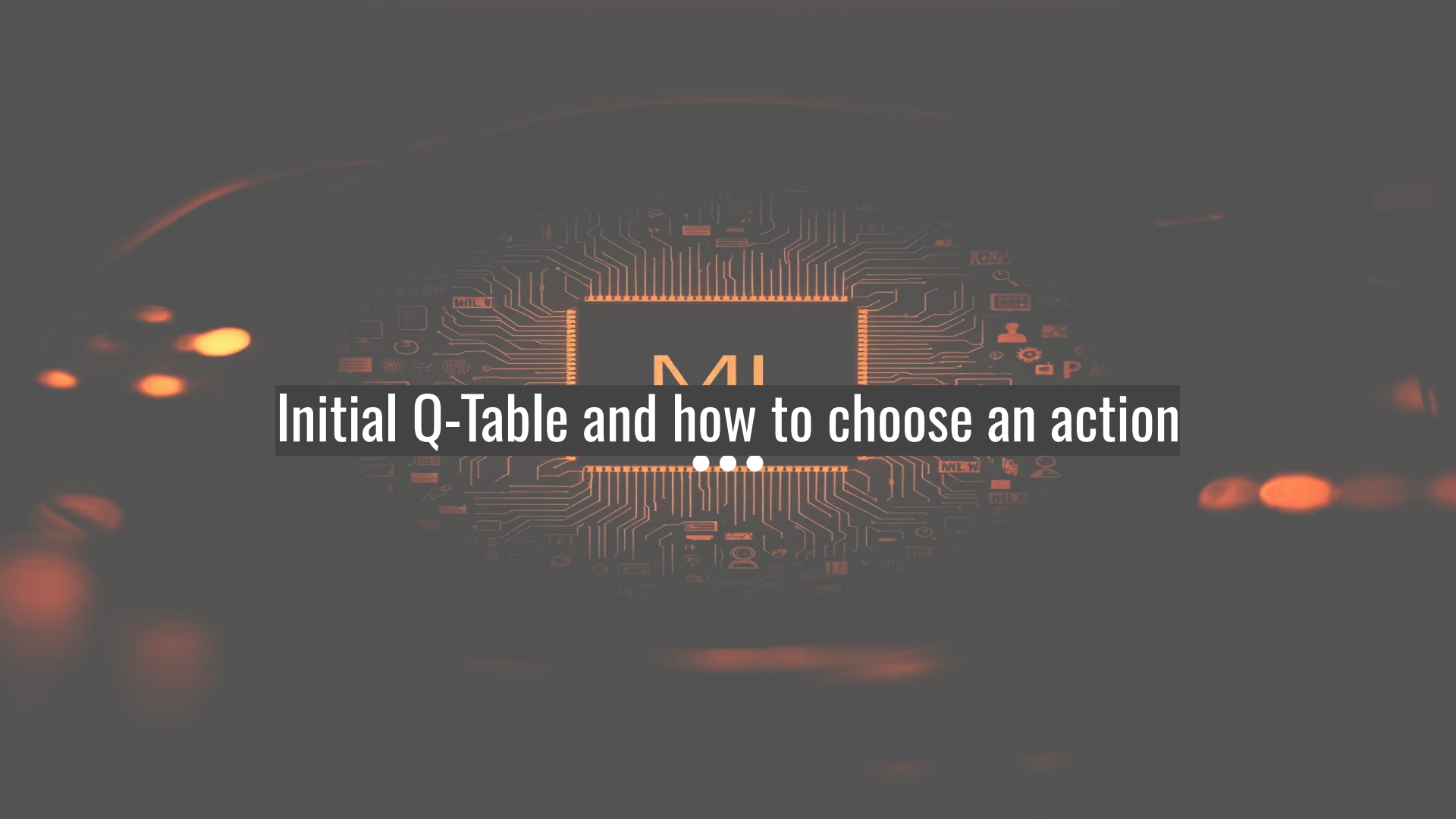
The Q-Table, the **columns are the actions and the rows are the states.**

- **Actions Space: 4 possible actions move up, down, left or right**
- **States space: 5 state start, nothing (blank square), power, mine, end**
- **Rewards: loss 1 for each step, loss 100 for mine, gain 100 for end , gain 1 for power**

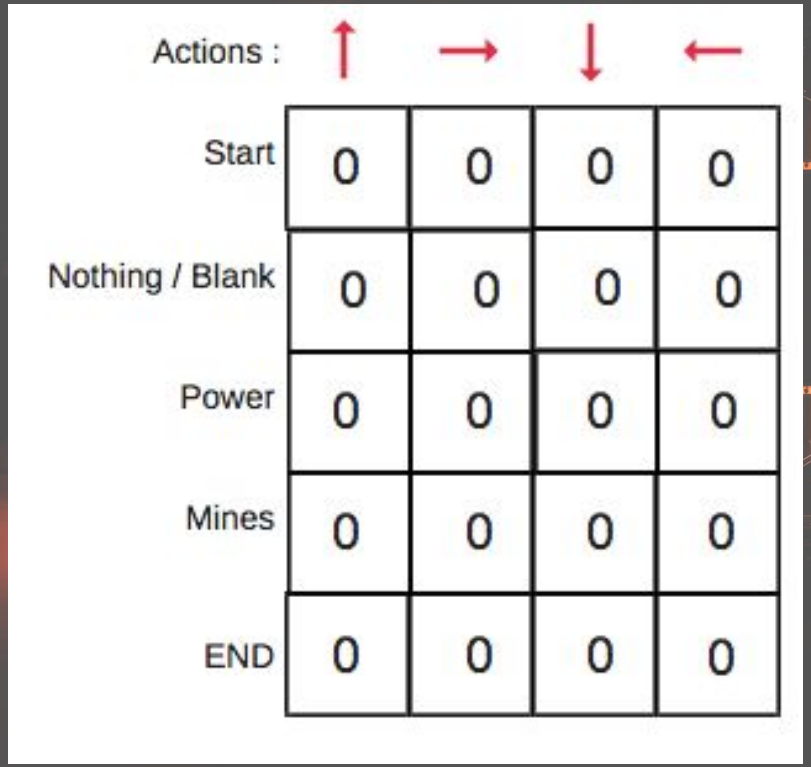

Procedure

Reinforcement Learning: Q-Learning




Initial Q-Table and how to choose an action
...

Reinforcement Learning: Q-Learning



Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

- Initialize the Q values:, randomly, in this example we will initialize all values to zero
- Choose an action (a) in the state (s) based on the Q-Table
  - One can use different strategies to select the best action
  - in this case the cation is chosen randomly using epsilon greedy strategy

Reinforcement Learning: Q-Learning

if random() < ε
random action
otherwise
action = argmax(Q(state, a)) for all actions a

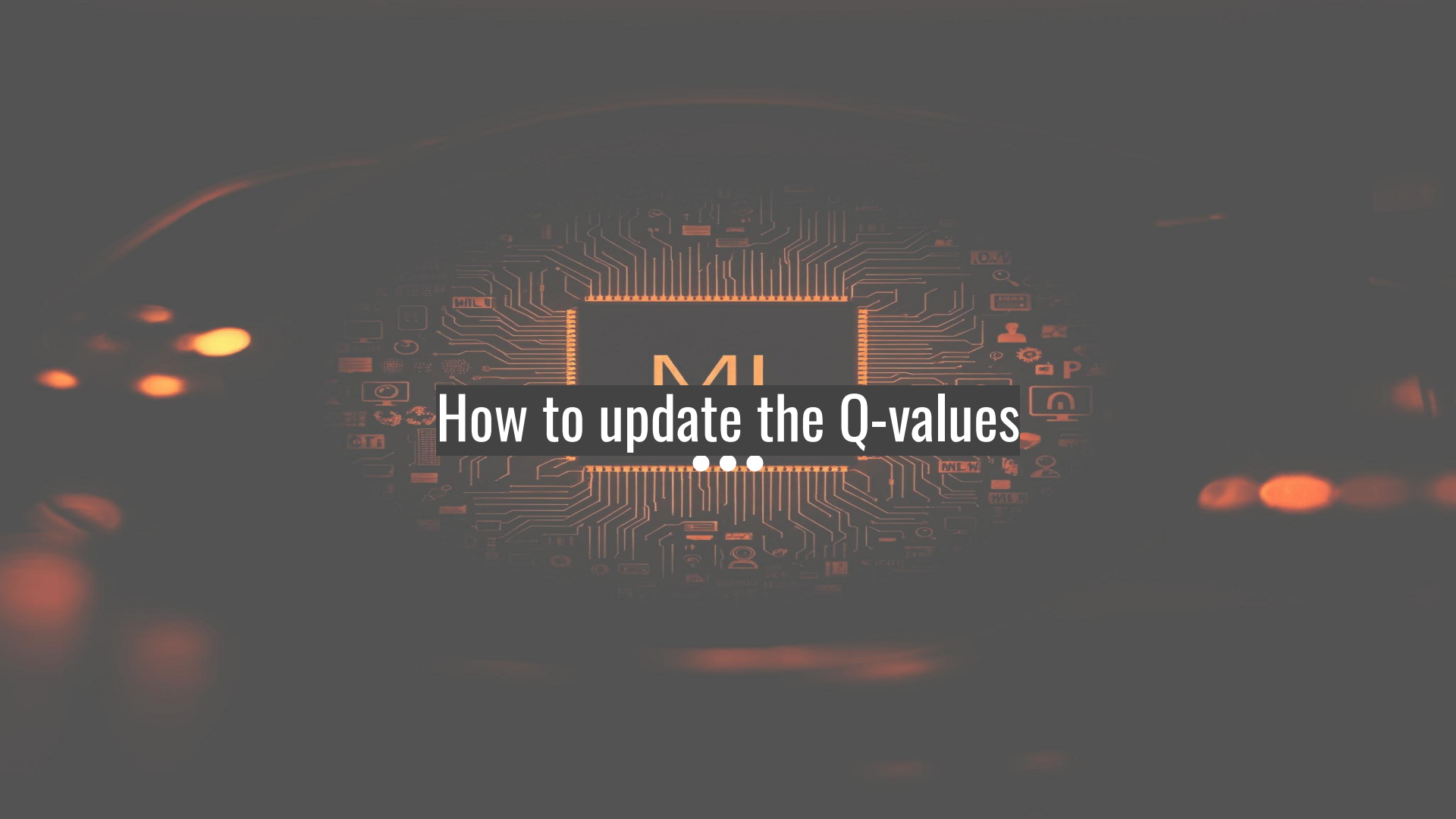
argmax(Q(state, a)) for all actions a: **This part calculates the action with the highest Q-value in the current state**

Reinforcement Learning: Q-Learning

Actions :	↑	→	↓	←
Start	5	10	0	-2
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

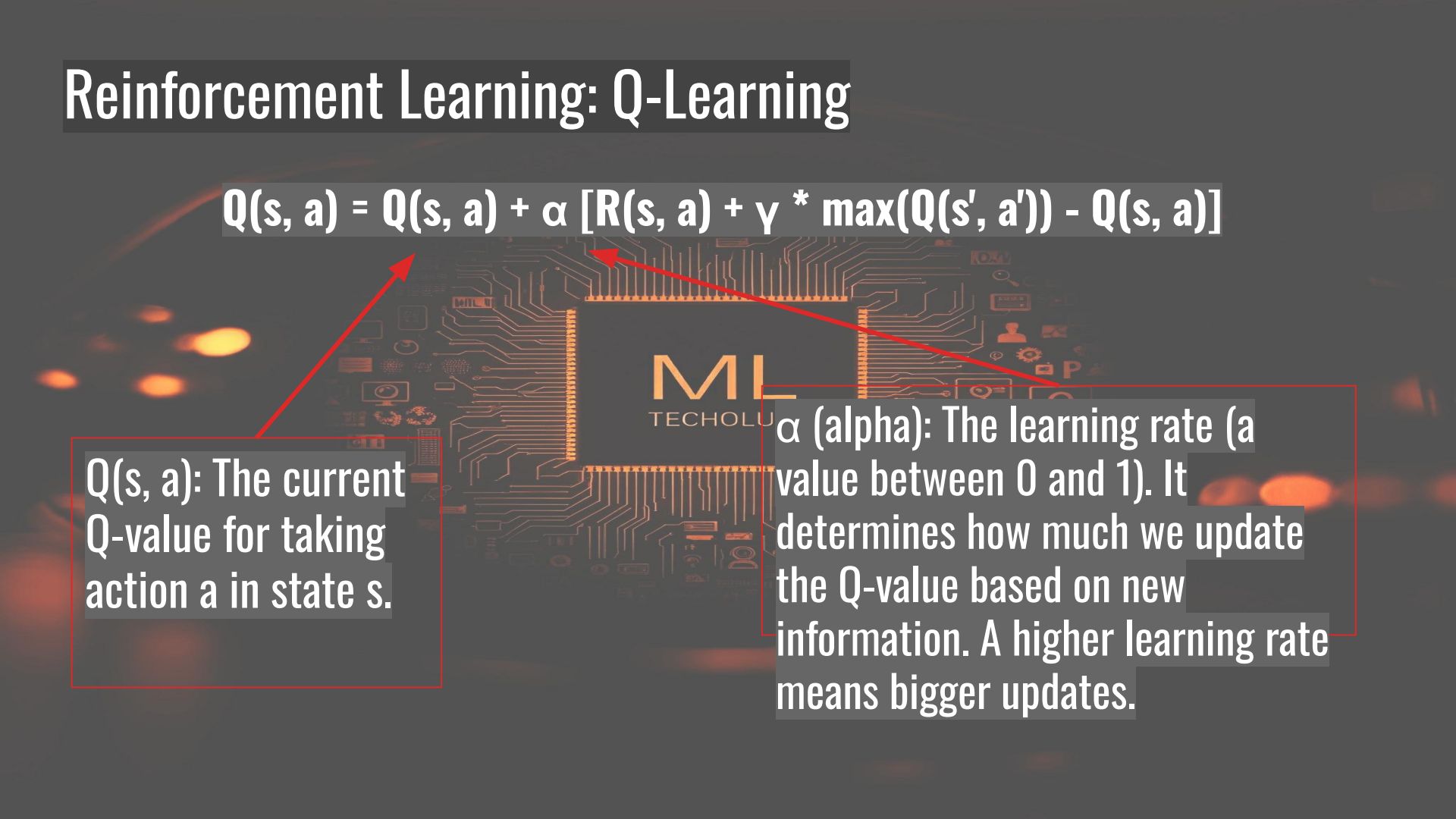
if ε = 0.1 (10% chance of exploration)., so 90% of times we select the best action accordingly to argmax

if we are in state 0 = Start in this case clearly the best action is Right


How to update the Q-values
...

Reinforcement Learning: Q-Learning

$$Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma * \max(Q(s', a')) - Q(s, a)]$$

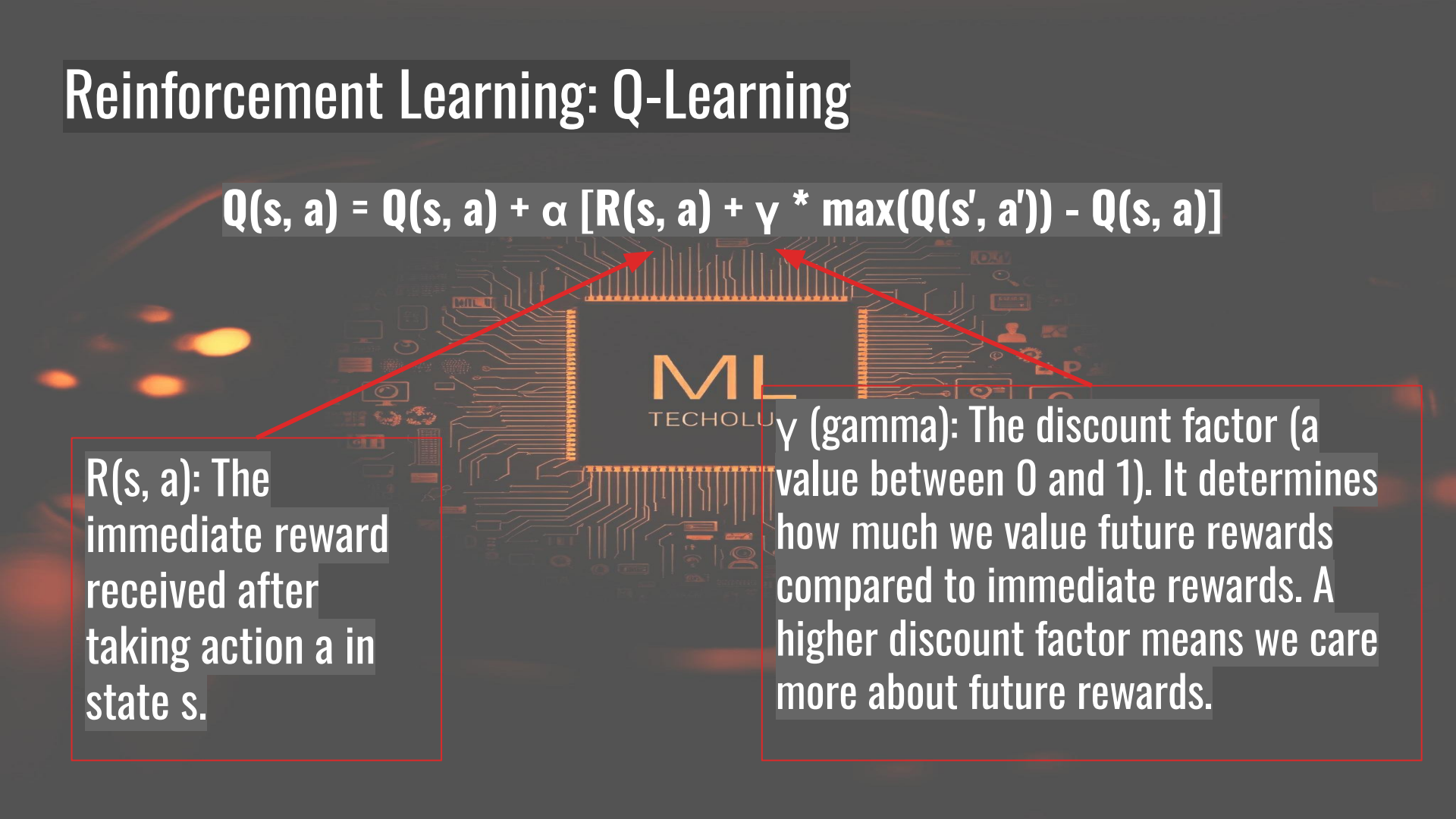


Q(s, a): The current Q-value for taking action a in state s.

α (alpha): The learning rate (a value between 0 and 1). It determines how much we update the Q-value based on new information. A higher learning rate means bigger updates.

Reinforcement Learning: Q-Learning

$$Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma * \max(Q(s', a')) - Q(s, a)]$$

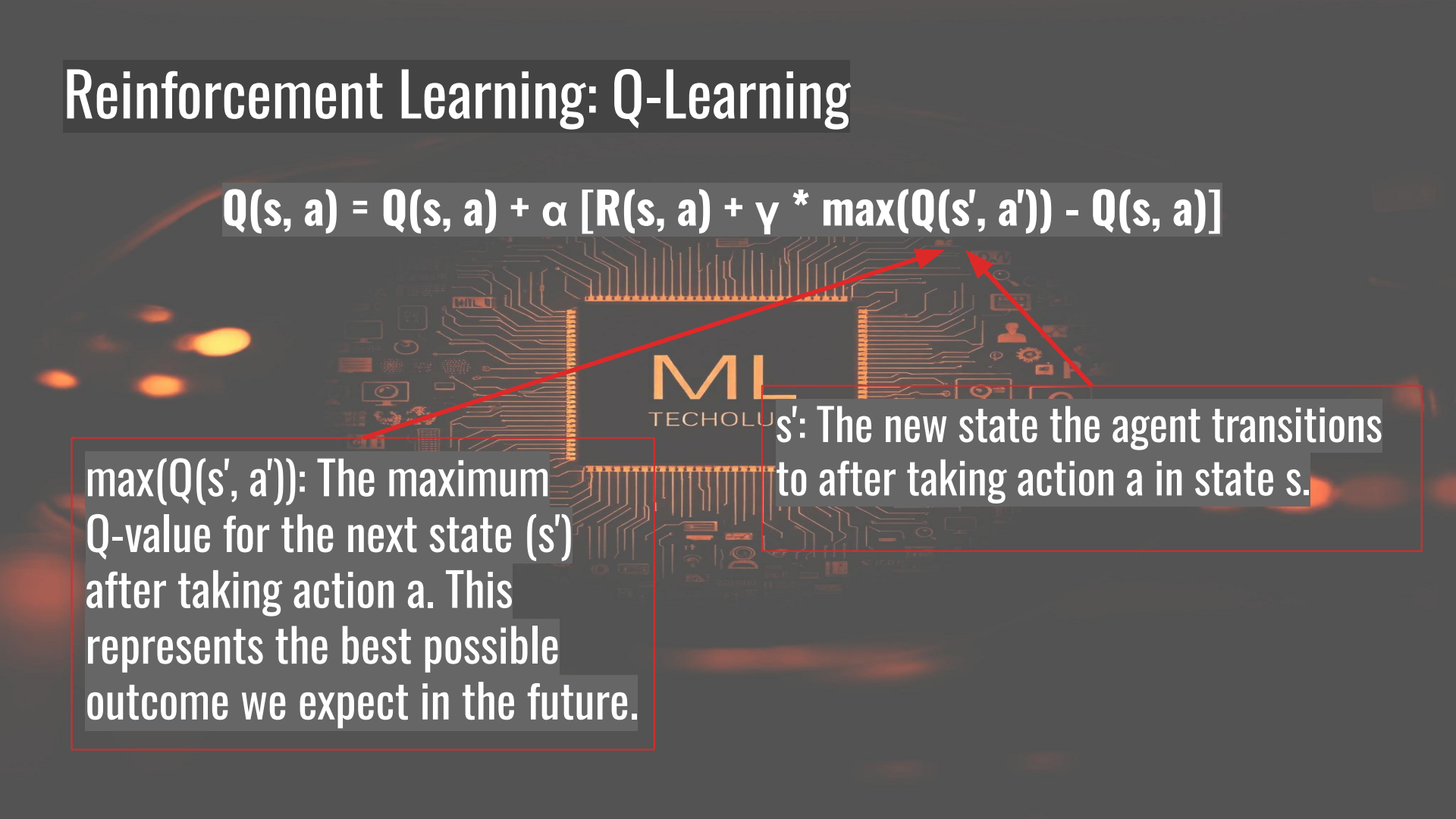


R(s, a): The immediate reward received after taking action a in state s.

γ (gamma): The discount factor (a value between 0 and 1). It determines how much we value future rewards compared to immediate rewards. A higher discount factor means we care more about future rewards.

Reinforcement Learning: Q-Learning

$$Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma * \max(Q(s', a')) - Q(s, a)]$$



max(Q(s', a')): The maximum Q-value for the next state (s') after taking action a. This represents the best possible outcome we expect in the future.

s': The new state the agent transitions to after taking action a in state s.

Reinforcement Learning: Q-Learning

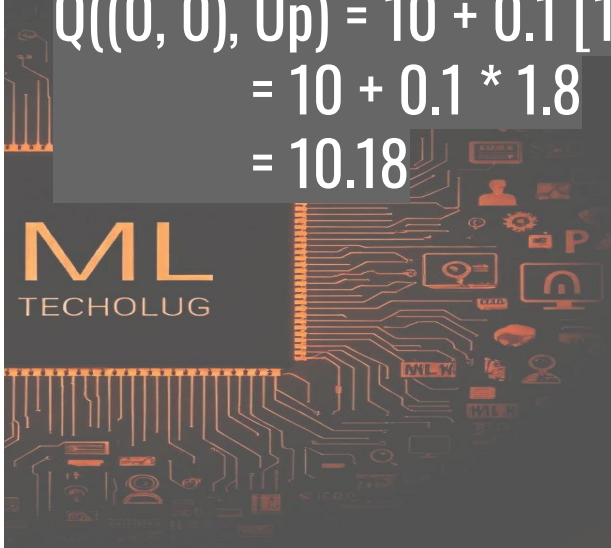
Actions :	↑	→	↓	←
State 0	10	5	-2	8
State 1	-5	12	7	-1
State 2	3	6	15	0
State 3	-10	-8	-5	1
State 4	5	10	10	-2

Current state (s): 0
Action (a): Up
Reward (R(s, a)): 1 (let's say it gets a small reward for moving up)
Next state (s'): 1
Learning rate (α): 0.1
Discount factor (γ): 0.9
Current Q-value (Q(s, a)): 10 (from the Q-table)
max(Q(s', a')): 12 (the highest Q-value in state 1 is for action "Right")

Reinforcement Learning: Q-Learning

Actions :	↑	→	↓	←
State 0	10.18	5	-2	8
State 1	-5	12	7	-1
State 2	3	6	15	0
State 3	-10	-8	-5	1
State 4	5	10	10	-2

$$Q((0, 0), Up) = 10 + 0.1\,[1 + 0.9 * 12 - 10]$$
$$= 10 + 0.1 * 1.8$$
$$= 10.18$$

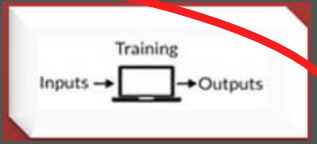


- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

Machine Learning

Machine learning techniques can be divided into two foremost types:

- **Unsupervised**: find hidden patterns or intrinsic structures in data. They are used to draw inferences from data sets consisting of input data without labeled responses (i.e. clustering algorithms)
- **Reinforcement Learning**: the algorithms learn to react to an environment on their own. An agent is in a situation of trial and error, where the consequences of its actions have an impact on the environment and also on the problem's goal. The agent is punished or rewarded on the basis of its behavior, with the idea that, in the future, it will prefer optimal actions (i.e. our intelligent cache system)
- **Supervised**: used when you want to predict or explain the data you possess. A supervised algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions



Supervised Machine Learning

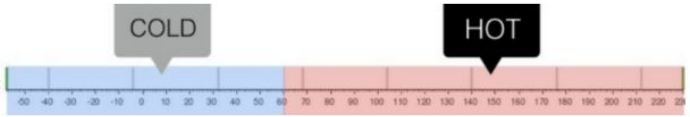
Regression

What will be the temperature tomorrow?

Fahrenheit

Classification

Will it be hot or cold tomorrow?



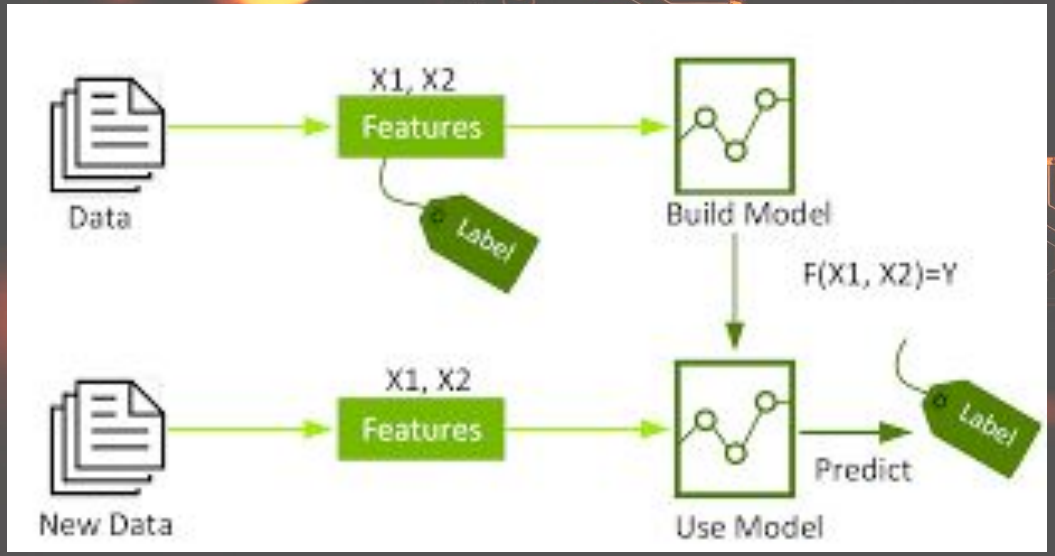
Fahrenheit

Features could be: the day of the year and the today temperature

Label: is the temperature for the regression and

Machine Learning

Supervised: used when you want to predict or explain the data you possess. A supervised algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions

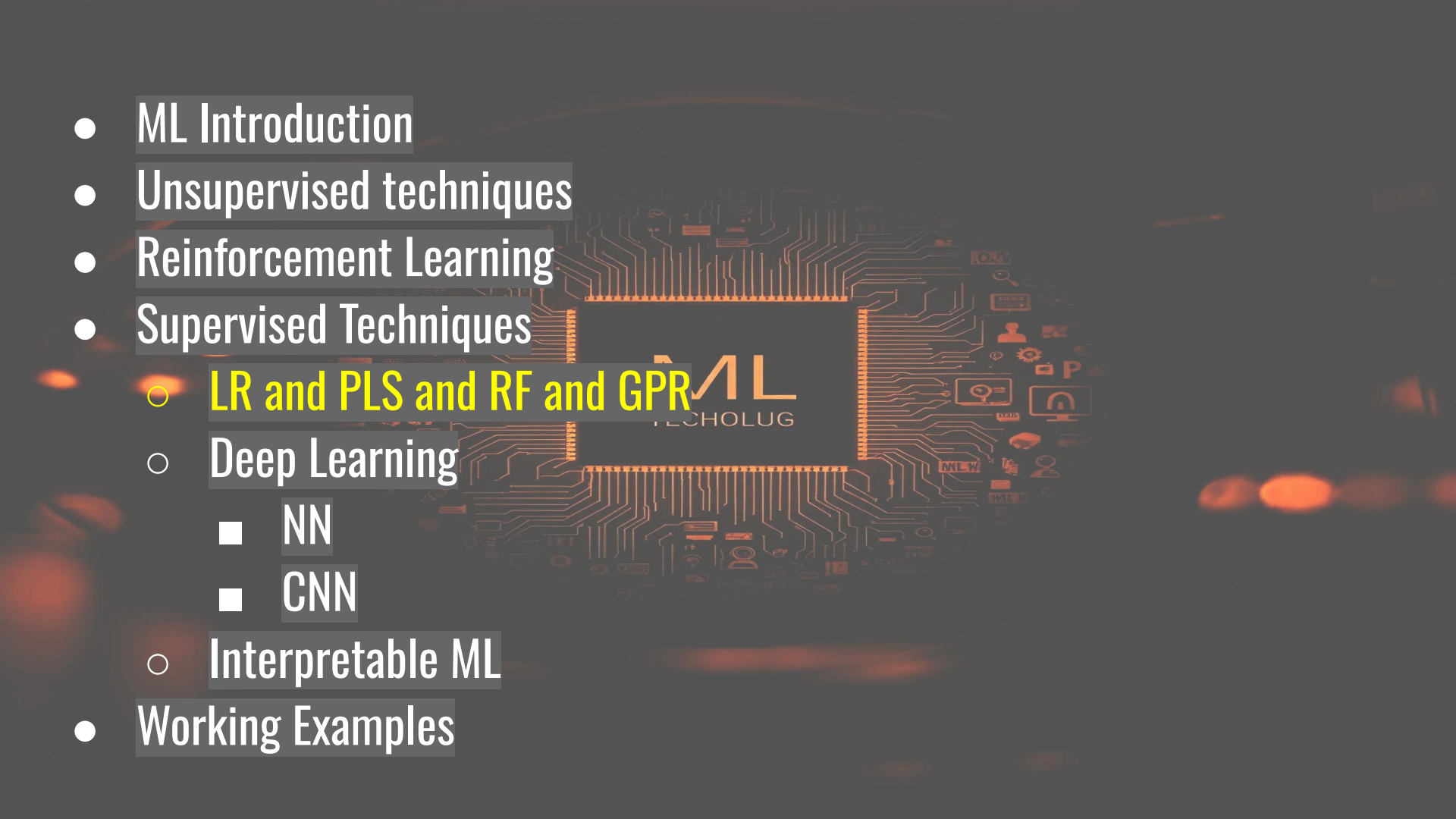


$$Y = F_{a,b,c}(X)$$

Labels: dependent variables (e.g. pK_a values , could be also a class pass or not the BBB)

Features (descriptors): independent variables (e.g. Molecular weight, fingerprints)

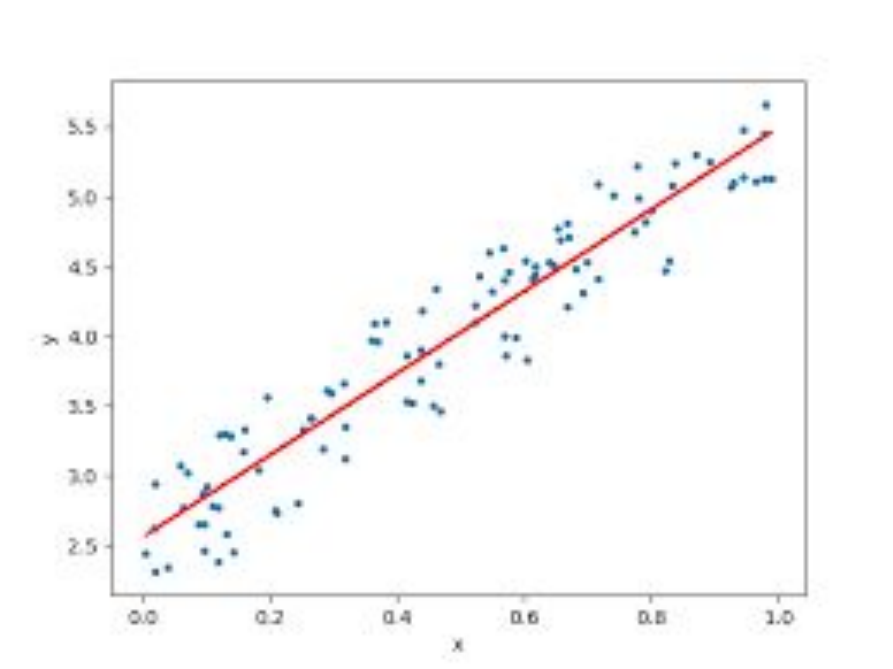
Models: Linear Regression, Random Forest, Artificial Neural Network , Partial Leat Square



- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

ML
TEC
LR
UG

Linear Regression



Implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and $\mathbf{x}$: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$.

Linear Regression

```
df_sal = pd.read_csv('Salary_Data.csv')
df_sal.head()
```

✓ 0.0s Open 'df_sal' in Data Wrangler

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

Linear Regression

```
plt.scatter(df_sal['YearsExperience'], \
            df_sal['Salary'], color = 'lightcoral')
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

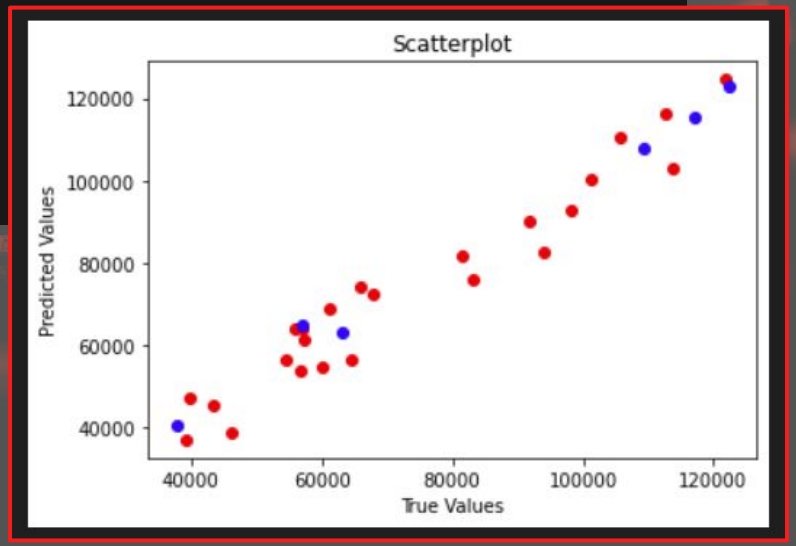
Linear Regression

```
X = df_sal.iloc[:, :1]
y = df_sal.iloc[:, 1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, \
                                   test_size = 0.2, random_state = 0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
y_pred_test = regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)
```

```
Coefficient: [[9312.57512673]]
Intercept: [26780.09915063]
```

Linear Regression

```
plt.scatter(y_train, y_pred_train, color = 'red')
plt.scatter(y_test, y_pred_test, color='blue')
plt.title('Scatterplot')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.show()
```



Linear Regression

$$R^2 = \frac{\text{RegSS}}{\text{TSS}} = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}$$

R^2 measures how well the regression line fits the data points.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE stands for Root Mean Squared Error. It's a common metric used to evaluate the accuracy of a regression model, or more generally, to measure the difference between predicted values and actual values.

Linear Regression

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

r2_train = r2_score(y_train, y_pred_train)
r2_test = r2_score(y_test, y_pred_test)
rmse_train = np.sqrt(mean_squared_error(y_train, y_pred_train))
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
print(f'R2 train: {r2_train}')
print(f'R2 test: {r2_test}')
print(f'RMSE train: {rmse_train}')
print(f'RMSE test: {rmse_test}')
```

```
R2 train: 0.941194962056212 6
R2 test: 0.98816951572912 6
RMSE train: 6012.459573099956
RMSE test: 3580.979237321345
```

PLS

Partial Least Squares Approach

$$y_{nj} = \sum_{i=0}^{k} \beta_i x_{ni} + \varepsilon_{nj}$$

It is a linear relation but instead of the pure X variables we are using LV (Latent Variables) similar to PCR (Principal Components Regression) but LV are build to “better correlate” also Y variable respect to PC (Principal Components).

Partial Least Squares Approach

$y_{nj} =$

The hyperparameter here is represented by the number of latente variables used

s used to
s
removed (eg.

It is a lin
the pure
(Latent V
(Principa
but LV are build to "better correlate"
also Y variable respect to PC
(Principal Components).

the number
d looking for
set while
increasing the number of LV (Latente Variables)

Partial Least Squares Approach

- Both PLS and PCR perform multiple linear regression, that is they build a linear model, Y=XB+EY=XB+E
- n PCR (Principal Component Regression)) the set of measurements XX is transformed into an equivalent set X’=XWX’=XW by a linear transformation WW, such that all the new ‘features (which are the principal components) are linearly independent.
- PLS is based on finding a similar linear transformation, but accomplishes the same task by maximising the covariance between YY and X’X’

Partial Least Squares Approach

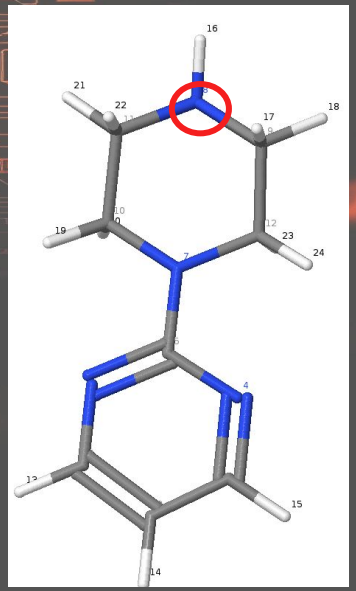
```
df = pd.read_csv('./data/fingerpls.txt', sep=' ', header=None)
print(df.head())
X = df.iloc[:, 1:-1].values
X = X.astype(float)
y = df.iloc[:, -1].values
print(X.shape)
print(y.shape)
```

```
        0     1     2     3     4     5     6     7     8     9     ...  3022  \
0   47748_2     0     0     0     0     0     0     0     0     0  ...     0
1   71274_4     0     0     0     0     0     0     0     0     0  ...     0
2   99679_2     0     0     0     0     0     0     0     0     0  ...     0
3  126628_1     0     0     0     0     0     0     0     0     0  ...     0
4  127995_1     0     0     0     0     0     0     0     0     0  ...     0

   3023  3024  3025  3026  3027  3028  3029  3030     3031
0     0     0     0     0     0     0     0     0 -2.87789
1     0     0     0     0     0     0     0     0 -4.41142
2     0     0     0     0     0     0     0     0 -0.99876
3     0     0     0     0     0     0     0     0 -3.92674
4     0     0     0     0     0     0     0     0 -2.63751

[5 rows x 3032 columns]
(207, 3030)
(207,)
```

Fingweorints

The molecular environment is described by a tree-structured molecular fingerprint with a length of 10 bond distances

0	1	8	N_3H 122			
1	2	9	C.3 326	11	C.3 326	
2	2	12	C.3 629	10	C.3 629	
3	1	7	N.3_ar 1016			
4	1	5	C.ar+ 1250			
5	2	4	NPYM 1706	6	NPYM 1706	
6	2	3	C.ar+ 1856	1	C.ar+ 1856	



Fingweorints

The
leng

If we consider 10 atom types and a fingerprint with a depth of 7

1	0	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	00	0	0	2	0	0	0	0

```
0
1
2
3
4
5
6
```

Partial Least Squares Approach

Cross validation K-folds:

- Data Splitting:
  - the dataset is randomly divided into K equal-sized parts (folds).
- Training and Testing:
  - The model is trained K separate times.
  - In each iteration, K-1 folds are used for training the model, and the remaining 1 fold is used for testing.

This way, each fold gets a chance to be the test set while the rest are used for training.

Partial Least Squares Approach

Performance Evaluation:

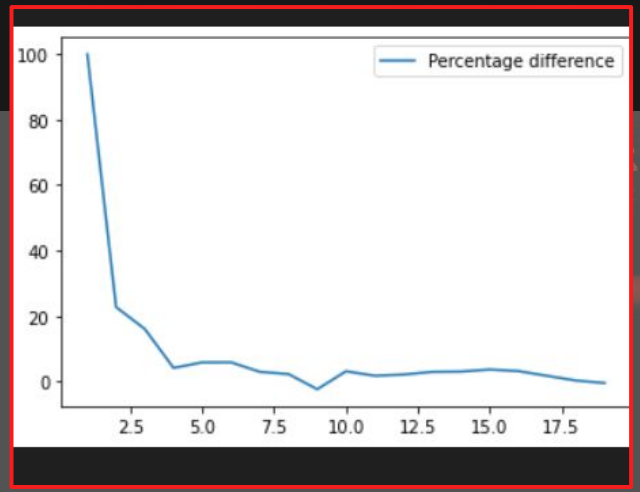
- For each iteration, a performance metric (e.g., accuracy, precision, recal, RMSE) is calculated on the held-out test fold.
- This results in K performance scores.
- The final performance score is calculated by averaging the K individual scores. This provides a more robust estimate of the model's generalization performance compared to a single train-test split.

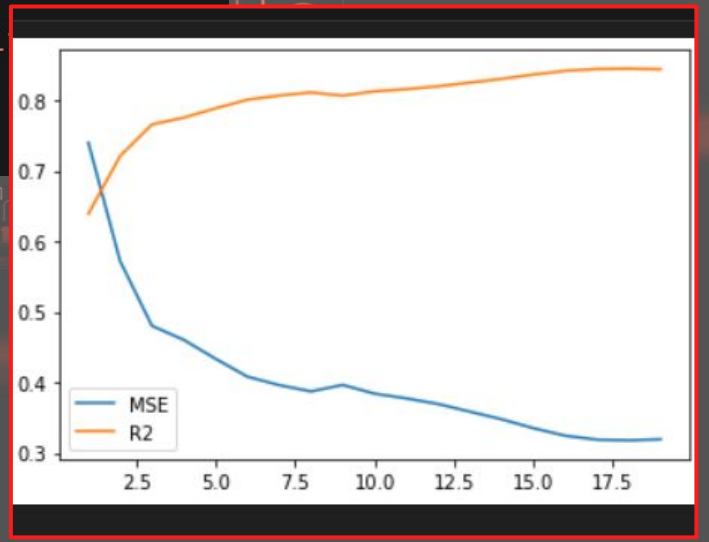
Partial Least Squares Approach

```
mses = []
r2s = []
percdiff = []
for n in range(1, 20):
    pls = PLSRegression(n_components=n)
    y_pred = cross_val_predict(pls, X, y, cv=10)
    mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)
    mses.append(mse)
    r2s.append(r2)

    if n > 1:
        percdiff.append((mses[-2] - mses[-1]) / mses[-2] * 100)
    else:
        percdiff.append(100)
```

Partial Least Squares Approach

```
plt.plot(range(1, 20), mses, label='MSE')
plt.plot(range(1, 20), r2s, label='R2')
plt.legend()
plt.show()

plt.plot(range(1, 20), percdiff, label='Percentage di
plt.legend()
plt.show()
```



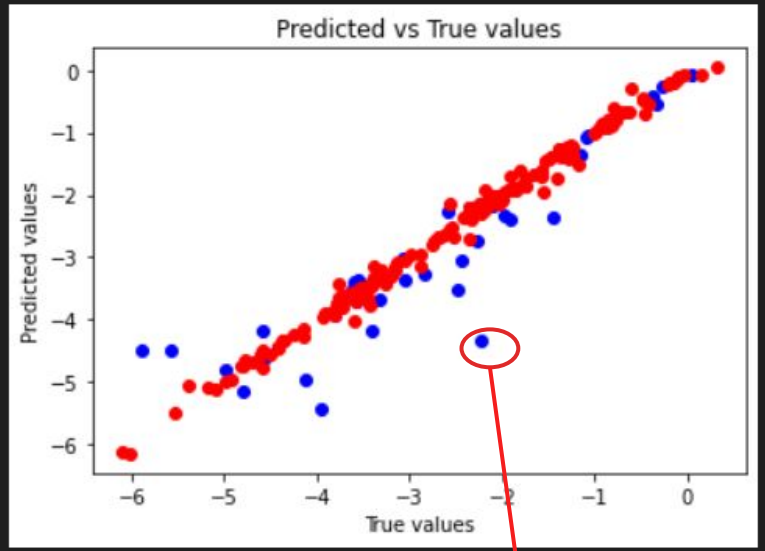


Partial Least Squares Approach

```
# splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, \
                                                    random_state = 0)
# build a PLS model using 12 components
pls = PLSRegression(n_components=11)
pls.fit(X_train, y_train)
y_pred_test = pls.predict(X_test)
msetest = mean_squared_error(y_test, y_pred_test)
r2test = r2_score(y_test, y_pred_test)

y_pred_train = pls.predict(X_train)
msetrain = mean_squared_error(y_train, y_pred_train)
r2train = r2_score(y_train, y_pred_train)
```

Partial Least Squares Approach

```
print('MSE test:', msetest)
print('R2 test:', r2test)
print('MSE train:', msetrain)
print('R2 train:', r2train)
plt.scatter(y_test, y_pred_test, color='blue')
plt.scatter(y_train, y_pred_train, color='red')
plt.title('Predicted vs True values')
plt.xlabel('True values')
plt.ylabel('Predicted values')
plt.show()
```




```
MSE test: 0.3604904172089445 4
R2 test: 0.8382020928663912
MSE train: 0.014821067700162026
R2 train: 0.992527008294187
```

Outlier

RF and classification

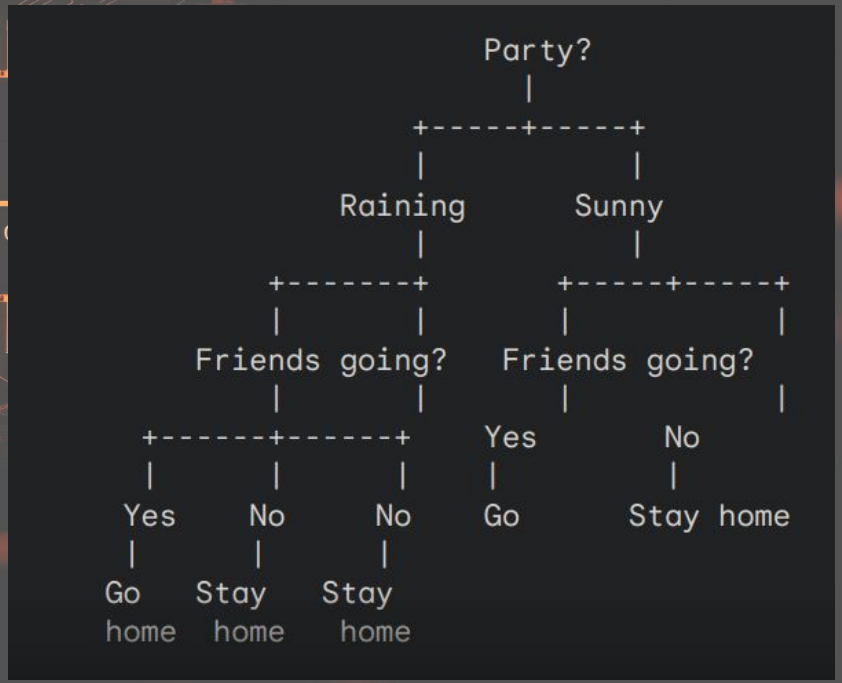
Decision Tree

Imagine you're trying to decide whether to go to a party. You might consider factors like:

1. Weather: Is it raining or sunny?
2. Friends: Are your friends going?
3. Time: Is it a weeknight or weekend?

...

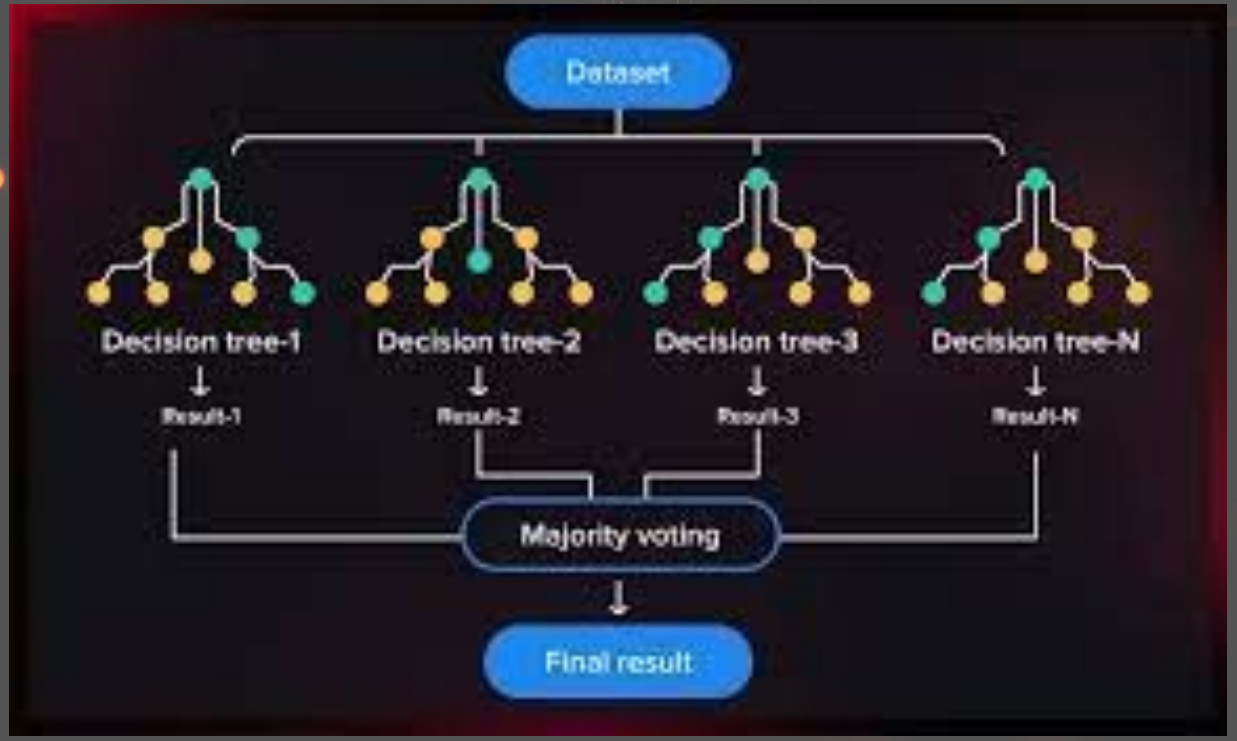
You could use a decision tree to map out your decision-making process



Random Forest Approach

- Imagine you have a complex problem to solve, and you gather a group of experts from different fields to provide their input. Each expert provides their opinion based on their expertise and experience. Then, the experts would vote to arrive at a final decision.
- In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data.
- Predictions are made by calculating the prediction for each decision tree and then taking the most popular result. (For regression, predictions use an averaging technique instead.)

Random Forest Approach



Random Forest Approach

Here we will build a classification model so let's define some metrics:

True Positive (TP): The model correctly predicted spam, and it was actually spam.

True Negative (TN): The model correctly predicted not spam, and it was actually not spam.

False Positive (FP) (Type I Error): The model predicted spam, but it was actually not spam (a false alarm).

False Negative (FN) (Type II Error): The model predicted not spam, but it was actually spam (a missed detection).

Random Forest Approach

Confusion Matrix

	Predicted Class 1	**Predicted Class 0**
Truly Class 1	True Positive (TP)	False Negative (FN)
Truly Class 0	False Positive (FP)	True Negative (TN)

Random Forest Approach

Recall = TP / (TP + FN) essentially measures the ability of a classifier to find all the positive instances in your datase
Accuracy = (TP + TN) / (TP + TN + FP + FN) is a common metric that measures the overall correctness of a model's predictions
Precision = TP / (TP + FP)evaluating the performance of a classifier, particularly when you want to minimize false positives

Random Forest Approach

```
df = pd.read_csv('./data/bbb.csv', sep=';')
df = df[df['Objects'].str.contains('_c0')]
df['Objects'] = df['Objects'].str.replace('_c0', '')
df_label = pd.read_csv('./data/bbb_label.csv', sep=' ', header=None)
# give a name to the columns
df_label.columns = ['Objects', 'label']
# select only row with the same name as in the label file
df = df[df['Objects'].isin(df_label['Objects'])]
df = df.set_index('Objects')
df_label = df_label.set_index('Objects')
# add all in a single dataframe
df = df.join(df_label)
print(df.head())
```

Random Fore

```
df = pd.read_csv('
df = df[df['Object
df['Objects'] = df
df_label = pd.read
# give a name to t
df_label.columns =
# select only row
df = df[df['Object
df = df.set_index(
df_label = df_labe
# add all in a sin
df = df.join(df_la
print(df.head())
```

Objects	V	S	R	G	W1	W2
MOL_0001	747.125	482.581	1.54819	1.33424	1098.380	579.750
MOL_0002	952.125	637.029	1.49463	1.58788	1294.120	597.500
MOL_0004	549.625	368.758	1.49047	1.21149	798.000	324.500
MOL_0009	595.750	414.276	1.43805	1.33090	898.375	406.875
MOL_0012	448.500	316.026	1.41919	1.17786	692.250	323.000

Objects	W4	W5	W6	...	L4LgS	DD1	DD2
MOL_0001	106.375	48.875	13.875	...	-0.067867	61.750	32.500
MOL_0002	104.625	54.875	29.750	...	-0.137969	92.875	37.500
MOL_0004	35.875	15.500	6.625	...	0.012077	0.000	0.000
MOL_0009	98.000	51.625	25.125	...	-0.315401	30.500	20.125
MOL_0012	63.000	29.375	12.125	...	0.158180	0.000	0.000

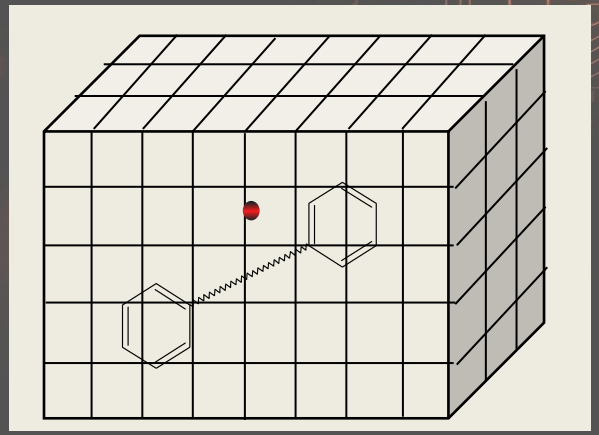
Objects	DD4	DD5	DD6	DD7	DD8	label
MOL_0001	11.250	7.875	5.625	4.750	3.500	1
MOL_0002	11.625	10.250	8.750	5.750	5.375	0
MOL_0004	0.000	0.000	0.000	0.000	0.000	1
MOL_0009	5.000	4.625	3.125	1.625	0.750	1

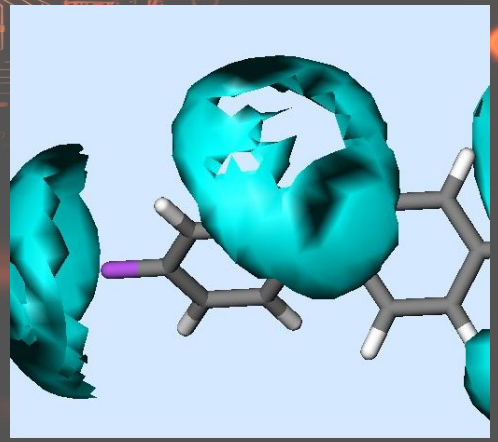
GRID Force-Fields

- **GRID program**: a computational procedure for determining energetically favourable binding sites on molecules for functional groups of known structure through the use of PROBES.
  - The PROBE is moved through a grid of points superimposed on the target molecule (to each atoms of the target and AtomType is assigned) . Its interaction energy with the target molecule is computed by an empirical energy function

$$E_{XYZ} = \Sigma[E_{LJ}] + \Sigma[E_{HB}] + \Sigma[E_Q] + [S] + \dots$$

E_{LJ} = Lennard-Jones potential E_{HB} = hydrogen bonding interaction energy E_Q = electrostatic function S= entropic term





Volsurf Descriptors

Descriptors	Probes*			Description
	OH2	DRY	O	
V	X			Molecular volume
S	X			Molecular surface
POL				Polarizability
MW				Molar mass
HB1–HB8			X	Hydrogen bonding
A				Amphiphilic moment
BV	X		X	Best volumes
W1–W8	X			Hydrophilic regions
ID1–ID8		X		Hydrophobic integy moment
Cw1–Cw8	X			Capacity factor
D1–D8		X		Hydrophobic regions
CP				Critical packing
LOG P				logarithm of partition coefficient
DIFF				Diffusivity

* Blank, other ways of calculation. For deatails, see reference Cruciani et al. (2000).

Random Forest Approach

```
(2103, 128)
(2103, 68)
```

```python
X = df.drop('label', axis=1)
X = X.fillna(0)
y = df['label']
print(X.shape)
CORRCUT = 0.95
corr_matrix = X.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), \
                                  k=1).astype(bool))
to_drop = [column for column in upper.columns if any(upper[column] > CORRCUT)]
X = X.drop(X[to_drop], axis=1)
print(X.shape)
```

Random Forest Approach

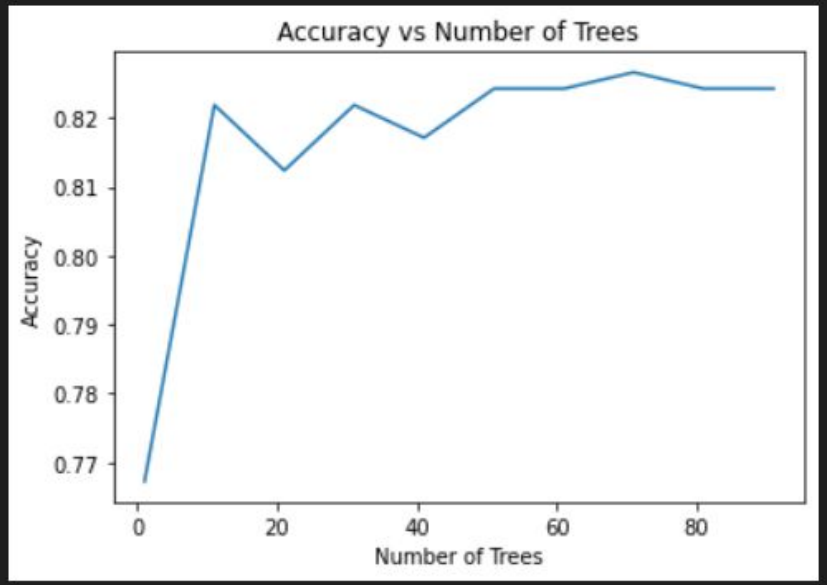
```
X_train, X_test, y_train, y_test = train_test_split(X, y, \
                                   test_size=0.2, random_state=42)
accuracys = []
numoftrees = []
for numofest in range(1, 100, 10):
    rf = RandomForestClassifier(n_estimators=numofest, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracys.append(acc)
    numoftrees.append(numofest)
    #print(f"Accuracy for {numofest} trees:", accuracy)
```

Random Forest Approach

```
import matplotlib.pyplot as plt
plt.plot(numoftrees, accuracys)
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Number of Trees')
plt.show()
print("Max Accuracy:", max(accuracys))
bestnoftrees = numoftrees[accuracys.index(max(accuracys))]
print("Number of Trees:", bestnoftrees)
```

Random Forest Approach

```
import matplotlib.pyp
plt.plot(numoftrees,
plt.xlabel('Number o
plt.ylabel('Accuracy
plt.title('Accuracy
plt.show()
print("Max Accuracy:
bestnoftrees = numof
print("Number of Tre
```




```
Max Accuracy: 0.8266033254156769
Number of Trees: 71
```

```
rf = RandomForestClassifier(n_estimators=bestnoftrees, \
                            random_state=42)
rf.fit(X_train, y_train)
# print out the max detpth used in the trees
print("Max Depth:", rf.max_depth)
y_pred = rf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
# the same for the training set
y_pred_train = rf.predict(X_train)
print("Accuracy on training set:", \
      accuracy_score(y_train, y_pred_train))
print("Confusion Matrix on training set:")
print(confusion_matrix(y_train, y_pred_train))
print("Precision on training set:", \
      precision_score(y_train, y_pred_train))
print("Recall on training set:", \
      recall_score(y_train, y_pred_train))
```

```
rf = RandomForestClassifier(n_estimators=bestnoftrees, \
                            random_state=42)

rf.fit(X_tr
# print out
print("Max
y_pred = rf
print("Accu
print("Conf
print(confu
print("Prec
print("Reca
# the same
y_pred_trai
print("Accu
      accur
print("Conf
print(confu
print("Prec
      preci
print("Reca
      recall_score(y_train, y_pred_train))
```

```
Max Depth: None
Accuracy: 0.826603254156769
Confusion Matrix:
[[ 43  58]
 [ 15 305]]
Precision: 0.8402203856749312
Recall: 0.953125
Accuracy on training set: 0.999405469678953 6
Confusion Matrix on training set:
[[ 357    1]
 [   0 1324]]
Precision on training set: 0.999245283018868
Recall on training set: 1.0
```

Tree growth: Each tree in the forest will be allowed to grow until all leaves are pure (all data points in a leaf belong to the same class) possible overfitting

Random Forest Approach

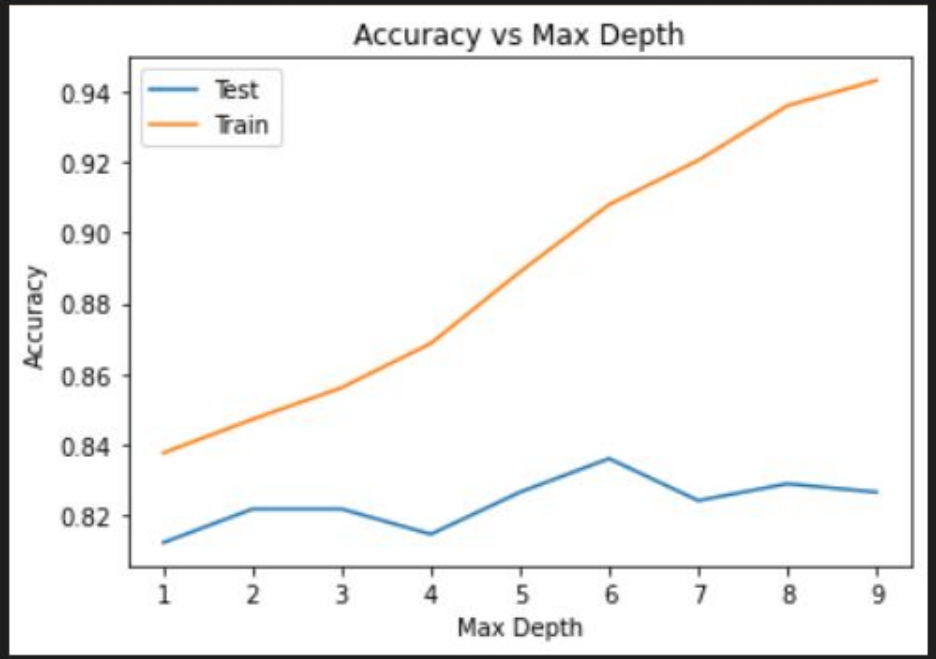
```
testaccuracy = []
trainaccuracy = []
for maxdepth in range(1, 10):
    rf = RandomForestClassifier(n_estimators=bestnoftrees, \
                                max_depth=maxdepth, \
                                random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    testaccuracy.append(accuracy_score(y_test, y_pred))
    y_pred_train = rf.predict(X_train)
    trainaccuracy.append(accuracy_score(y_train, y_pred_train))
```

Random Forest Approach

```
plt.plot(range(1, 10), testaccuracy, label='Test')
plt.plot(range(1, 10), trainaccuracy, label='Train')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Max Depth')
plt.legend()
plt.show()
bestdepth = testaccuracy.index(max(testaccuracy))+1
print ("Max Accuracy:", max(testaccuracy),\
       "Max Depth:", bestdepth)
```

Random

```
plt.plot
plt.plot
plt.xlab
plt.ylab
plt.titl
plt.lege
plt.show
bestdept
print ('Max Accuracy: 0.836104513064133 Max Depth: 6
```



```
rf = RandomForestClassifier(n_estimators=bestnoftrees, \
                            max_depth=bestdepth, \
                            random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
# the same for the training set
y_pred_train = rf.predict(X_train)
print("Accuracy on training set:", \
      accuracy_score(y_train, y_pred_train))
print("Confusion Matrix on training set:")
print(confusion_matrix(y_train, y_pred_train))
print("Precision on training set:", \
      precision_score(y_train, y_pred_train))
print("Recall on trai(variable) y_train: Any
      recall_score(y_train, y_pred_train))
```

```
rf = RandomForestClassifier(n_estimators=bestnoftrees, \

rf.fit(X_t
y_pred = r
print("Acc
print("Cor
print(conf
print("Pre
print("Rec
# the same
y_pred_tra
print("Acc
      accu
print("Cor
print(conf
print("Pre
      prec
print("Rec
        re
```

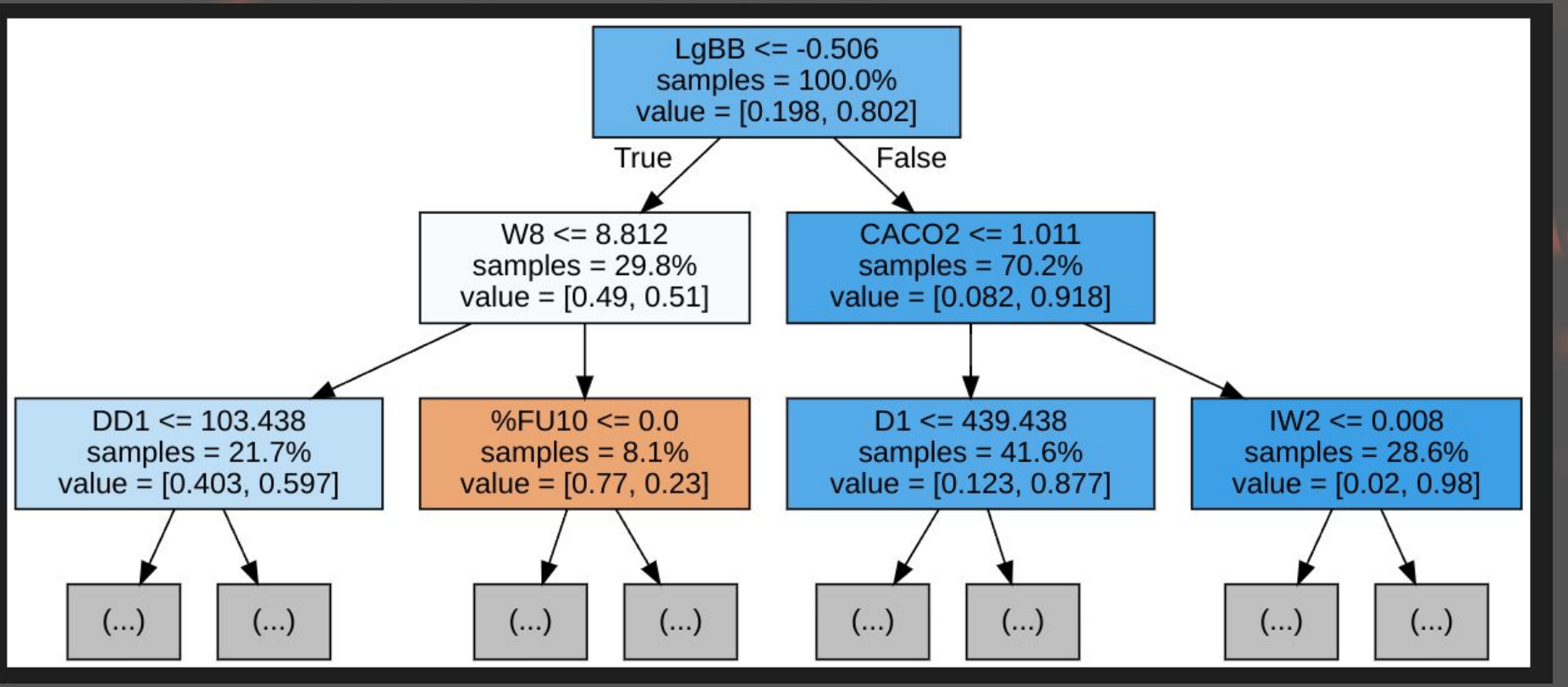
```
Accuracy: 0.836104513064133
Confusion Matrix:
[[ 37  64]
 [  5 315]]
Precision: 0.8311345646437994
Recall: 0.984375
Accuracy on training set: 0.9078478002378121
Confusion Matrix on training set:
[[ 204  154]
 [   1 1323]]
Precision on training set: 0.895734597156398 1
Recall on training set: 0.9992447129909365
```

Generate + Code

Random Forest Approach

```
for i in range(3):
    tree = rf.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_train.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    display(graph)
```

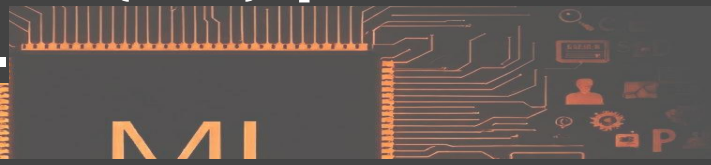
Random Forest Approach



GPR

Gaussian Process Regression

Gaussian Process Regression (GPR), predictions are based on the similarity between points.



Kernel Function: The core of GPR is the kernel function, which defines the similarity or covariance between data points. This function determines how much information is shared between points – points that are more similar according to the kernel will have more influence on each other's predictions.

Gaussian Process Regression

Prediction Process: When making a prediction for a new point, GPR considers the similarity between that new point and all the points in the training data. Points that are more similar to the new point (according to the kernel) will have a greater weight in determining the prediction.

•••

Imagine you're trying to predict the temperature at a new location. You have temperature readings from several nearby weather stations. In GPR, the kernel function would be like a measure of how close the new location is to each weather stations

Gaussian Process Regression

```
import pandas as pd
import numpy as np

df = pd.read_csv('data/surface.csv')
df = df.drop('dE', axis=1)
print(df.shape)
print(df.head())
```

0.3s

Gaussian Process Regression

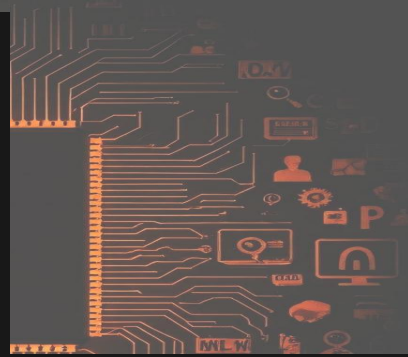
```
import pandas as pd
import nump

df = pd.rea
df = df.dro
print(df.sh
print(df.he
```

0.3s

```
(40, 71)
   v\T           100           200           300           4
0    1  4.670000e-17  1.850000e-16  5.200000e-16  1.220000e-
1    2  1.050000e-16  4.160000e-16  1.170000e-15  2.720000e-
2    3  1.790000e-16  7.010000e-16  1.960000e-15  4.560000e-
3    4  2.710000e-16  1.050000e-15  2.930000e-15  6.790000e-
4    5  3.850000e-16  1.470000e-15  4.110000e-15  9.490000e-

            600           700           800           900  .
0  5.420000e-15  1.030000e-14  1.780000e-14  2.860000e-14  .
1  1.200000e-14  2.270000e-14  3.930000e-14  6.260000e-14  .
2  2.000000e-14  3.760000e-14  6.480000e-14  1.030000e-13  .
```

Gaussian Process Regression

```
v = df['v\T']
T = df.columns[1:]
T = [int(t) for t in T]
v = [int(v) for v in v]
print(v)
print(T)
```



```
x, y = np.meshgrid(T, v)
print(x.shape)
print(y.shape)
z = np.array(df.iloc[:,1:])
print(z.shape)
```

Gaussian Process Regression

```
v = df['v\T']
T = df.columns[1:]
T = [int(t) for t in T]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 13(
(40, 70)
(40, 70)
(40, 70)
```

```
print(y.shape)
z = np.array(df.iloc[:,1:])
print(z.shape)
```

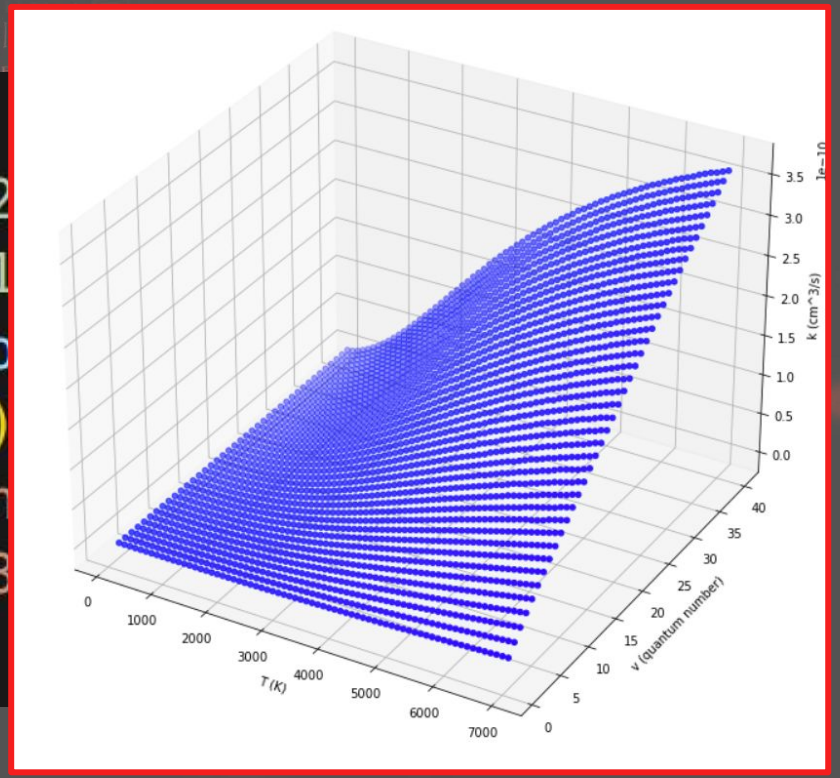
Gaussian Process Regression

```
fig = plt.figure()
fig.set_size_inches(12, 12)
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, color='b')
ax.set_xlabel('T (K)')
ax.set_ylabel('v (quantum number)')
ax.set_zlabel('k (cm^3/s)')
plt.show()
```

Gaussian Process Regression

$$N_2(v) + H_2(0) \longrightarrow N_2(v - \Delta v) + H_2(0), \quad \Delta v = 1, 2, 3.$$

```
)
fig.set_size_inches(12
ax = fig.add_subplot(1
ax.scatter(x, y, z, co
ax.set_xlabel('T (K)')
ax.set_ylabel('v (quan
ax.set_zlabel('k (cm^3
plt.show()
```



Gaussian Process Regression

```
t = x.reshape(-1)
v = y.reshape(-1)
X = np.column_stack((t, v))
Y = z.reshape(-1)
print(X.shape, X[0])
print(Y.shape, Y[0])
X_train, X_test, Y_train, Y_test = train_test_split(\
    X, Y, test_size=0.1, random_state=42)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

0.2s

```
(2800, 2) [100   1]
(2800,) 4.67e-17
(2520, 2) (280, 2) (2520,) (280,)
```

Gaussian Process Regression

```
kernel = 1.0 * Matern(length_scale=1.0, nu=2.5)
gpr = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10,\
                               normalize_y=False, random_state=42)
gpr.fit(X_train, Y_train)
Y_pred, Y_std = gpr.predict(X_test, return_std=True)
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred))
r2 = r2_score(Y_test, Y_pred)
print("RMSE %6.2e, R2 %6.2f" % (rmse, r2))
plt.scatter(Y_test, Y_pred)
plt.xlabel('True values')
plt.ylabel('Predicted values')
plt.title('Gaussian Process Regression')
plt.show()
```

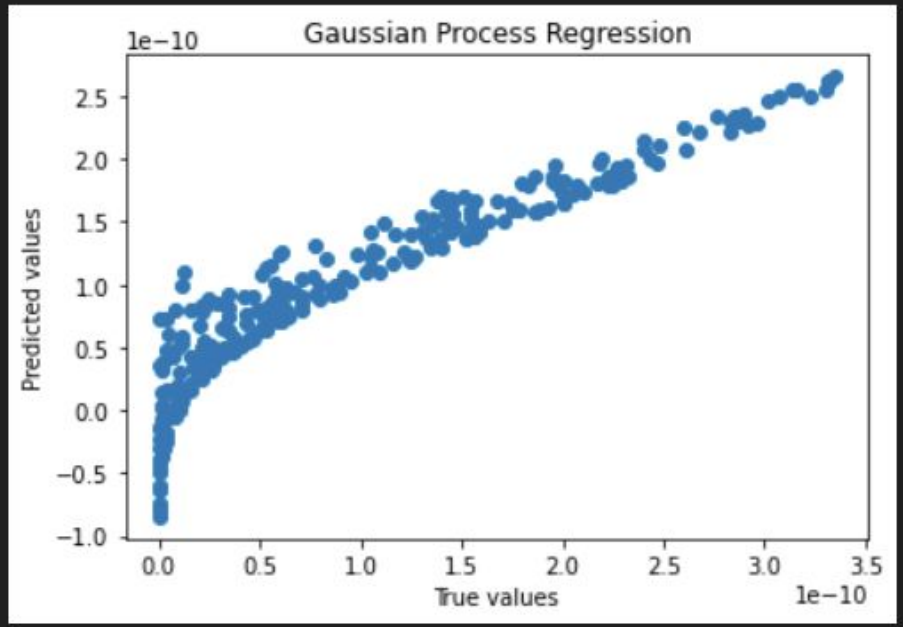
26.1s

Gaussian Process Regression

```
kernel = 1.0 *
gpr = GaussianP                                    s_optimizer=10,\
                                                   m_state=42)
gpr.fit(X_train
Y_pred, Y_std =
rmse = np.sqrt(
r2 = r2_score(Y
print("RMSE %6.
plt.scatter(Y_t
plt.xlabel('Tru
plt.ylabel('Pre
plt.title('Gaus
plt.show()
```

26.1s

```
RMSE 3.41e-11, R2   0.86
```



Gaussian Process Regression

```
print(np.min(Y_test), np.max(Y_test))
print(np.min(Y_pred), np.max(Y_pred))
```

✓ 0.0s

```
1.05e-16 3.35e-10
-8.452855806257396e-11 2.655611032996181e-10
```

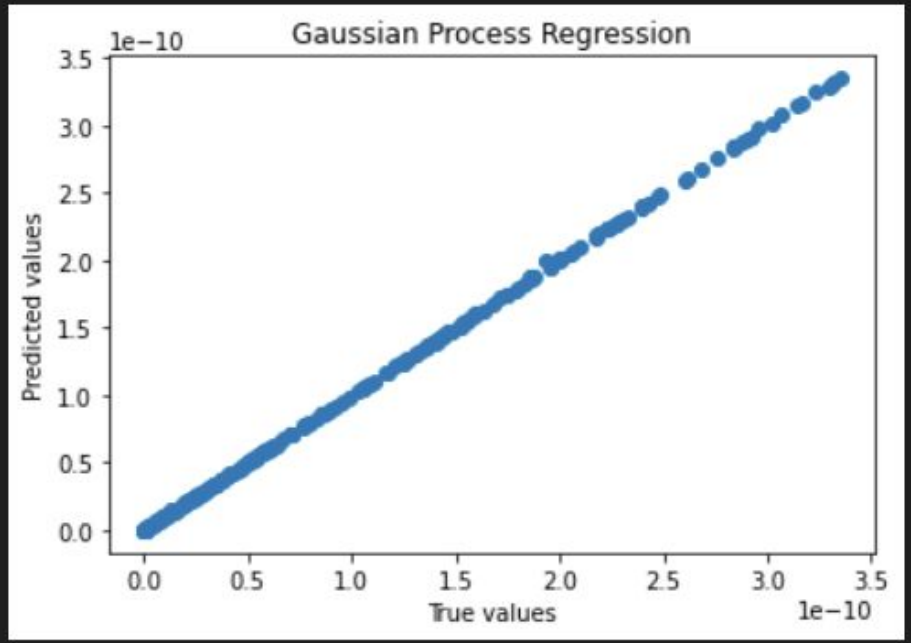
Gaussian Process Regression

```
Y_trainL = np.log10(Y_train)
kernel = 1.0 * Matern(length_scale=1.0, nu=2.5)
gpr = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10,\
                               normalize_y=False, random_state=42)
gpr.fit(X_train, Y_trainL)
Y_predL = gpr.predict(X_test)
Y_pred = 10**Y_predL
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred))
r2 = r2_score(Y_test, Y_pred)
print("RMSE %6.2e, R2 %6.2f" % (rmse, r2))
plt.scatter(Y_test, Y_pred)
plt.xlabel('True values')
plt.ylabel('Predicted values')
plt.title('Gaussian Process Regression')
plt.show()
```

Gaussian Process Regression

```
Y_trainL = n
kernel = 1.0
gpr = Gaussi

gpr.fit(X_tr
Y_predL = gp
Y_pred = 10*
rmse = np.sq
r2 = r2_scor
print("RMSE
plt.scatter(
plt.xlabel('
plt.ylabel('
plt.title('G
plt.show()
```

```
RMSE 6.02e-13, R2   1.00
```




```
imizer=10,\
te=42)
```

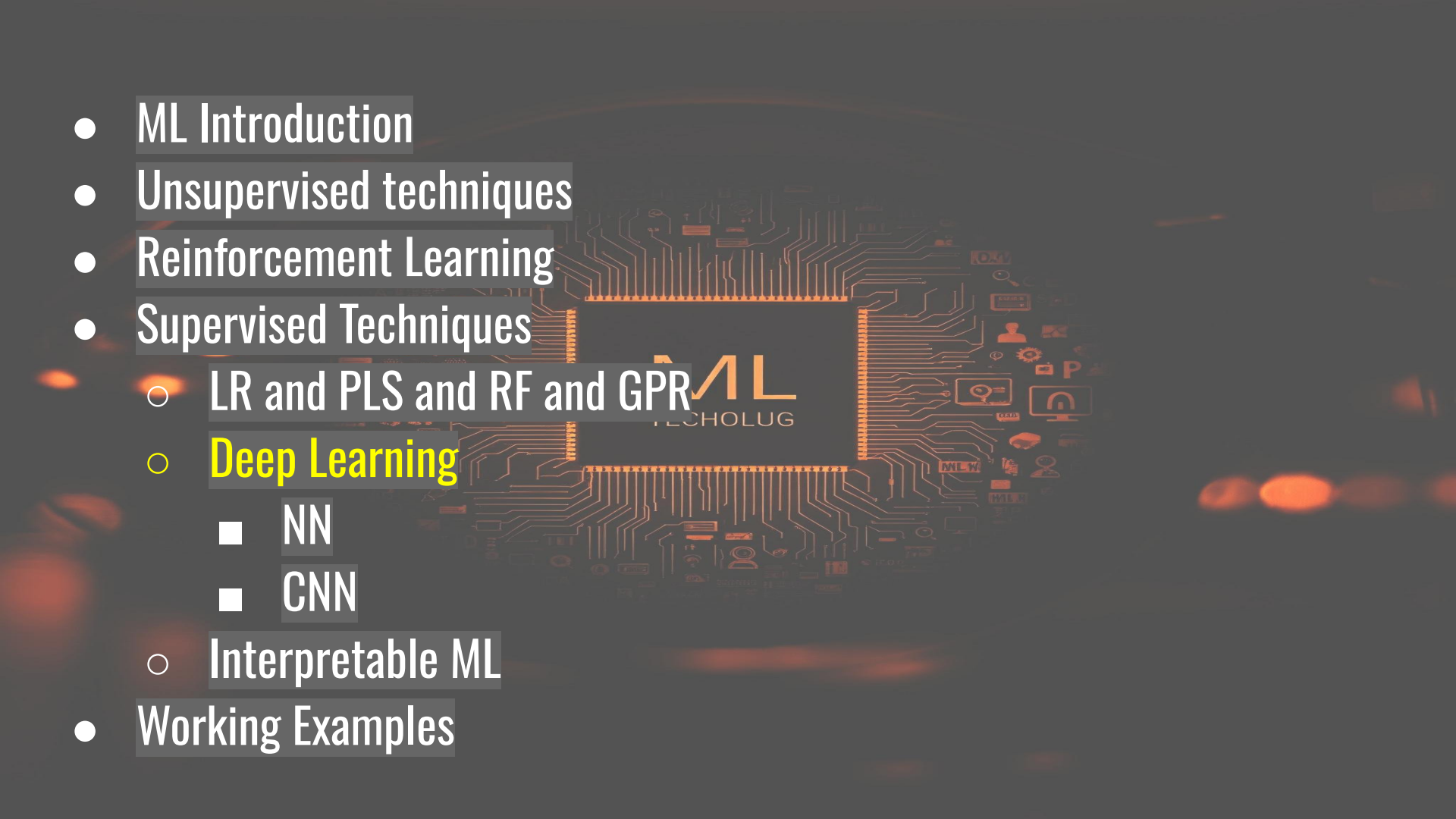
Gaussian Process Regression

```
print(np.min(Y_test), np.max(Y_test))
print(np.min(Y_pred), np.max(Y_pred))
```

✓ 0.0s

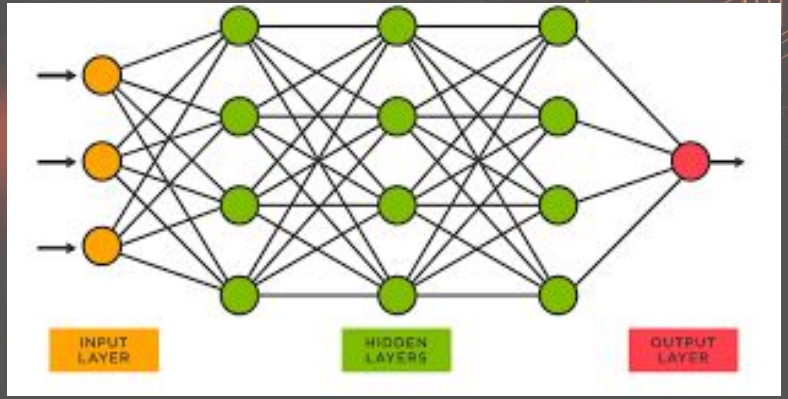
```
1.05e-16 3.35e-10
1.0112154496134055e-16 3.3487860307754 83e-10
```

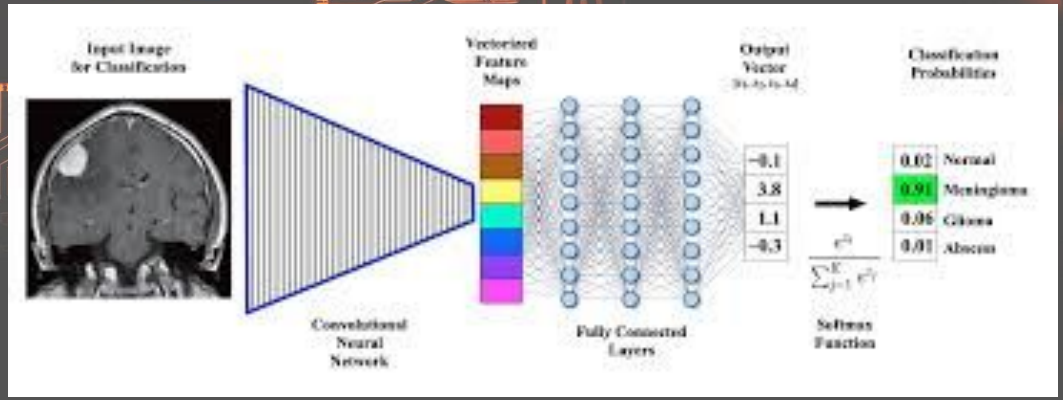
- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

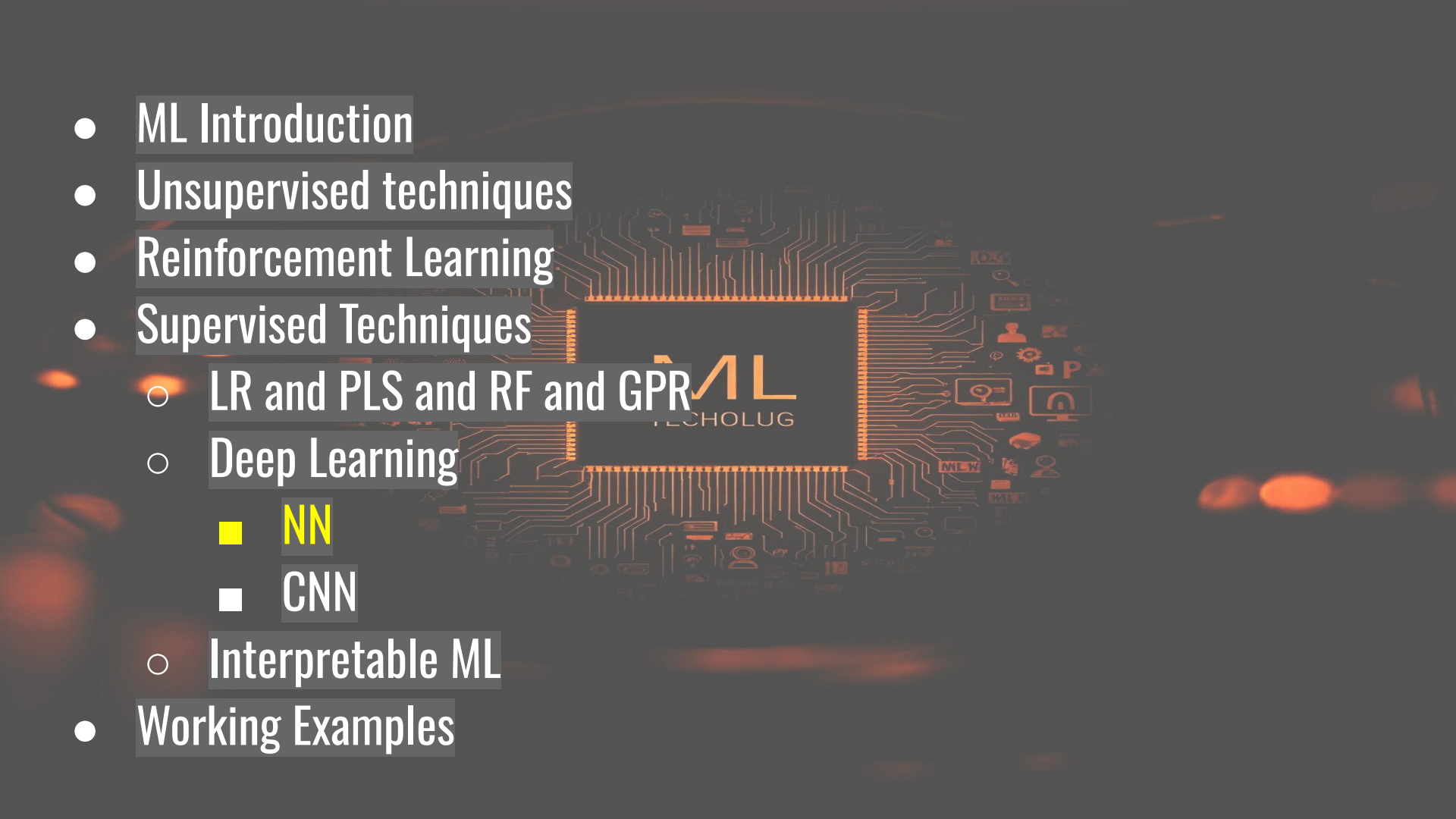


Deep Learning

Deep learning techniques involve training artificial neural networks with multiple layers (hence "deep") to learn complex patterns and representations from data



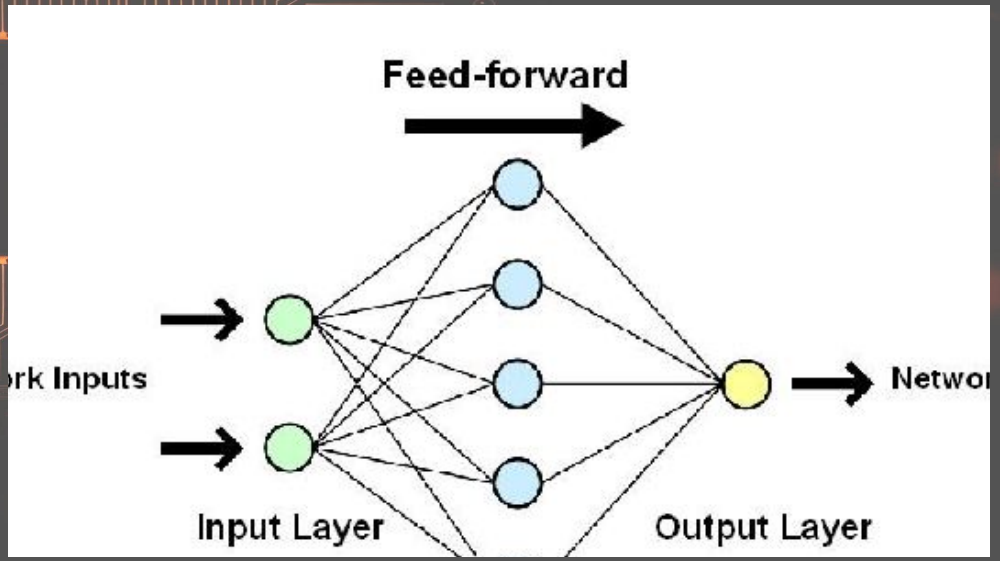




- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

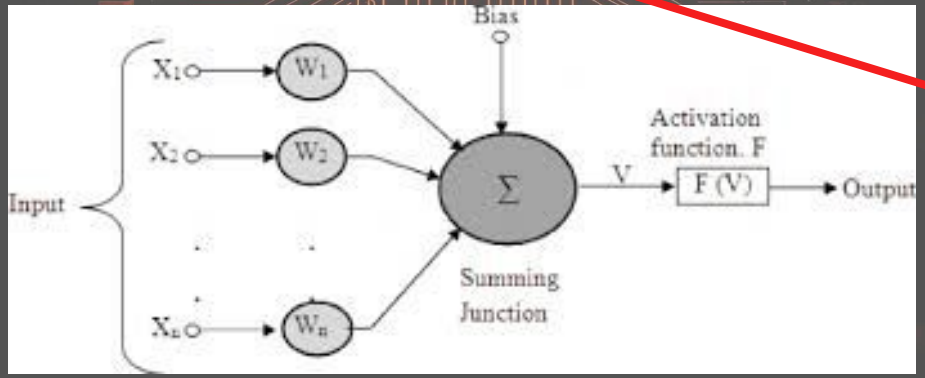
Artificial Neural Network

There are three layers in the network architecture: the input layer, the hidden layer (more than one), and the output layer. A typical feedforward network processes information in one direction, from input to output.



Artificial Neural Network

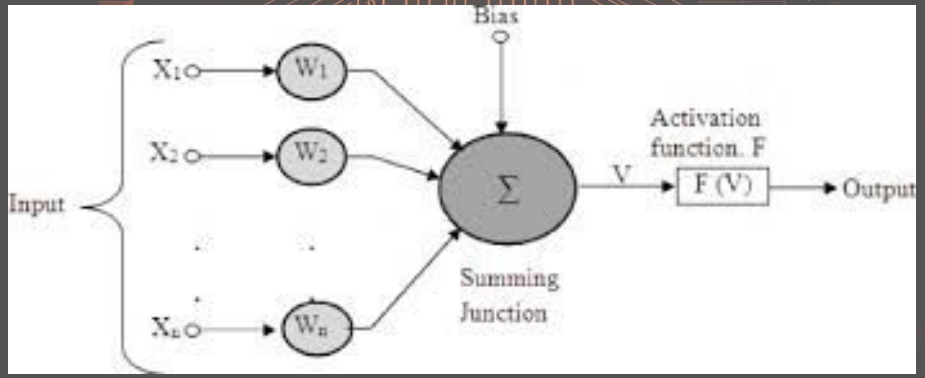
Weighted Sum: The neuron receives input signals from other neurons or from the input layer. **Each input signal is multiplied by a weight**, and these weighted inputs are summed together.



they are updated during the training

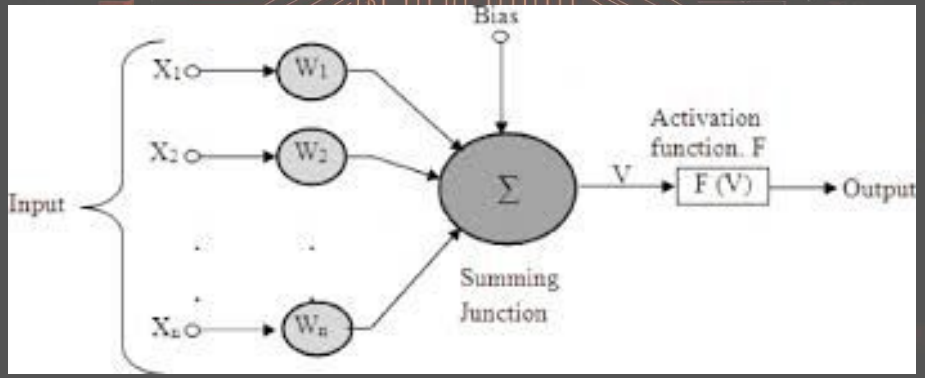
Artificial Neural Network

Bias: A bias term is added to the weighted sum. This bias **allows the neuron to shift the activation function** and learn more complex patterns.



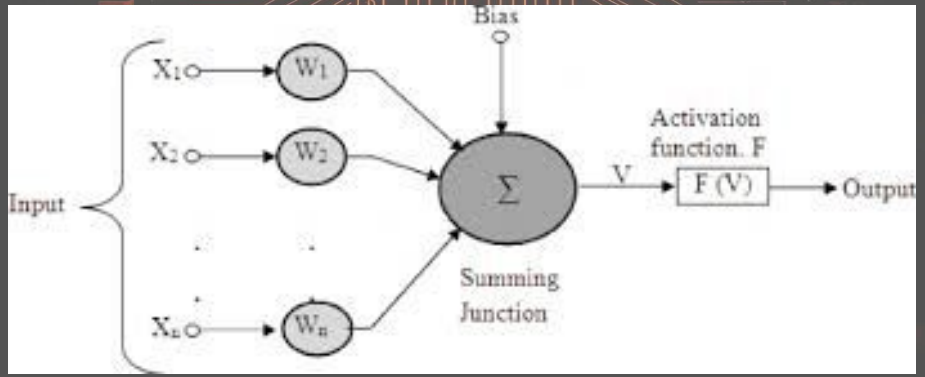
Artificial Neural Network

Activation Function: The activation function is applied to the sum of the weighted inputs and the bias. This function introduces non-linearity into the network, enabling it to learn complex relationships in the data.



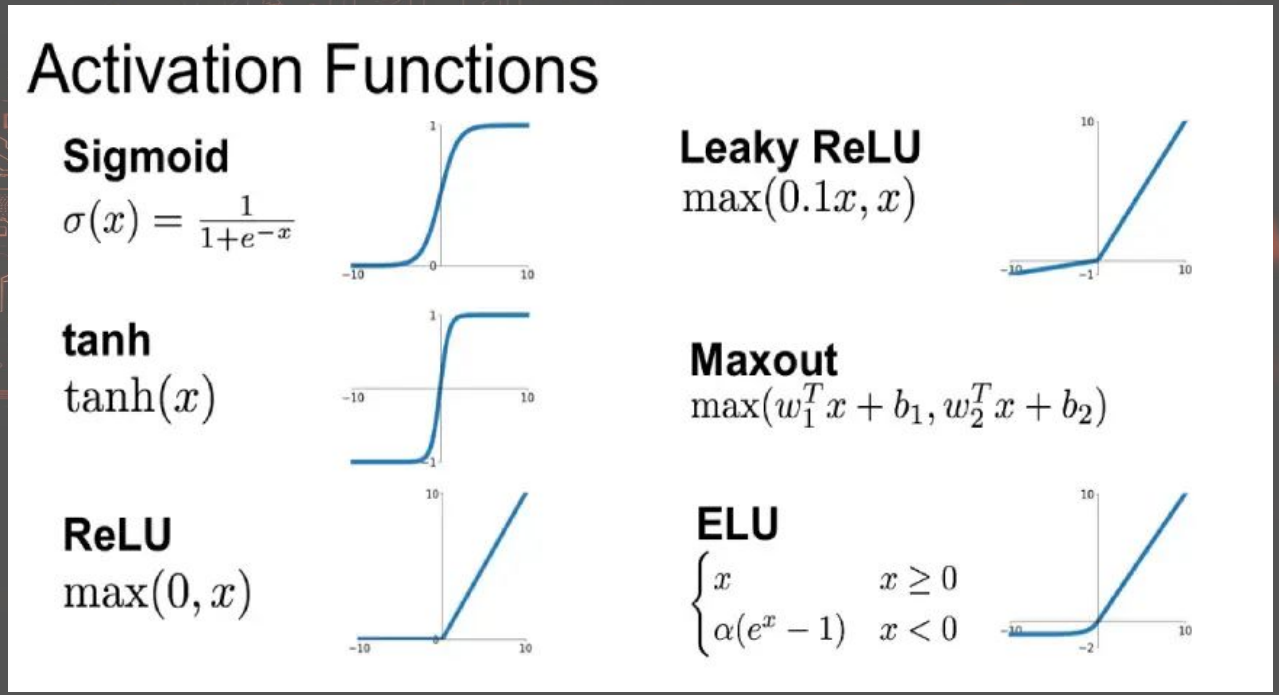
Artificial Neural Network

Output: **The output of the activation function** is the neuron's output, which is then passed on to other neurons in the next layer.



Artificial Neural Network

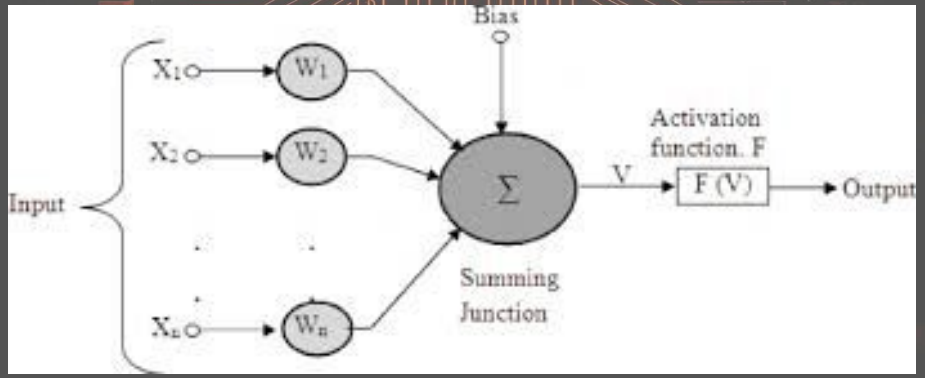
The activation function is important for two reasons: first, it allows you to turn on your computer. It contributes to the conversion of the input into a more usable final output.



Artificial Neural Network

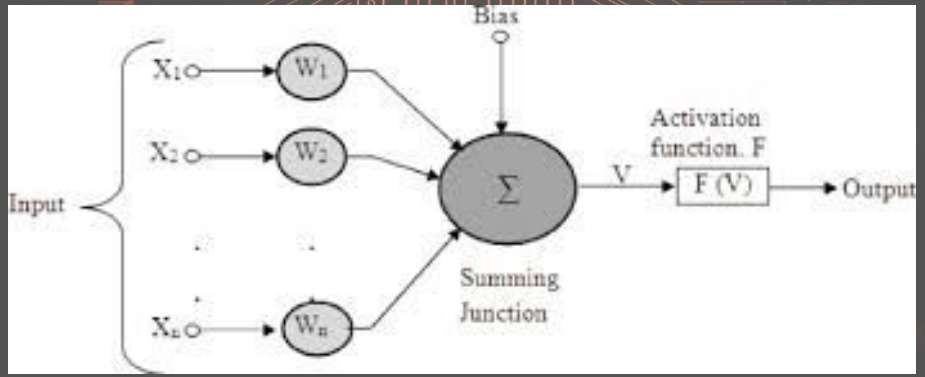
How the weights change during training

Initialization: Initially, the weights are assigned random values.



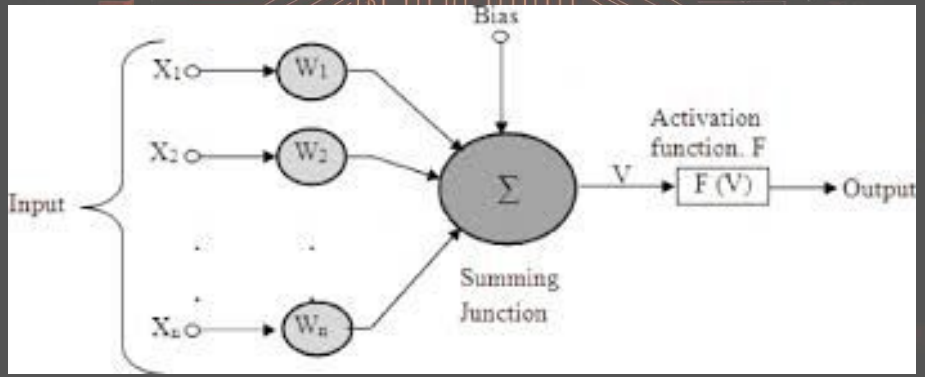
Artificial Neural Network

Forward Pass: The input data is fed through the network, and the activations of the neurons are calculated layer by layer. This process produces an output prediction.



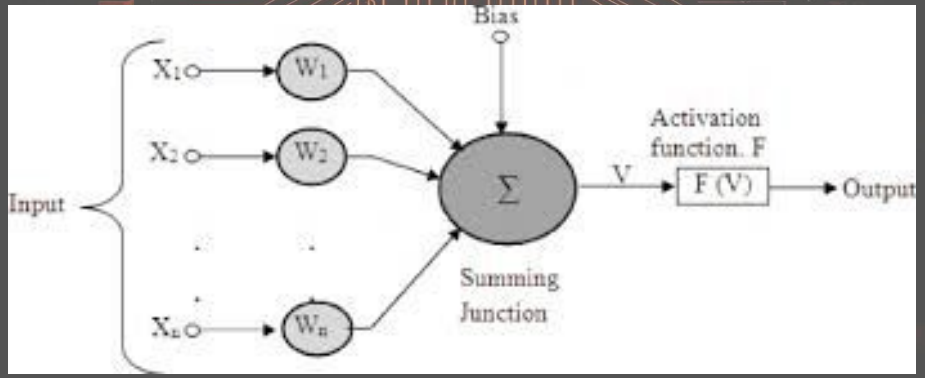
Artificial Neural Network

Loss Function: The difference between the predicted output and the actual target value is calculated using a loss function. This loss represents the error of the network.



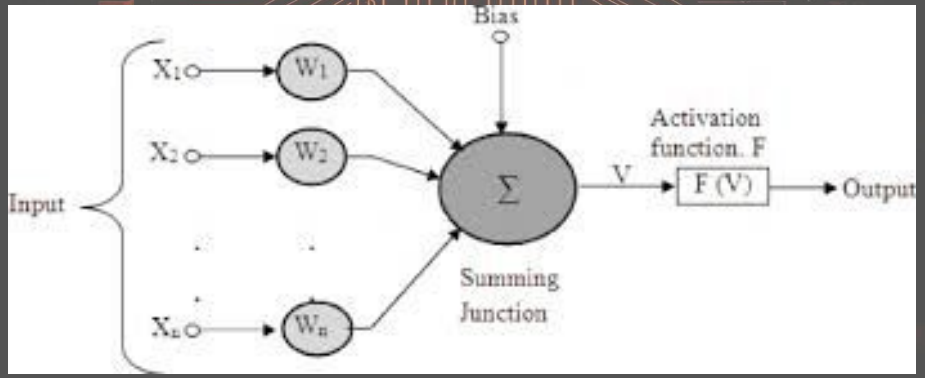
Artificial Neural Network

Backpropagation: The error is propagated back through the network, and the gradients of the loss with respect to each weight are calculated. These gradients indicate the direction and magnitude of the weight adjustments needed to reduce the error.



Artificial Neural Network

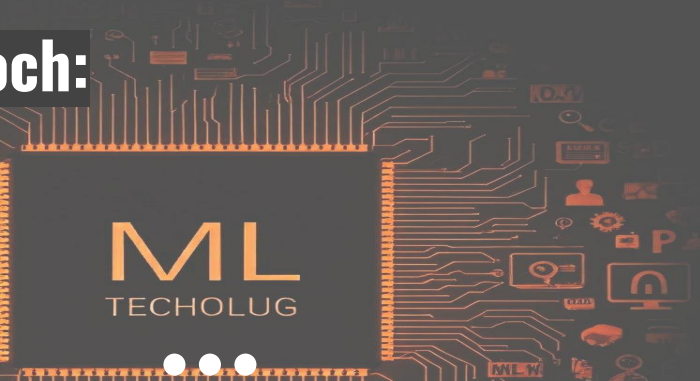
Weight Update: An optimization algorithm (like gradient descent) uses the gradients to update the weights. The weights are adjusted in the direction that minimizes the loss.



Artificial Neural Network

What happens in each epoch:

1. Data Shuffle
2. Batching (Optional)
3. Iteration
    a. Forward Pass
    b. Loss Calculation
    c. Backpropagation
    d. Weight Update
4. Epoch Completion



Artificial Neural Network

What happens in each epoch:

1. **Data Shuffle**
2. **Batching (Optional)**
3. **Iteration**
   a. **Forward Pass**
   b. **Loss Calculation**
   c. **Backpropagation**
   d. **Weight Update**
4. **Epoch Completion**

The training data is typically shuffled at the beginning of each epoch. This helps prevent the network from learning the order of the data and encourages better generalizatio

Artificial Neural Network

What happens in each epoch:

1. **Data Shuffle**
2. **Batching (Optional)**
3. **Iteration**
   a. **Forward Pass**
   b. **Loss Calculation**
   c. **Backpropagation**
   d. **Weight Update**
4. **Epoch Completion**

The training data is often divided into smaller batches. This is especially useful for large datasets that might not fit into memory all at once.

Artificial Neural Network

What happens in each epoch:

1. Data Shuffle
2. Batching (Optional)
3. Iteration
    a. Forward Pass
    b. Loss Calculation
    c. Backpropagation
    d. Weight Update
4. Epoch Completion

For each batch (or the whole dataset if not using batches)

...

Artificial Neural Network

What happens in each epoch:

1. **Data Shuffle**
2. **Batching (Optional)**
3. **Iteration**
   a. **Forward Pass**
   b. **Loss Calculation**
   c. **Backpropagation**
   d. **Weight Update**
4. **Epoch Completion**

The input data is fed through the network, and the activations of the neurons are calculated layer by layer, producing an output prediction.

Artificial Neural Network

What happens in each epoch:

1. Data Shuffle
2. Batching (Optional)
3. Iteration
   a. Forward Pass
   b. Loss Calculation
   c. Backpropagation
   d. Weight Update
4. Epoch Completion

he difference between the predicted output and the actual target value is calculated using a loss function.

Gaussian Process Regression

What happens in each epoch:

1. **Data Shuffle**
2. **Batching (Optional)**
3. **Iteration**
   a. **Forward Pass**
   b. **Loss Calculation**
   c. **Backpropagation**
   d. **Weight Update**
4. **Epoch Completion**

- the error is propagated back through the network, and the gradients of the loss with respect to the weights are calculated

Artificial Neural Network

What happens in each epoch:

1. Data Shuffle
2. Batching (Optional)
3. Iteration
   a. Forward Pass
   b. Loss Calculation
   c. Backpropagation
   d. Weight Update
4. Epoch Completion

- An optimization algorithm uses the gradients to update the weights of the network

Artificial Neural Network

What happens in each epoch:

1. **Data Shuffle**
2. **Batching (Optional)**
3. **Iteration**
   a. **Forward Pass**
   b. **Loss Calculation**
   c. **Backpropagation**
   d. **Weight Update**
4. **Epoch Completion**

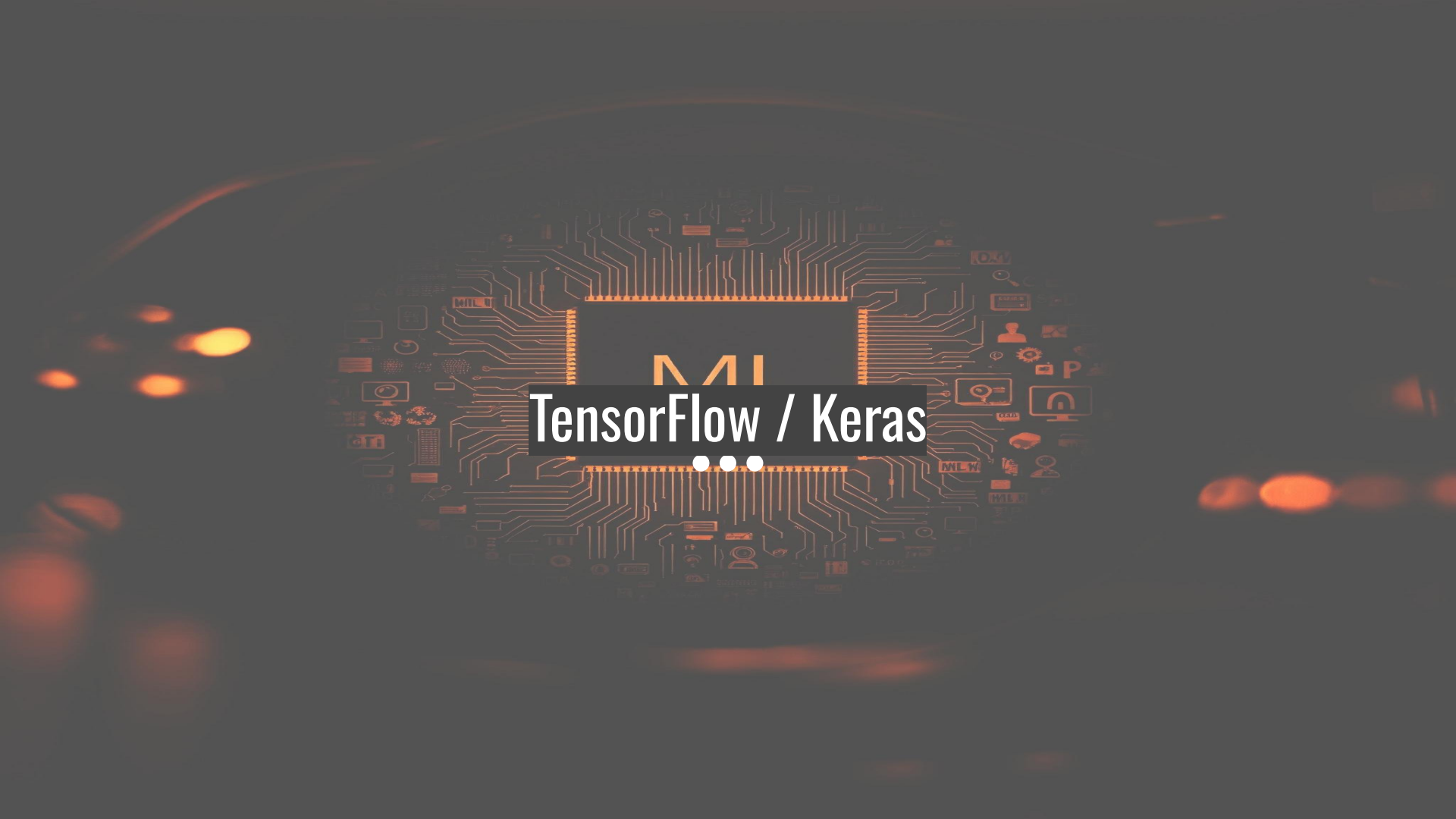
- Repeat, steps are repeated for all batches (or the whole dataset) until all the training data has been processed.

Artificial Neural Network

What happens in each epoch:

1. Data Shuffle
2. Batching (Optional)
3. Iteration
   a. Forward Pass
   b. Loss Calculation
   c. Backpropagation
   d. Weight Update
4. Epoch Completion

- Once all the training data has been processed, one epoch is complete.


TensorFlow / Keras

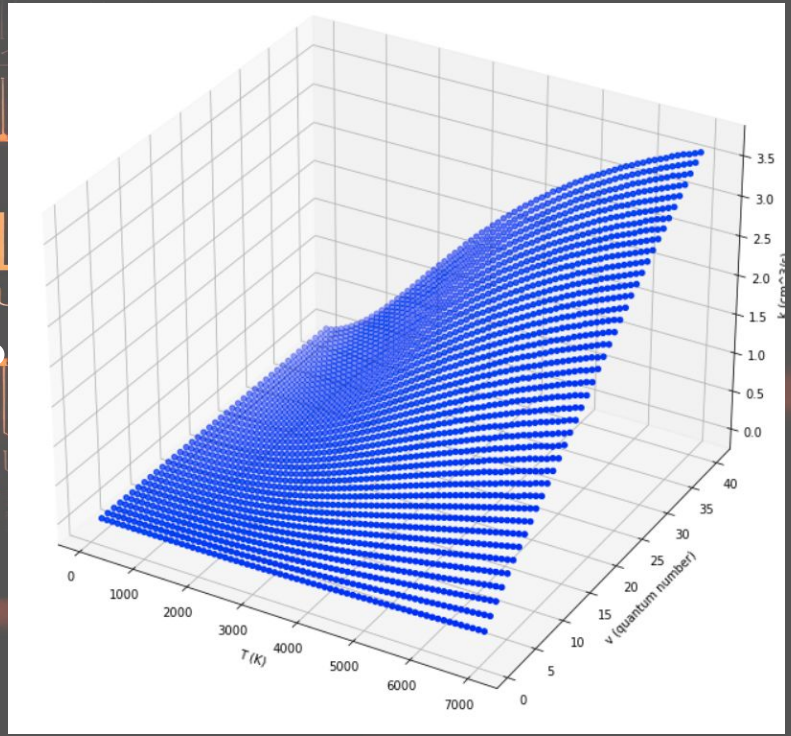
Artificial Neural Network

Tensorflow was previously the most widely used Deep Learning library, however, it was tricky to figure with for newbies. A simple one-layer network involves a substantial amount of code. With Keras, however, the entire process of creating a Neural Network's structure, as well as training and tracking it, becomes exceedingly straightforward.

Keras is a high-level API built on top of TensorFlow (and other backends like Theano and CNTK, though TensorFlow is the most common and officially supported one now)

Artificial Neural Network

```
df = pd.read_csv('data/surface.csv')
df = df.drop('dE', axis=1)

v = df['v\T']
T = df.columns[1:]
T = [int(t) for t in T]
v = [int(v) for v in v]
x, y = np.meshgrid(T, v)
z = np.array(df.iloc[:,1:])

fig = plt.figure()
fig.set_size_inches(12, 12)
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, color='b')
ax.set_xlabel('T (K)')
ax.set_ylabel('v (quantum number)')
ax.set_zlabel('k (cm^3/s)')
plt.show()
```



Artificial Neural Network

```
t = x.reshape(-1)
v = y.reshape(-1)
X = np.column_stack((t, v))
Y = z.reshape(-1)

YL = np.log10(Y)
X_train, X_test, Y_train, Y_test = train_test_split(\
    X, YL, test_size=0.1, random_state=42)
```

Artificial Neural Network

```
model = Sequential()
model.add(InputLayer(input_shape=(2,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', \
              optimizer=adam.Adam(learning_rate=0.001), \
              metrics=['mse'])
history = model.fit(X_train, Y_train, epochs=20, \
                    batch_size=64, verbose=1, validation_split=0.1)
model.save('model.h5')
```

Artificial Neural Network

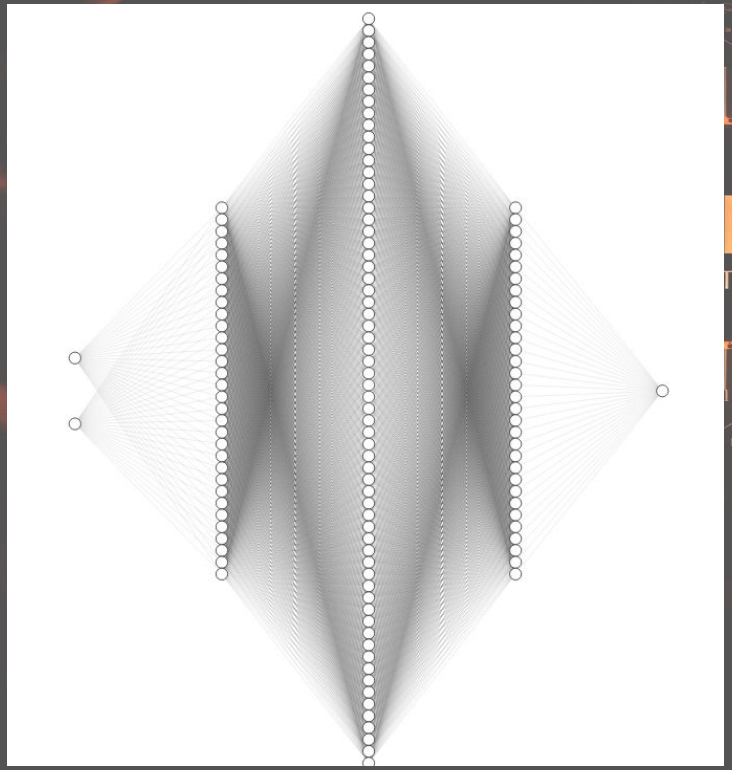
```
model = Sequential()
model.add(InputLayer(input_shape=(2,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', \
              optimizer=adam.Adam(learning_rate=0.001), \
              metrics=['mse'])
history = model.fit(X_train, Y_train, epochs=20, \
                    batch_size=64, verbose=1, validation_split=0.1)
model.save('model.h5')
```

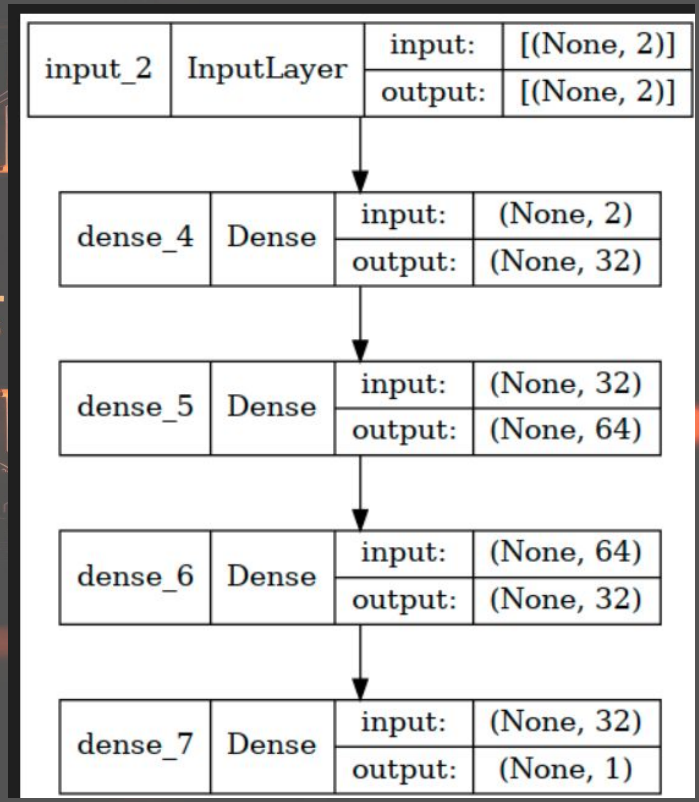
Artificial Neural Network

```
model = Sequential()
model.add(InputLayer(input_shape=(2,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', \
              optimizer=adam.Adam(learning_rate=0.001), \
              metrics=['mse'])
history = model.fit(X_train, Y_train, epochs=20, \
                    batch_size=64, verbose=1, validation_split=0.1)
model.save('model.h5')
```

Artificial Neural Network

```
# plot the layers structure using keras
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png',\
           show_shapes=True, show_layer_names=True)
```





Artificial Neural Network

```
Epoch 1/20
36/36 [==============================] - 1s 5ms/step - loss: 42596.5156 - mse: 425
Epoch 2/20
36/36 [==============================] - 0s 1ms/step - loss: 743.3082 - mse: 743.3
Epoch 3/20
36/36 [==============================] - 0s 1ms/step - loss: 54.6704 - mse: 54.670
Epoch 4/20
36/36 [==============================] - 0s 1ms/step - loss: 38.0262 - mse: 38.026
Epoch 5/20
36/36 [==============================] - 0s 2ms/step - loss: 37.2296 - mse: 37.229
Epoch 6/20
36/36 [==============================] - 0s 3ms/step - loss: 36.8328 - mse: 36.832
Epoch 7/20
36/36 [==============================] - 0s 2ms/step - loss: 36.6504 - mse: 36.650
Epoch 8/20
```

```
mod
mod
mod
mod
mod
mod
mod

his
                    batch_size=64, verbose=1, validation_split=0.1)
model.save('model.h5')
```

Artificial Neural Network

```
print("min loss: ", min(history.history['loss']))
print("min val_loss: ", min(history.history['val_loss']))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

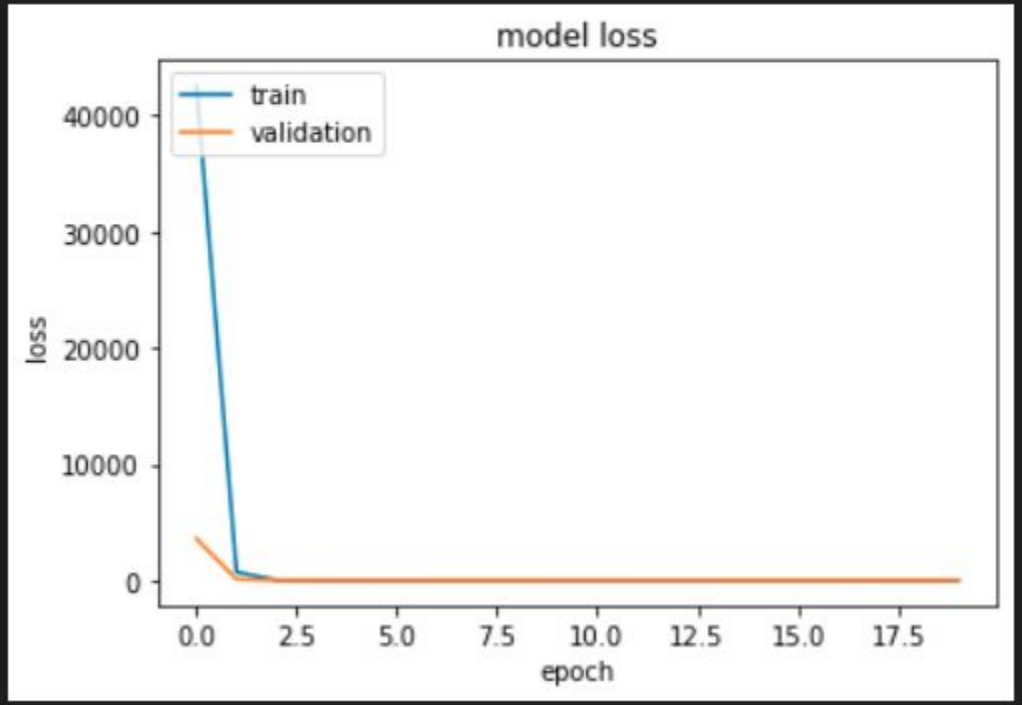
0.2s

Artificial

```
print("min
print("min
plt.plot(h
plt.plot(h
plt.title(
plt.ylabel
plt.xlabel
plt.legend
plt.show()
```

0.2s

```
min loss:  30.040092468261 72
min val_loss:  26.838100433349 61
```



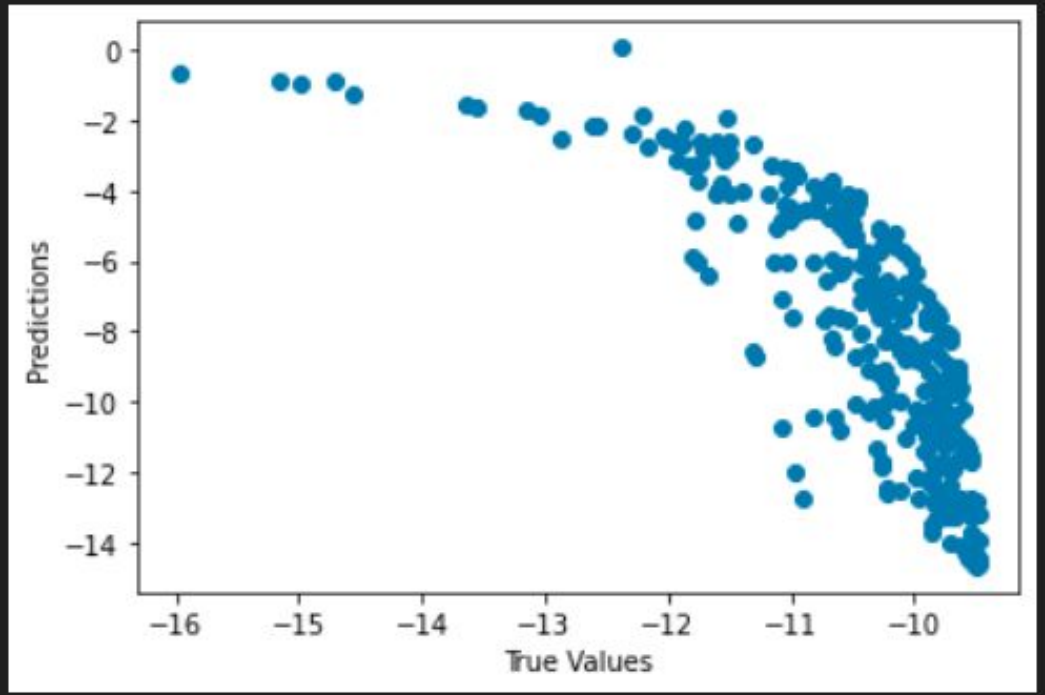
Artificial Neural Network

```
from sklearn.metrics import mean_squared_error

Y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred))
print('rmse: ', rmse)
plt.scatter(Y_test, Y_pred)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.show()
```

Artificial

```
from skle                                                    f

Y_pred =
rmse = np                                                    red))
print('rm
plt.scatt
plt.xlabe
plt.ylabe
plt.show(
```

```
rmse:  5.150392745149612
```



Artificial Neural Network

It's highly recommended and often crucial to normalize data when using neural networks, although not always strictly mandatory

```
scalerx = MinMaxScaler()
scalerx.fit(X)
Xs = scalerx.transform(X)

YL = np.log10(Y)
scalery = MinMaxScaler()
scalery.fit(YL.reshape(-1,1))
YLs = scalery.transform(YL.reshape(-1,1))

X_train, X_test, Y_train, Y_test = train_test_split(\
    Xs, YLs, test_size=0.1, random_state=42)
```

to be more correct it would be better to fite the scaler only on the training set

Artificial Neural Network

```
model = Sequential()
model.add(InputLayer(input_shape=(2,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', \
              optimizer=adam.Adam(learning_rate=0.001), \
              metrics=['mse'])
history = model.fit(X_train, Y_train, epochs=20, \
                    batch_size=64, verbose=1, validation_split=0.1)
model.save('model.h5')
```

1.5s

Artificial Neural Network

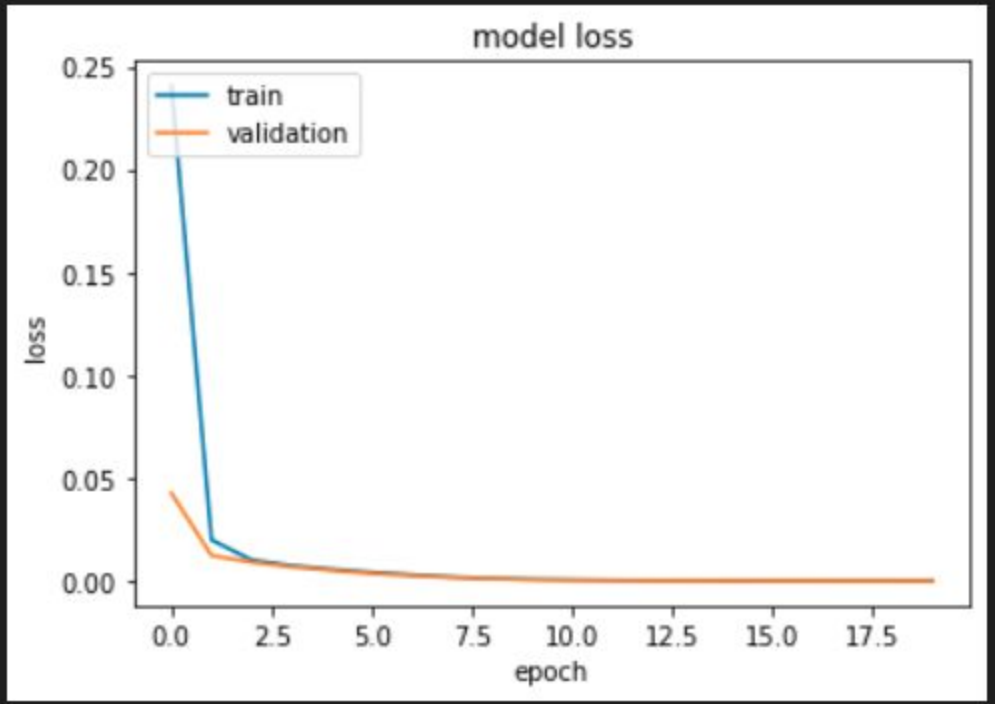
```
model = Sequential()
Epoch 1/20
36/36 [==============================] - 0s 4ms/step - loss: 0.2406 - mse: 0.2406
Epoch 2/20
36/36 [==============================] - 0s 1ms/step - loss: 0.0198 - mse: 0.0198
Epoch 3/20
36/36 [==============================] - 0s 1ms/step - loss: 0.0101 - mse: 0.0101
Epoch 4/20
36/36 [==============================] - 0s 2ms/step - loss: 0.0074 - mse: 0.0074
Epoch 5/20
36/36 [==============================] - 0s 2ms/step - loss: 0.0056 - mse: 0.0056
Epoch 6/20
                    batch_size=64, verbose=1, validation_split=0.1)
model.save('model.h5')
1.5s
```

Artificial Neural Network

```
print("min loss: ", min(history.history['loss']))
print("min val_loss: ", min(history.history['val_loss']))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Artificia

```
min loss:  5.30519027961418e-05
min val_loss:  5.0720085710054263e-05
```




```
print("
print("
plt.plo
plt.plo
plt.tit
plt.yla
plt.xla
plt.leg
plt.sho
```

```
s']))
)
```

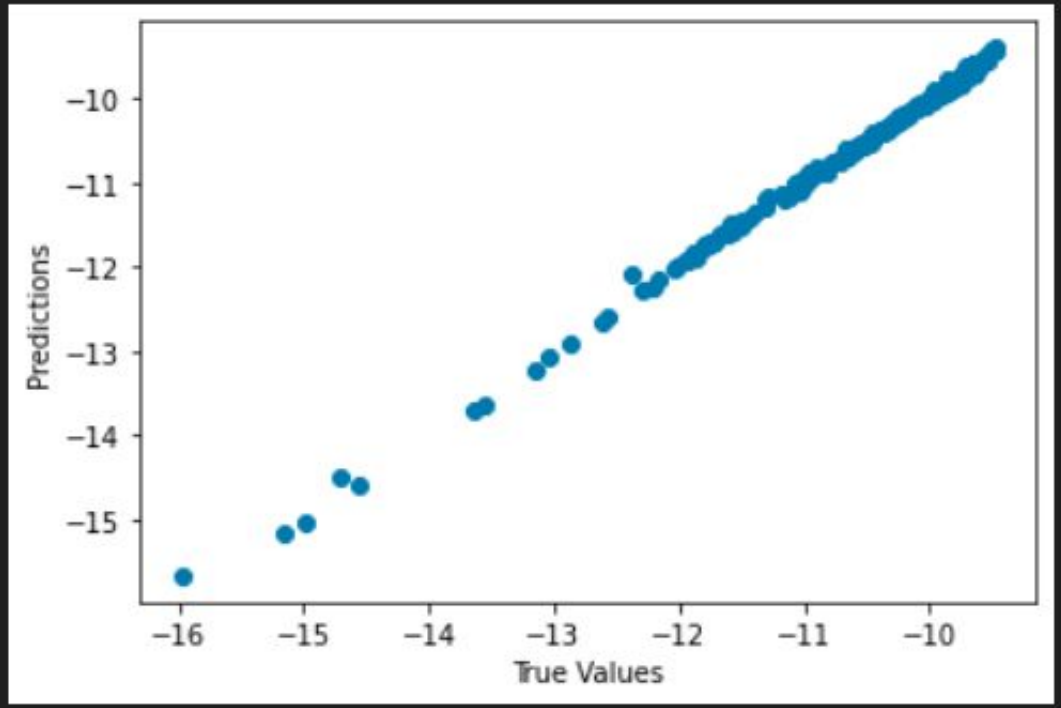
Artificial Neural Network

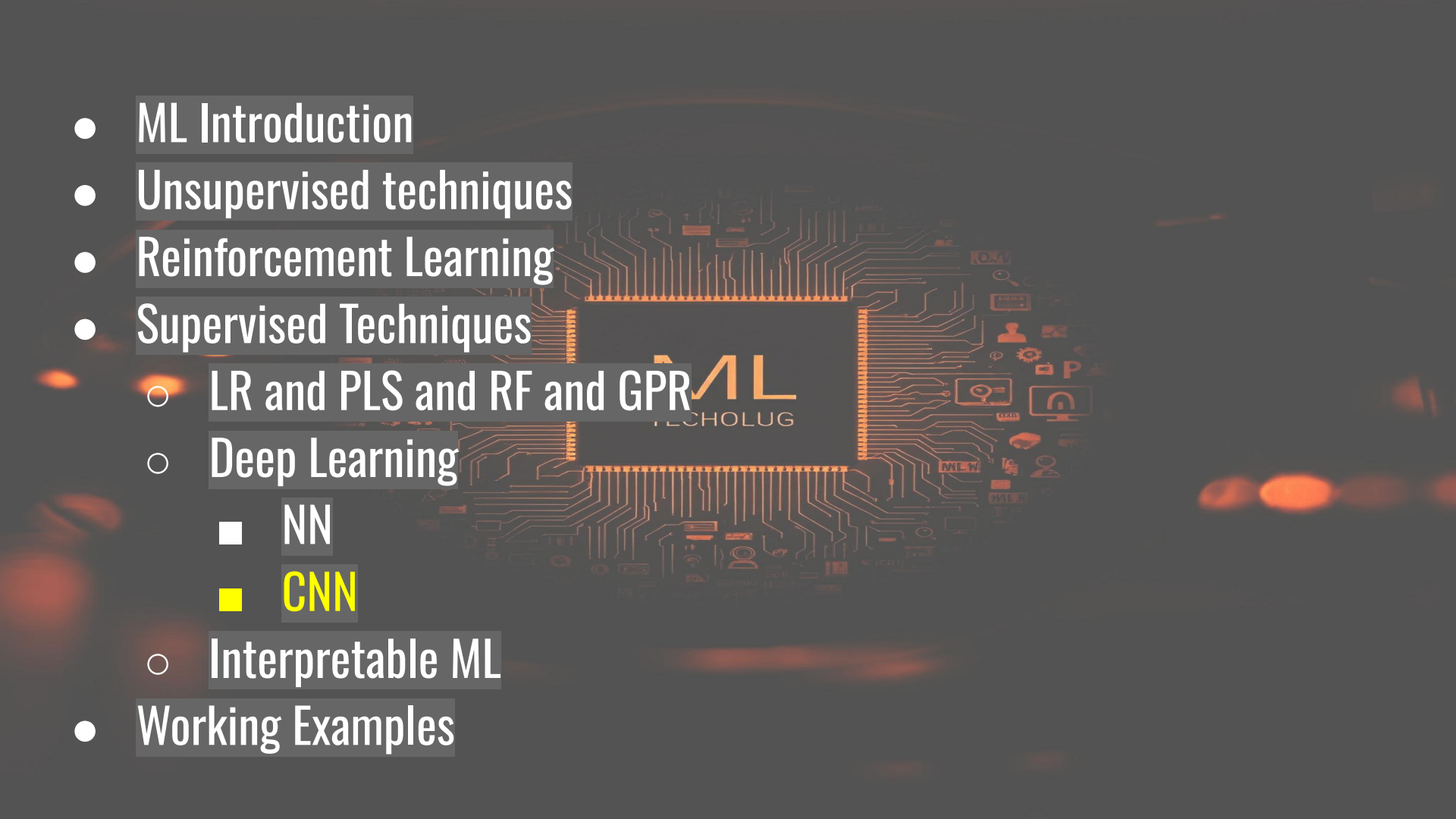
```
Y_pred = model.predict(X_test)
Y_pred = scalery.inverse_transform(Y_pred)
Y_test = scalery.inverse_transform(Y_test)
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred))
print('rmse: ', rmse)
plt.scatter(Y_test, Y_pred)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.show()
```

Artifici

```
Y_pred
Y_pred
Y_test
rmse =
print(
plt.sca
plt.xla
plt.yla
plt.sho
```

```
rmse:  0.047275215445131295
```

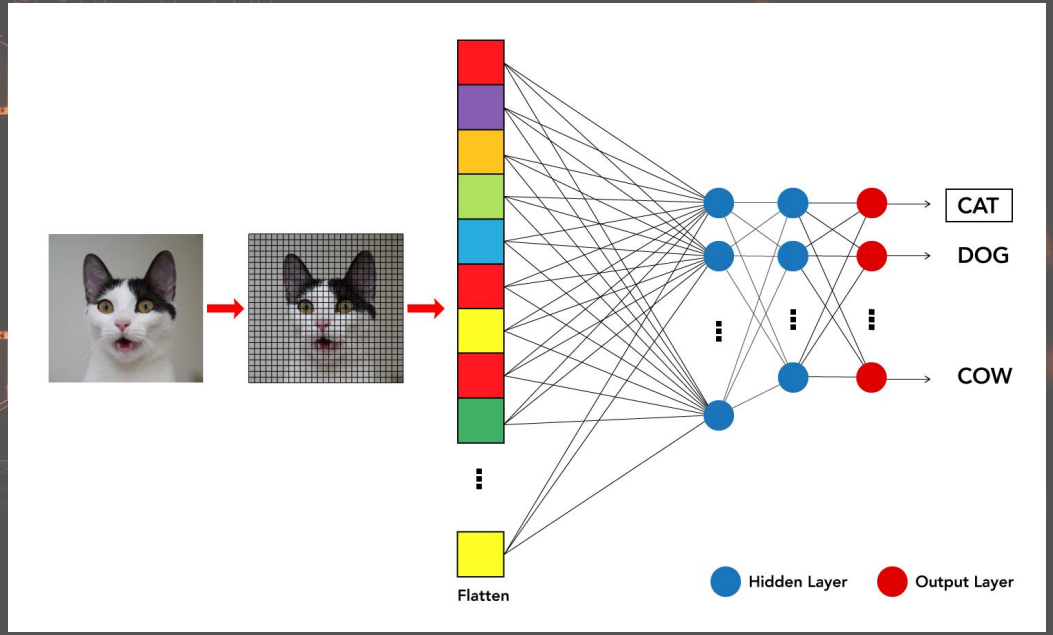




- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

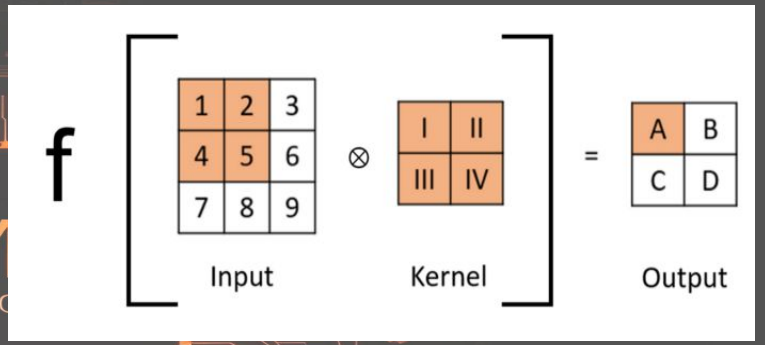
Image Recognition

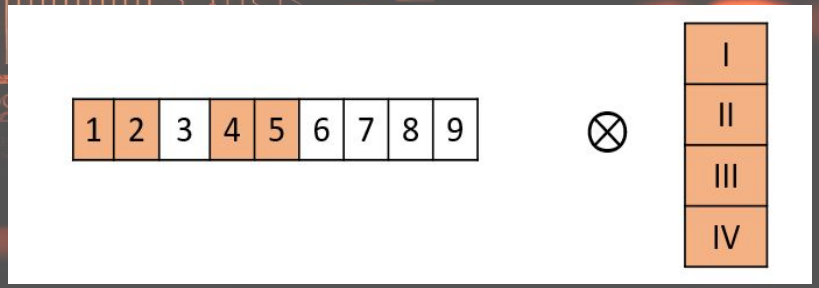
- Recognition of people, animals, objects, places etc from digital images
- Trained using thousands of pre-labelled images
- Uses the pixels in each image as descriptors
- Trained to recognise if the image shows a certain class



Convolutional Layers – extracting feature

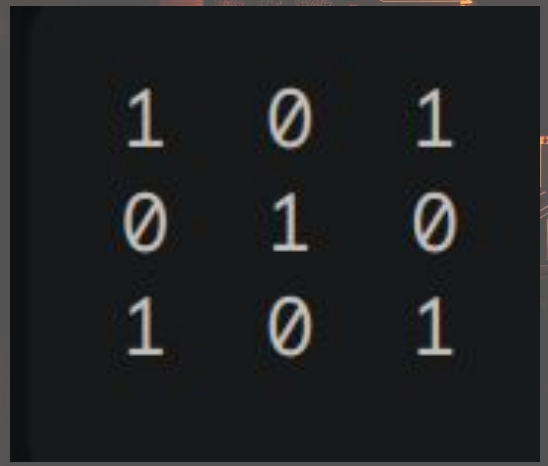
- An image is a cuboid having its length, width (dimension of the image), and height (i.e the channel 3 channels for RGB
- Kernel slides across the height and width of the image input and dot product of the kernel and the image are computed

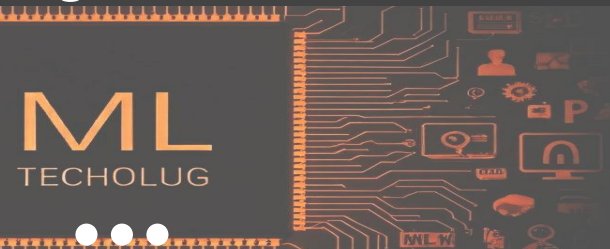




Convolutional Layers – extracting feature

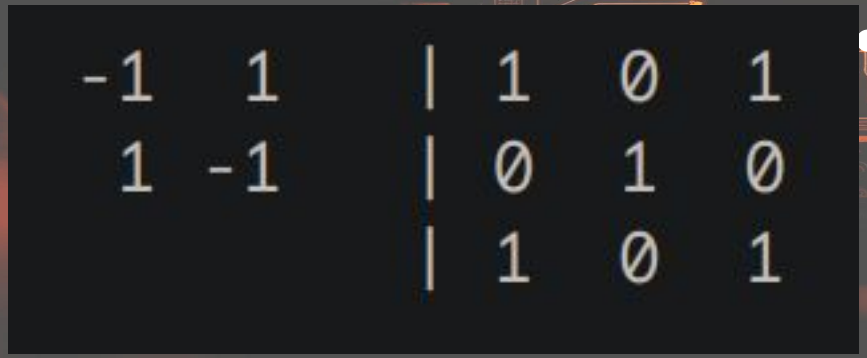
Imagine this 3x3 Black an White image: and consider a 2x2 filter:





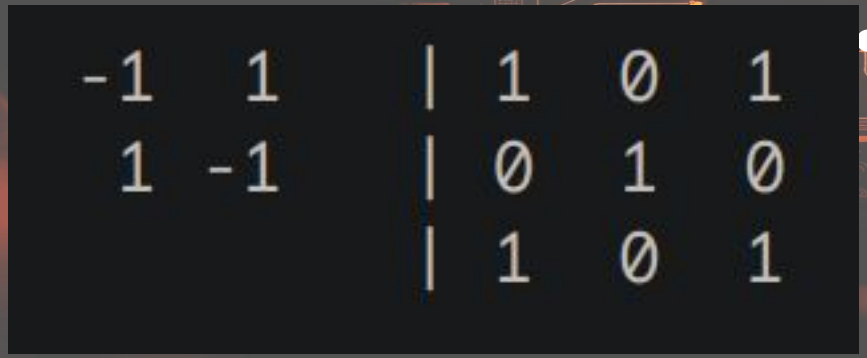
Convolutional Layers – extracting feature

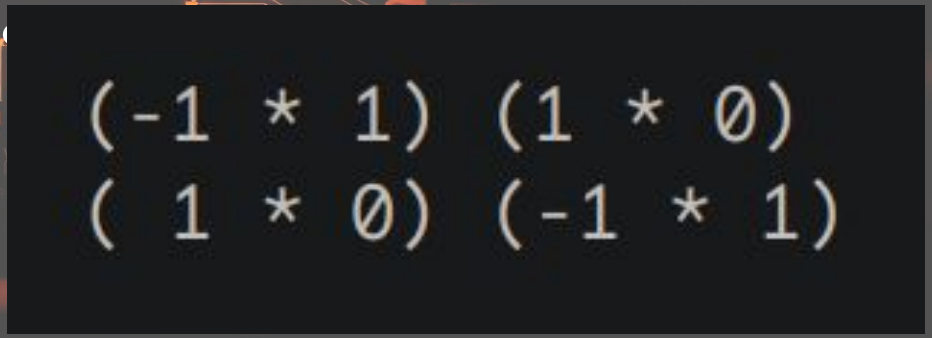
Place the filter: We start by placing the filter in the top-left corner of the image:



Convolutional Layers – extracting feature

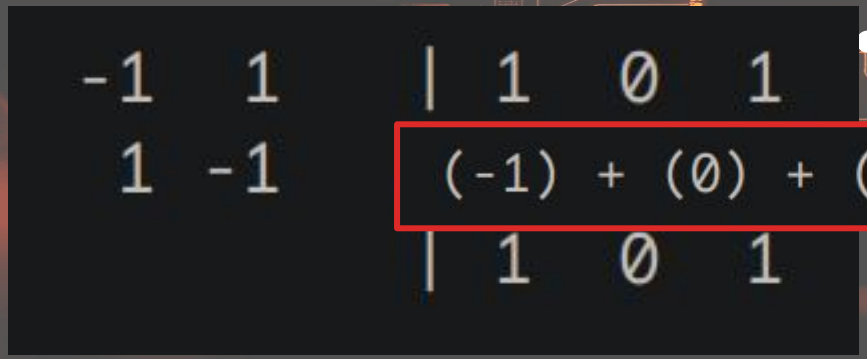
Element-wise multiplication: Multiply the corresponding elements of the filter and the image patch:

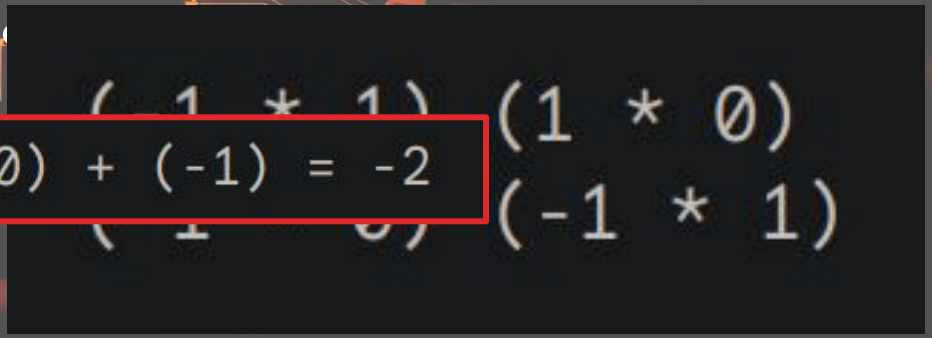




Convolutional Layers – extracting feature

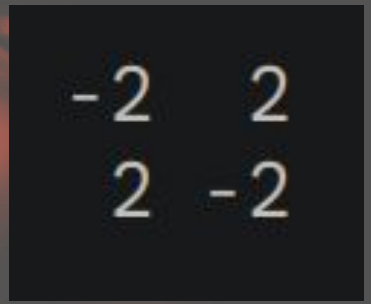
Summation: Add up the results of the multiplication:





Convolutional Layers – extracting feature

Slide the filter: we move the filter and we apply the same operations so we got the final result. After performing the convolution operation across the entire image, you'll get a smaller output matrix called a feature map. In this case, the feature map would be a 2x2 matrix:



This particular **filter is an example of an edge detection f**ilter. It highlights regions in the image where there's a change in intensity (from light to dark or dark to light). The negative values in the output feature map correspond to edges where the intensity decreases from left to right or top to bottom, while the positive values correspond to edges where the intensity increases.

Convolutional Layers – extracting feature

Slide the filter: we move the filter and we apply the same operations so we got the final result. After performing the convolution operation across the entire image, you'll ge feature map would b

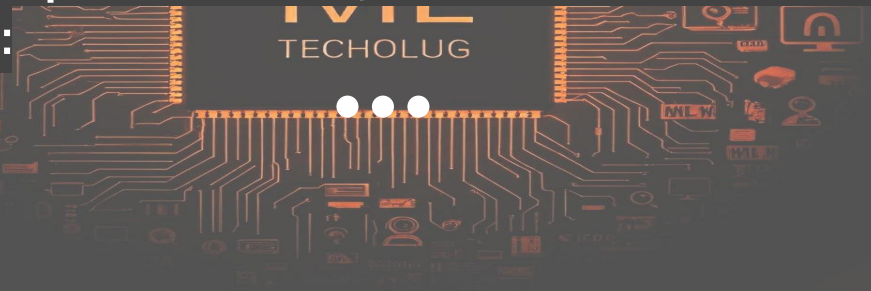
In a CNN, the activation function is applied to the output of each filter, to introduce non-linearity and shaping the feature representation.

etection filter. It
ge in intensity
lues in the output
y decreases from
es correspond to
edges where the intensity increases.

Convolutional Layers – extracting feature

During the training of a Convolutional Neural Network (CNN), the values within the filters (also called kernels or weights) are what change and are learned. This is how the network adapts and improves its ability to detect relevant features in the input data. Similarly to the NN:

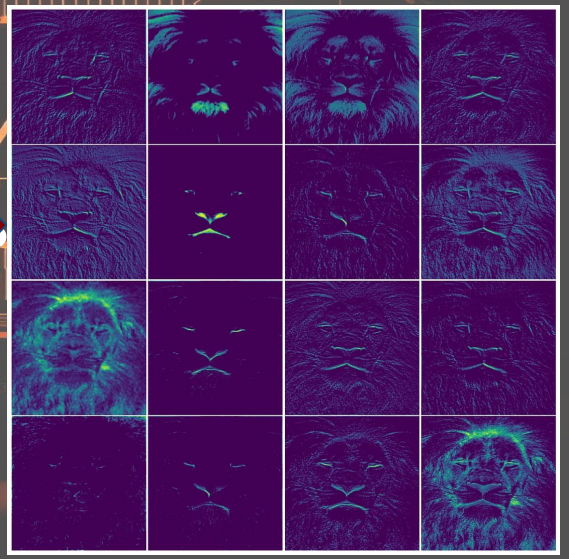
- Initialization
- Forward Pass
- Loss Calculation
- Backpropagation
- Weight Update.



Iteration:

Convolutional Layers – extracting feature

Convolutional layers often detect edges and geometries in the image (Colors: RGB three channels)



Predicting Gene Accessibility using CNNs

Kelley DR, Snoek J, Rinn JL. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Research*. 2016;26(7):990-999. doi:10.1101/gr.200535.115.

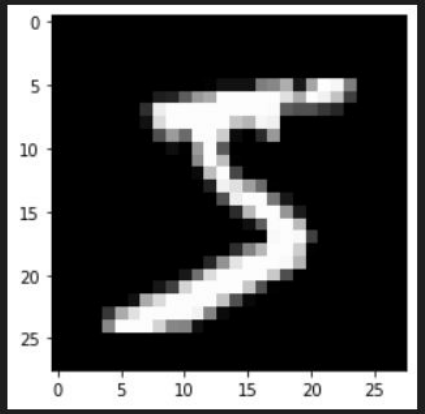
Convolutional Layers – extracting feature

```
(trainX, trainy), (testX, testy) = mnist.load_data()
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print(' Test: X=%s, y=%s' % (testX.shape, testy.shape))
print('')
for i in range(9):
  #plt.subplot(330 + 1 + i)
  print(trainX[i].shape)
  print(trainy[i])
  plt.imshow(trainX[i], cmap=plt.get_cmap('gray'))
  plt.show()
```

Convolu

```
(trainX,
print('T
print('
print(''
for i in
  #plt.s
  print(
  print(
  plt.im
  plt.sh
```

```
Train: X=(60000, 28, 28), y=(60000,)
 Test: X=(10000, 28, 28), y=(10000,)

(28, 28)
5
```




```
(28, 28)
0
```

```
e))
)
```

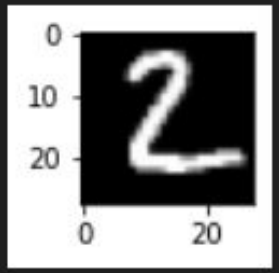
Convolutional Layers – extracting feature

```
(trainX, trainY), (testX, testY) = mnist.load_data()
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28, 1))
trainY = to_categorical(trainY)
testY = to_categorical(testY)

if DEBUGVIS:
    print(testY[1])
    plt.subplot(330 + 1)
    plt.imshow(testX[1], cmap=plt.get_cmap('gray'))#
    plt.show()
```

Convolutional Layers – extracting feature

```
(trainX, trainY), (testX, testY) = mnist.load_data()
trai                                                  1))
test
trai
test

if D
  pr
  pl
  plt.imshow(testX[1], cmap=plt.get_cmap('gray'))#
  plt.show()
```

```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```



```
trainX = trainX.astype('float32')
testX = testX.astype('float32')
trainX = trainX / 255.0
testX = testX / 255.0

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', \
                 kernel_initializer='he_uniform', \
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu', \
                kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', \
              metrics=['accuracy'])
```

```
trainX = trainX.astype('float32')
testX = testX.astype('float32')
trainX = trainX / 255.0
testX = testX / 255.0

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', \
                 kernel_initializer='he_uniform', \
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu', \
                kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', \
              metrics=['accuracy'])
```

32 filters/kernels each one 3x3 the input is a grayscale image, 1 channel only the ReLU is applied to each output

```
trainX = trainX.astype('float32')
testX = testX.astype('float32')
trainX = trainX / 255.0
testX = testX / 255.0

model = Sequential()
model.add(Conv2D(32, (3, 3), activati
                 kernel_initialize
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu', \
                kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', \
              metrics=['accuracy'])
```

This is **a max pooling layer, which reduces the spatial dimensions of the feature** maps generated by the convolutional layer. (2, 2): Specifies the size of the pooling window (2x2 pixels). This means **the layer will take the maximum value in each 2x2 region of the feature map.**

Convolutional Layers – extracting feature

```
trainX, validX, trainY, validY = train_test_split(trainX, trainY, \
                                                  test_size=0.20, random_s

history = model.fit(trainX, trainY, epochs=10, \
                    batch_size=32, validation_data=(validX, validY), verbose=1)
_, acc = model.evaluate(validX, validY, verbose=0)
print('Validation Set > %.3f' % (acc * 100.0))
_, acc = model.evaluate(trainX, trainY, verbose=0)
print('Training Set > %.3f' % (acc * 100.0))

plt.clf()
plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='orange', label='test')
plt.show()
```

Convolutional Layers – extracting feature

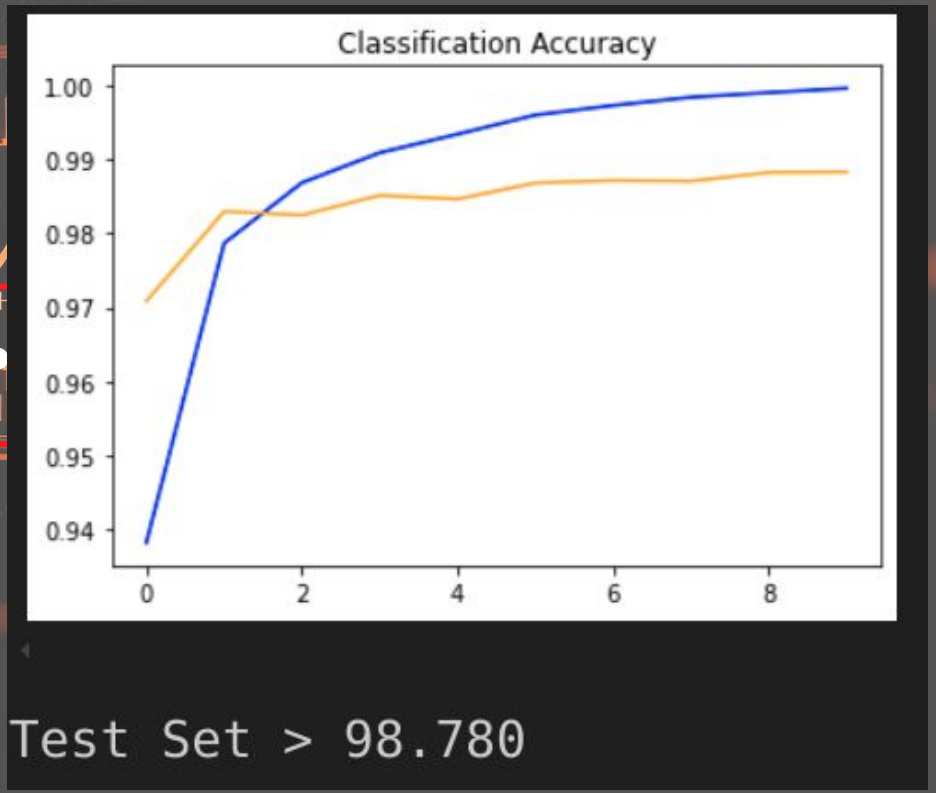
```
trainX, validX, trainY, validY = train_test_split(trainX, trainY, \
                                                  test_size=0.20, random_s
```

```
1500/1500 [==============================] - 21s 14ms/
Epoch 10/10
1500/1500 [==============================] - 20s 13ms/
Validation Set > 98.825
Training Set > 99.979
```

```
plt.clf()
plt.title('Classification Accuracy')
plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='orange', label='test')
plt.show()
```

Convolutional Layers – extracting feature

conv2d_1_input	InputLayer	input:	[(None, 28, 28, 1)]
		output:	[(None, 28, 28, 1)]
conv2d_1	Conv2D	input:	(None, 28, 28, 1)
		output:	(None, 26, 26, 32)
max_pooling2d_1	MaxPooling2D	input:	(None, 26, 26, 32)
		output:	(None, 13, 13, 32)
flatten_1	Flatten	input:	(None, 13, 13, 32)
		output:	(None, 5408)
dense_2	Dense	input:	(None, 5408)
		output:	(None, 100)
dense_3	Dense	input:	(None, 100)
		output:	(None, 10)



ML
ROC/AUC

ROC/AUC

ROC (Receiver Operating Characteristic) and AUC (Area Under the Curve) are powerful tools for evaluating the performance of classification models.

- True Positive Rate (TPR) / Sensitivity / Recall: The proportion of actual positive cases that are correctly identified by the model.

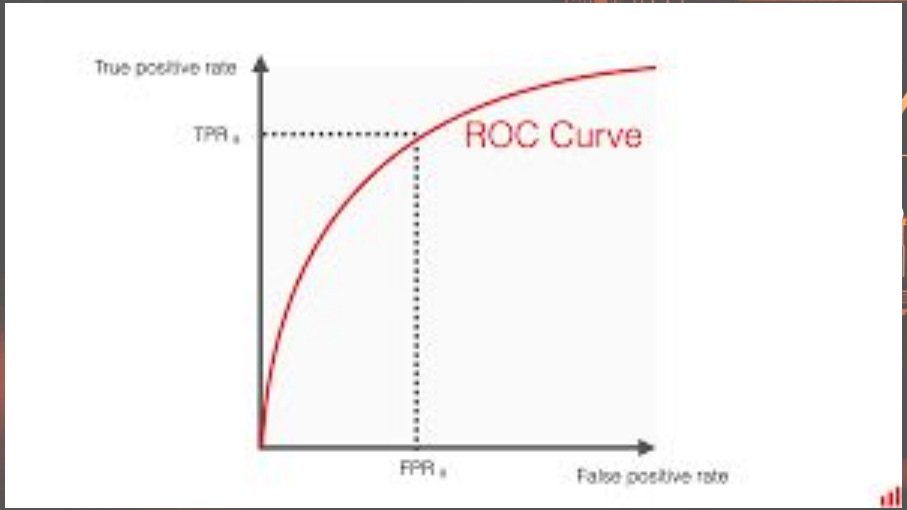
TECHOLUG

```
TPR = TP / (TP + FN)
```

- False Positive Rate (FPR) / 1 - Specificity: The proportion of actual negative cases that are incorrectly classified as positive.

```
FPR = FP / (FP + TN)
```

ROC/AUC

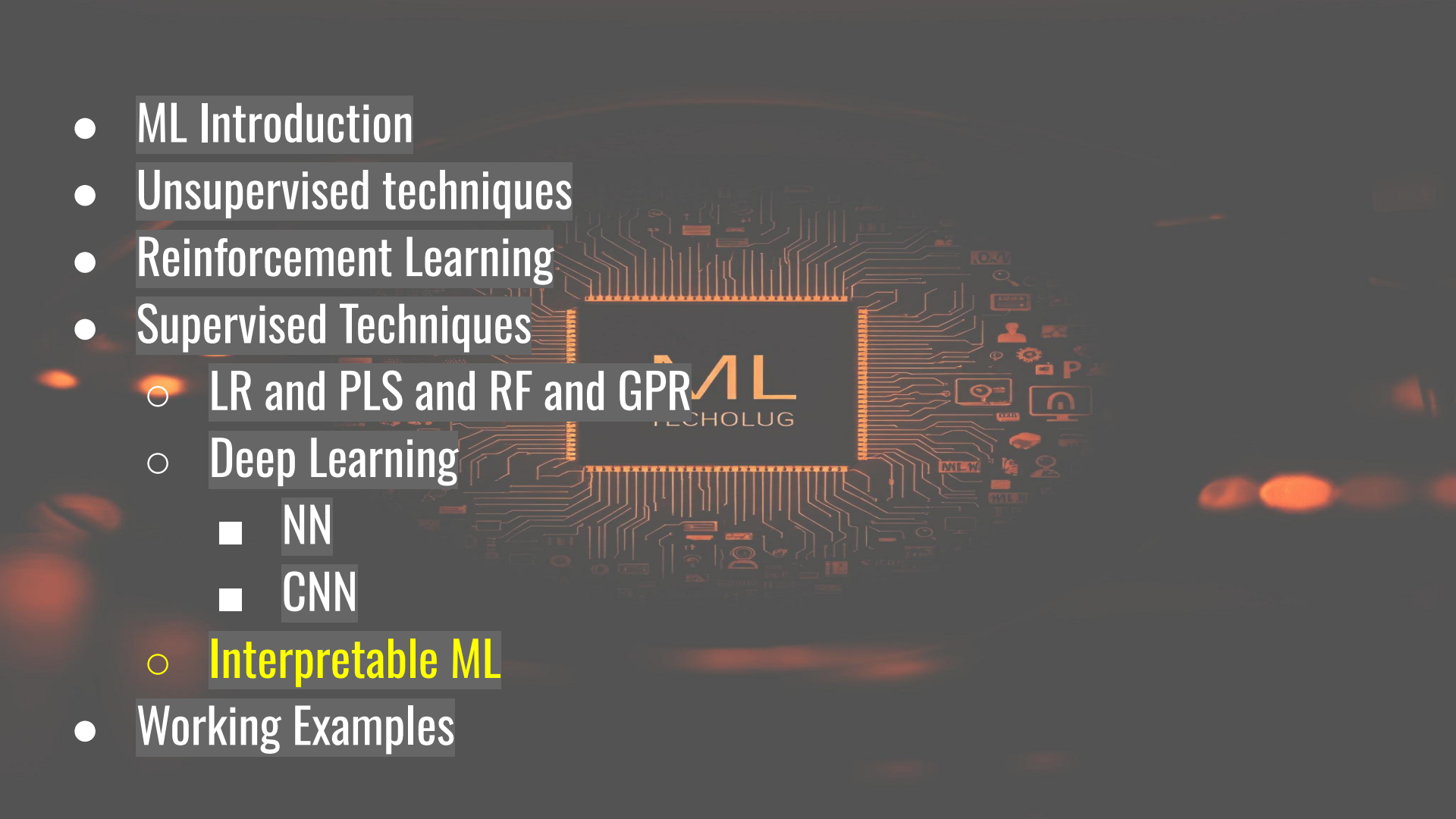


True Positive Rate (TPR) against the False Positive Rate (FPR) at different threshold settings.

Predictions: Obtain the predicted probabilities from your classification model for all instances in your dataset.

Thresholds: Select a range of thresholds between 0 and 1.

Calculate TPR and FPR: For each threshold:



- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

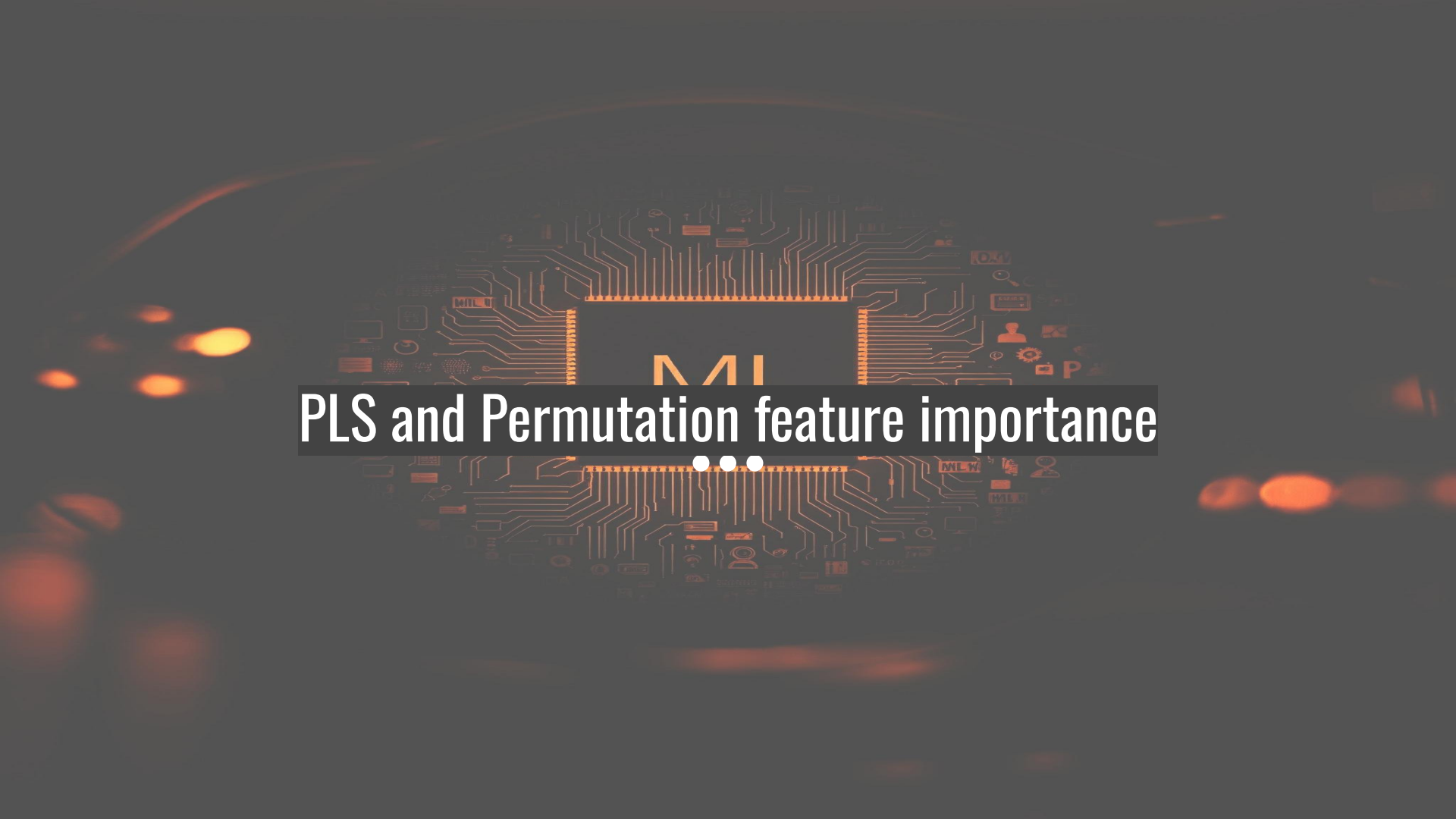
Interpretable ML

- **The Rise of Black Box Models:**
  - Machine learning models are becoming increasingly complex and accurate.
  - However, many models are "black boxes," meaning their internal workings are difficult to understand.
- The Need for Interpretability:
  - Understanding how models make decisions is crucial for trust, accountability, and fairness.
  - Interpretability helps identify biases, debug models, and gain insights into the data.

Interpretable ML

Some model are directly interepretable as Linear Regression, other model are not:

- **Model Specific: Techniques designed for particular model types (e.g., rule extraction from decision trees).**
- **Model Agnostic: Methods that work with any model, regardless of its internal structure.**


PLS and Permutation feature importance

Permutation feature importance

Permutation feature importance is a model-agnostic technique used to measure the importance of features

It works by randomly shuffling the values of a single feature and measuring how much the model's performance decreases. The more the performance drops, the more important that feature is

Permutation feature importance

Permutation feature importance is a model-agnostic technique used to measure the importance of features

1. **Baseline performance**
2. **Feature shuffling**
3. **Performance with shuffled feature**
4. **Importance calculation**
5. **Repeat for all features**

Permutation feature importance

Permutation feature importance is a model-agnostic technique used to measure the importance of features

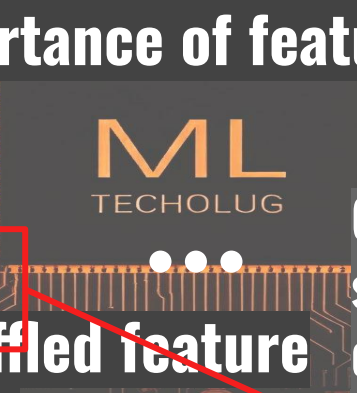
1. Baseline performance
2. Feature shuffling
3. Performance with shuffled feature
4. Importance calculation
5. Repeat for all features

Train your model on the original dataset and evaluate its performance using a suitable metric (e.g., accuracy, F1-score, R-squared). This establishes a baseline performance.

Permutation feature importance

Permutation feature importance is a model-agnostic technique used to measure the importance of features

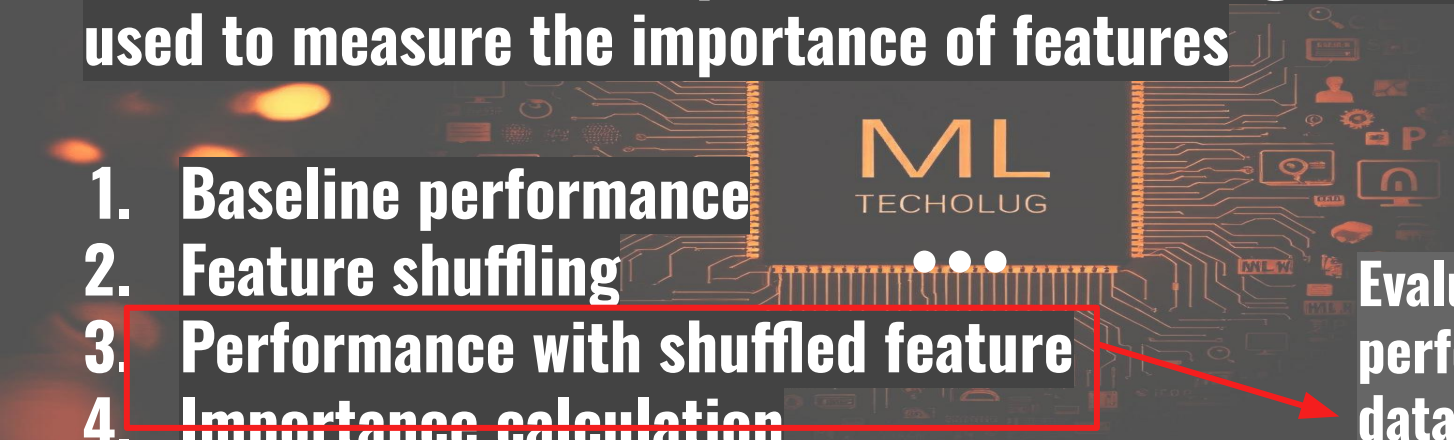
1. Baseline performance
2. Feature shuffling
3. Performance with shuffled feature
4. Importance calculation
5. Repeat for all features



Choose a feature and randomly shuffle its values within the dataset. This breaks the relationship between that feature and the target variable.

Permutation feature importance

Permutation feature importance is a model-agnostic technique used to measure the importance of features



1. Baseline performance
2. Feature shuffling
3. Performance with shuffled feature
4. Importance calculation
5. Repeat for all features

Evaluate the model's performance on the dataset with the shuffled feature.

Permutation feature importance

Permutation feature importance is a model-agnostic technique used to measure the importance of features

1. Baseline performance
2. Feature shuffling
3. Performance with shuffled feature
4. Importance calculation
5. Repeat for all features

Calculate the difference in performance between the baseline (original data) and the shuffled data. This difference represents the importance of the feature. A larger drop in performance indicates a more important feature.

Permutation feature importance

Permutation feature importance is a model-agnostic technique used to measure the importance of features

1. Baseline performance
2. Feature shuffling
3. Performance with shuffled feature
4. Importance calculation
5. Repeat for all features

Repeat steps 2-4 for each feature in your dataset to get an importance score for each feature.

Permutation feature importance

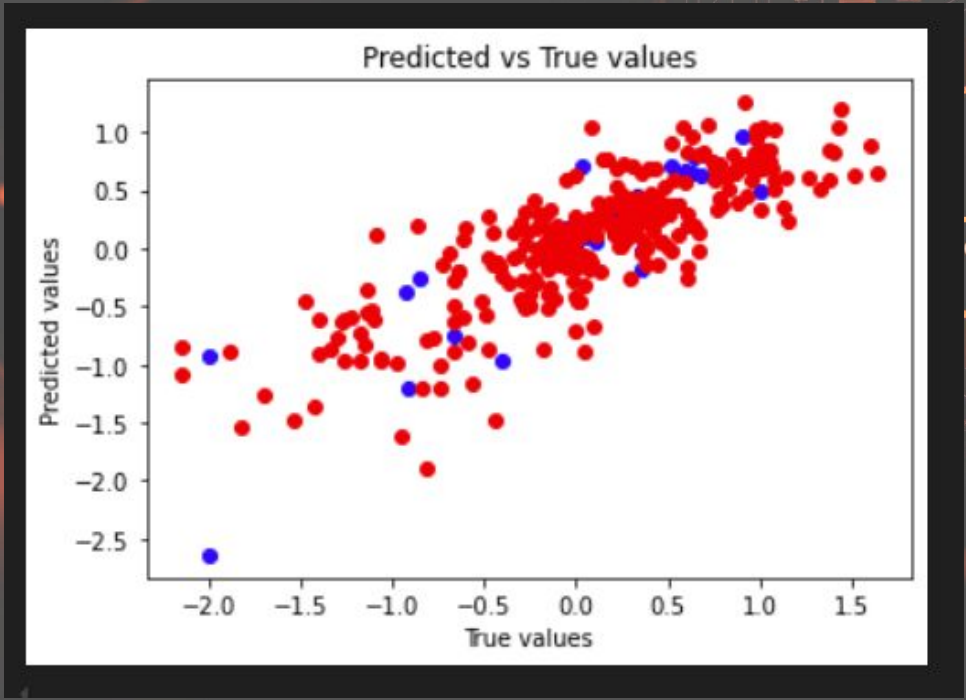
```
newdf = newdf.dropna(axis=1)
print(newdf.shape)
corr_matrix = newdf.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)
newdf = newdf.drop(newdf[to_drop], axis=1)
print(newdf.shape)
Y = newdf['logBB']
X = newdf.drop(['Objects', 'logBB', 'LgBB'], axis=1)
```

Permutation feature importance

```
pls = PLSRegression(n_components=20)
pls.fit(X_train, y_train)
y_pred_test = pls.predict(X_test)
msetest = mean_squared_error(y_test, y_pred_test)
r2test = r2_score(y_test, y_pred_test)

y_pred_train = pls.predict(X_train)
msetrain = mean_squared_error(y_train, y_pred_train)
r2train = r2_score(y_train, y_pred_train)
```

0.0s

Permutation feature importance



MSE test:	0.12616
R2 test:	0.75701
MSE train:	0.16289
R2 train:	0.66275

Permutation feature importance

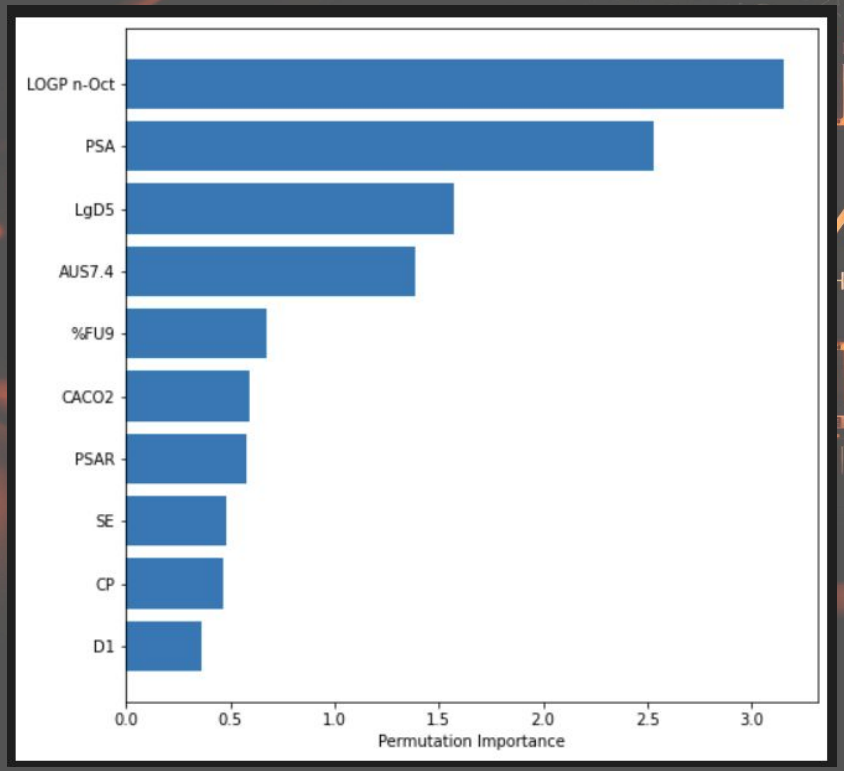
```
from sklearn.inspection import permutation_importance

result = permutation_importance(pls, X_train, y_train, \
                                n_repeats=10, random_state=0, \
                                n_jobs=2, scoring='r2')
sorted_idx = result.importances_mean.argsort()
sorted_idx_most = sorted_idx[-10:]
#sorted_idx_most = sorted_idx_most[::-1]

plt.barh(X.columns[sorted_idx_most], \
         result.importances_mean[sorted_idx_most])
plt.xlabel('Permutation Importance')
plt.show()
```

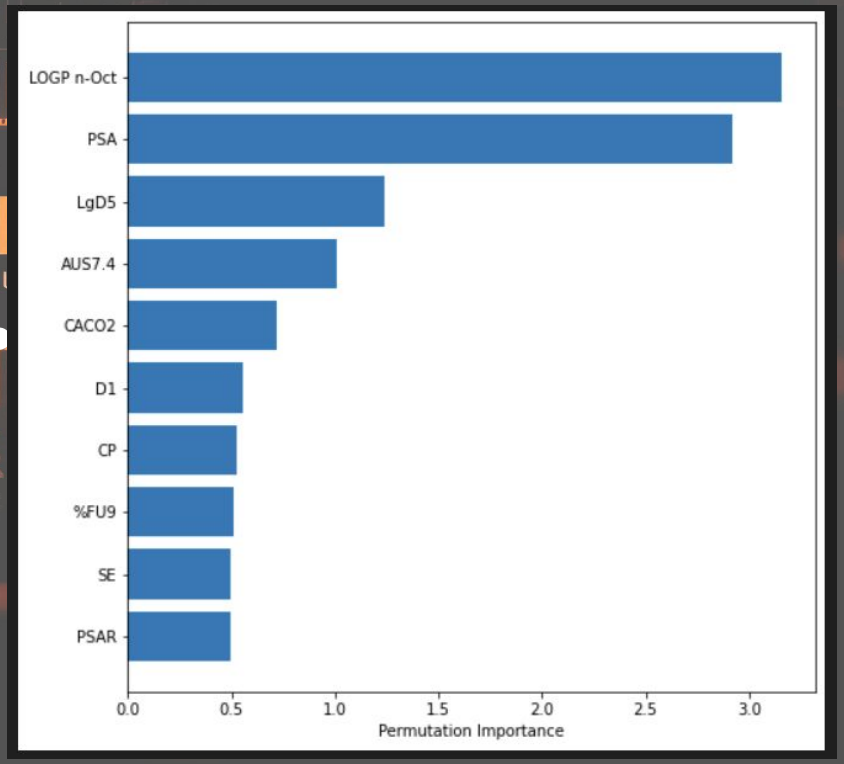
Permutation feature importance

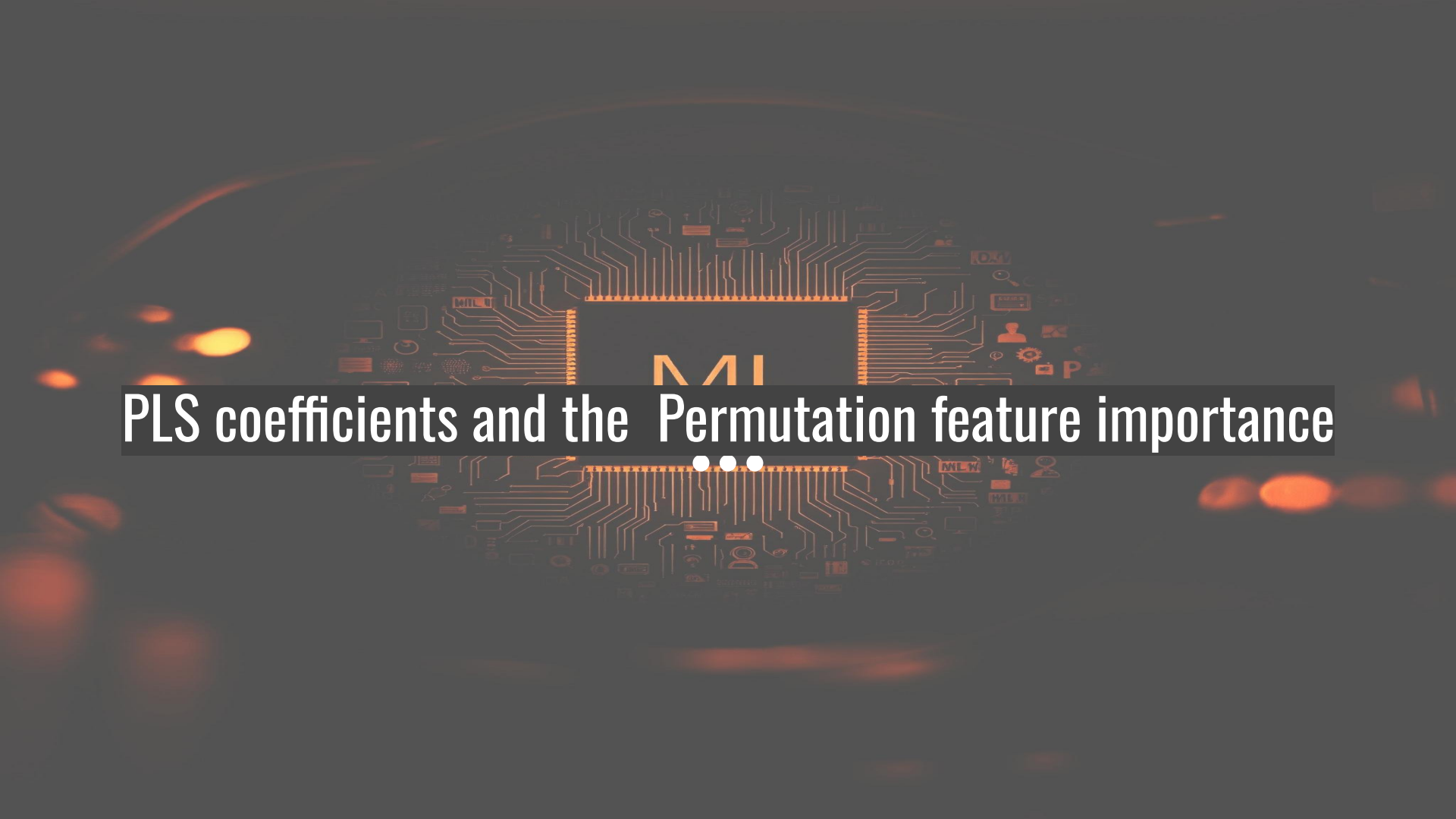
```
from sklearn.inspection import permutation_importance
```

```
from sklearn.inspection import permutation_importance
result = permutation_importance(pls, X_test, y_test, \
                                n_repeats=10, random_state=0, \
                                n_jobs=2, scoring='r2')
sorted_idx = result.importances_mean.argsort()
sorted_idx_most = sorted_idx[-10:]

plt.barh(X.columns[sorted_idx_most], \
         result.importances_mean[sorted_idx_most])
plt.xlabel('Permutation Importance')
plt.show()
```

Permutation feature importance






PLS coefficients and the Permutation feature importance
...

Permutation feature importance

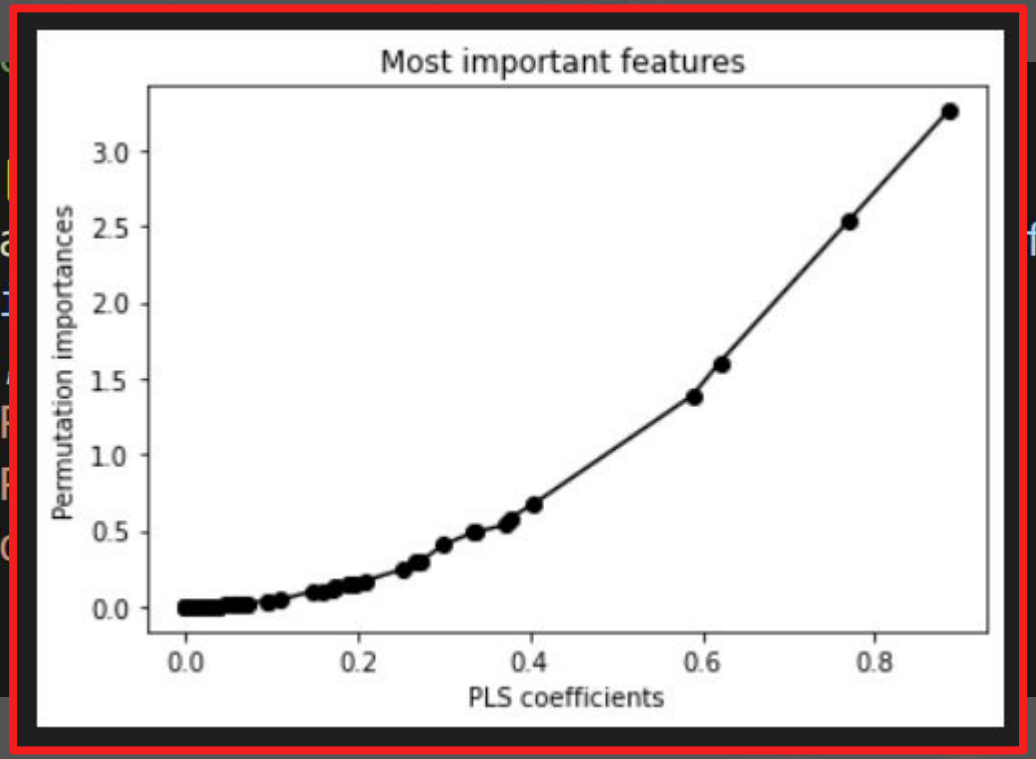
```
most_importante_features = []
result = permutation_importance(pls, X, Y, n_repeats=10, \
                                random_state=42, n_jobs=2,
                                scoring='r2')
pfi_sorted_idx = result.importances_mean.argsort()
#compute absolute values of the PLS coefficients
coef = np.abs(pls.coef_).flatten()
#sort the coefficients
sorted_idx = np.argsort(coef)
```

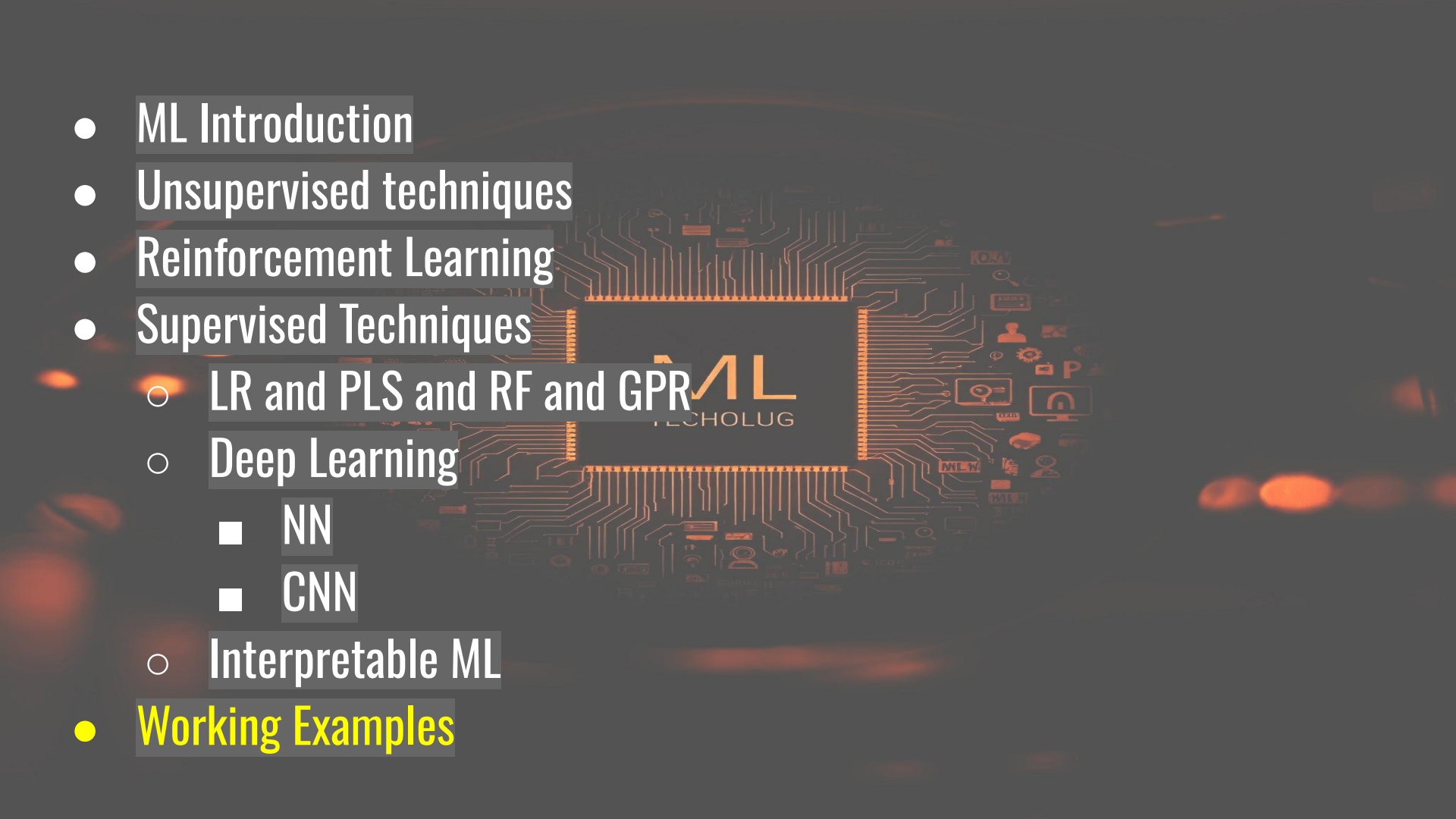
Permutation feature importance

```
# Scatter plot of the most important features
plt.clf()
plt.rcParams['figure.figsize'] = [8, 8]
fis = [np.mean(result.importances[i].T) for i in pfi_sorted_idx]
cfs = [coef[i] for i in sorted_idx]
plt.plot(cfs, fis, '-o', color='black')
plt.xlabel("PLS coefficients")
plt.ylabel("Permutation importances")
plt.title("Most important features ")
plt.show()
```

Permutation feature importance

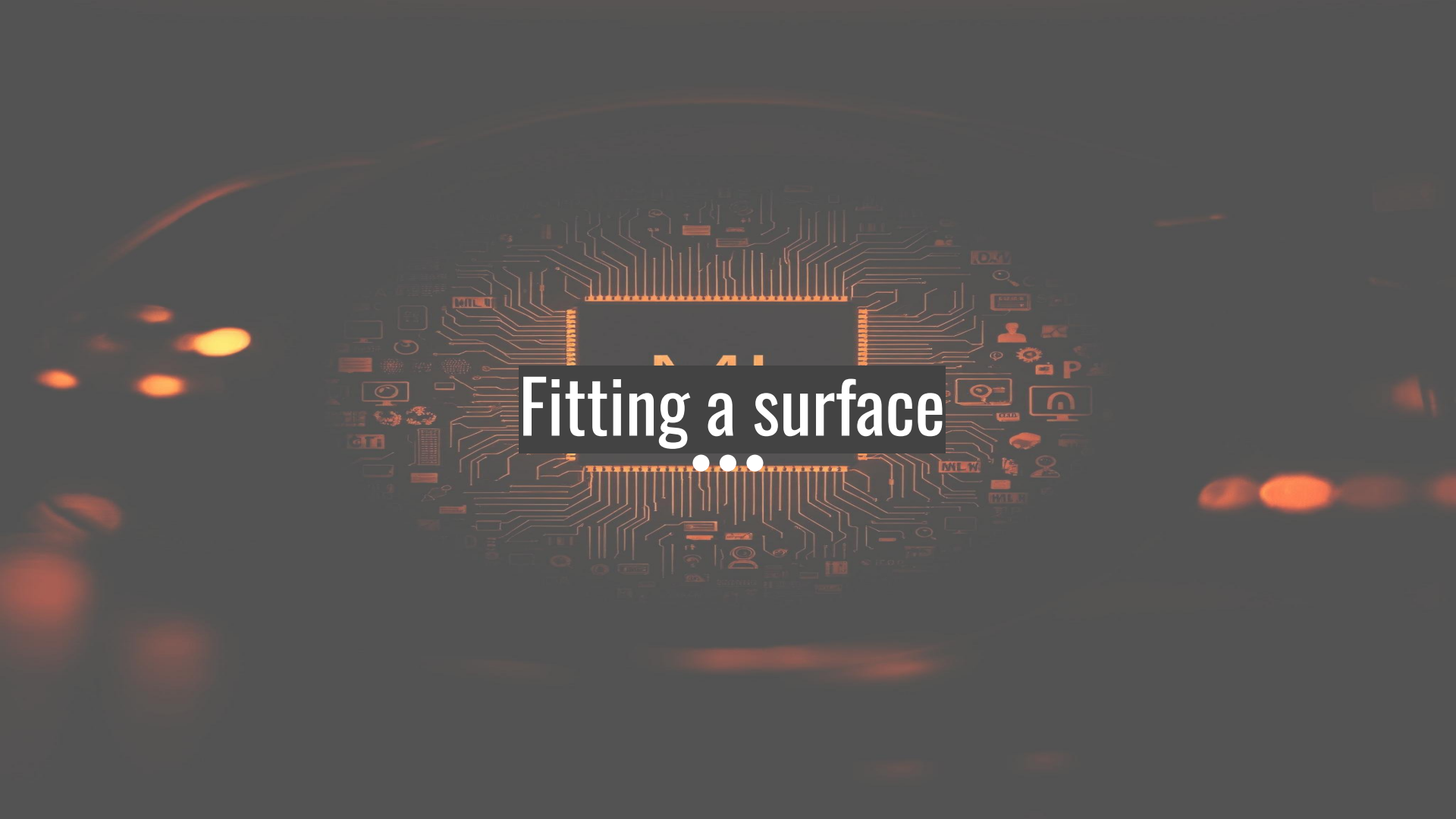
```
plt.clf()
plt.rcParams[
fis = [np.mea                                    fi_sorted_idx]
cfs = [coef[i
plt.plot(cfs,
plt.xlabel("F
plt.ylabel("F
plt.title("Mo
plt.show()
```





- ML Introduction
- Unsupervised techniques
- Reinforcement Learning
- Supervised Techniques
  - LR and PLS and RF and GPR
  - Deep Learning
    - NN
    - CNN
  - Interpretable ML
- Working Examples

Fitting a surface



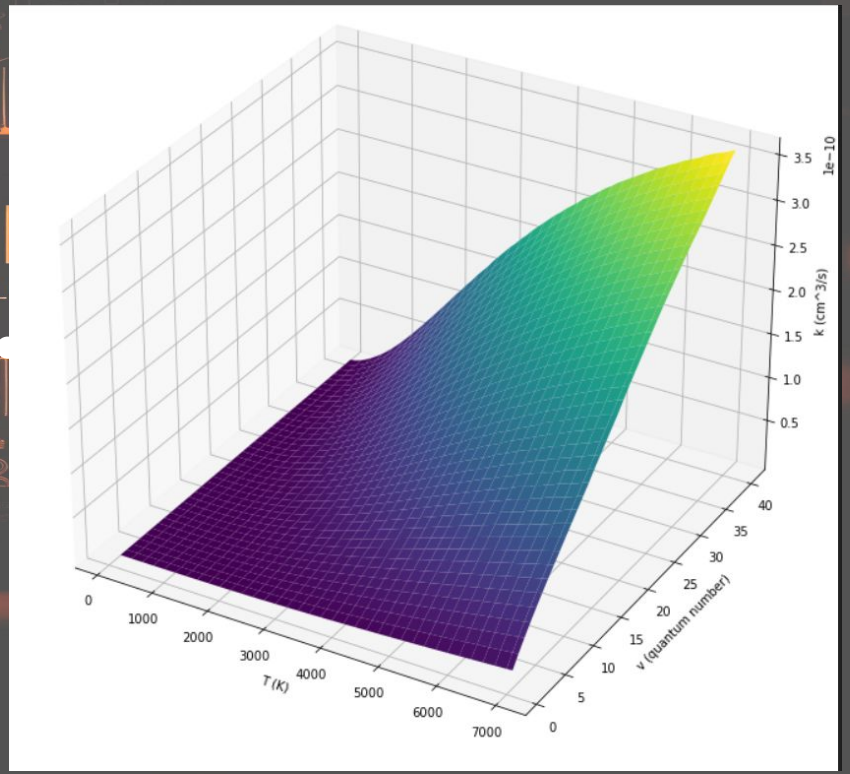
N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients

$$N_2(v) + H_2(0) \longrightarrow N_2(v - \Delta v) + H_2(0), \quad \Delta v = 1, 2, 3.$$

$\log_{10}(k)$ is the label

v, T are the two features

We want to test the performances of ttwo models NN and GPR



N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients

- Rate coefficients for vibrational energy transfer are calculated for collisions between molecular nitrogen and hydrogen in a wide range of temperature and of initial vibrational states
  - The calculations were performed by a mixed quantum-classical method

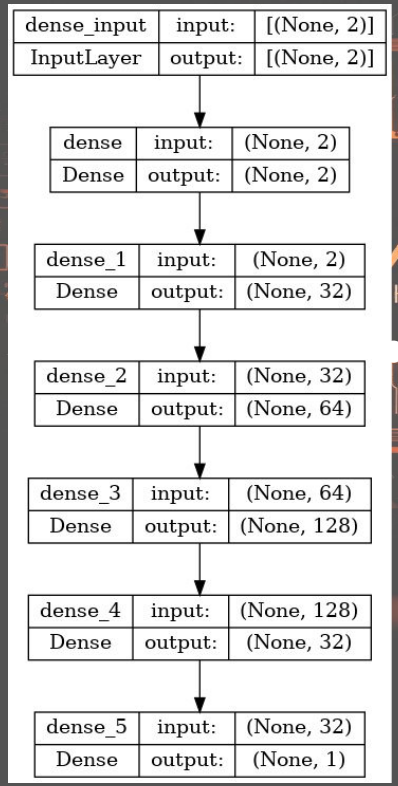
TECHOLUG

ML Goal Predict rate coefficients for vibrational energy transfer processes involving specific initial vibrational states, which are computationally expensive to calculate directly.

Qizhen Hong, Loriano Storchi, Massimiliano Bartolomei, Fernando Pirani, Quanhua Sun, Cecilia Coletti, "Inelastic N2+H2 collisions and quantum-classical rate coefficients: large datasets and machine learning predictions" The European Physical Journal D, DOI: 10.1140/epjd/s10053-023-00688-4 (2023

N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients

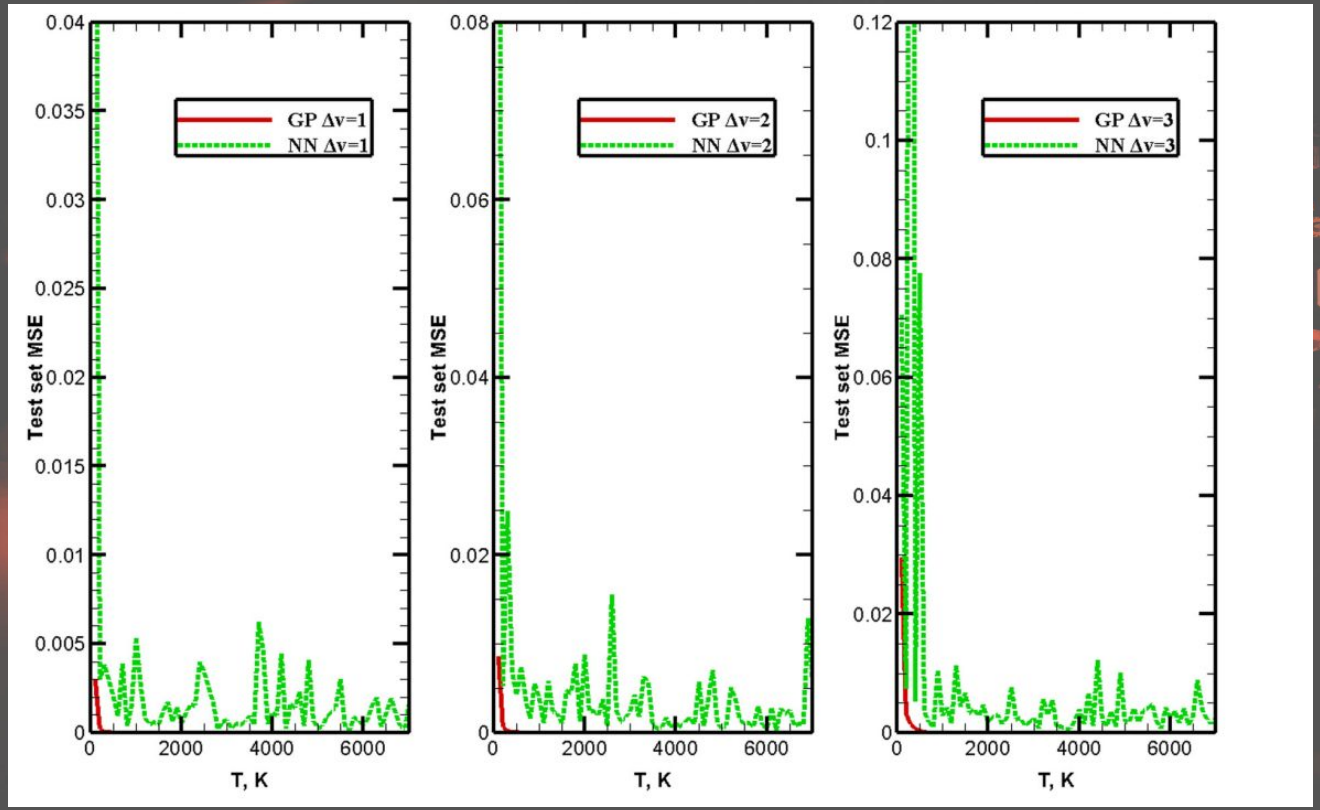
NN model unsinf Linear activation in input and output and ReLU



GPR using Matern Kernel
v = 5/2

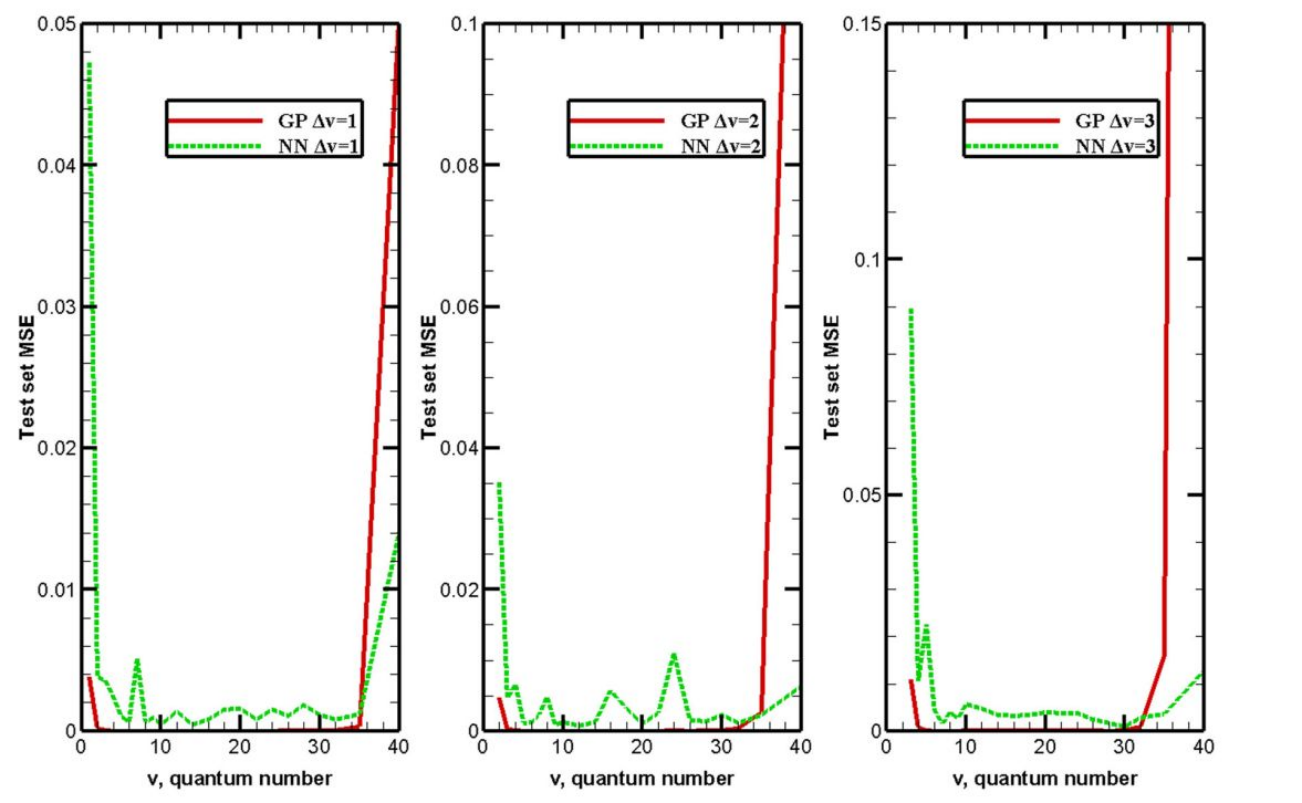
$$k(x_i, x_j) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{l}d(x_i, x_j)\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l}d(x_i, x_j)\right)$$

N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients



Test set MSE values as a function of temperature: $\log_{10}$ (k) values corresponding to a specific temperature T were removed from the training set and constitute the test set. The three panels correspond to processes (5) with Δv = 1, 2, 3, respectively

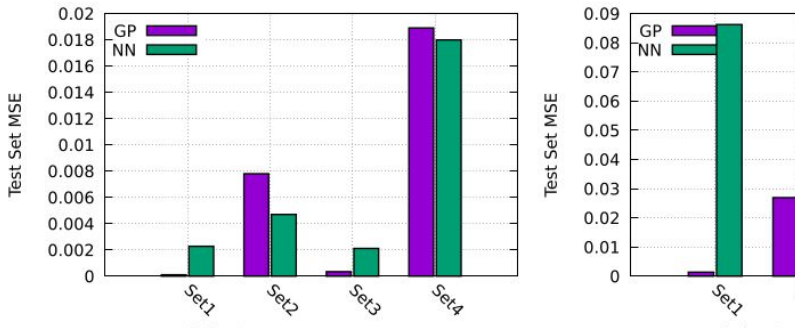
N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients



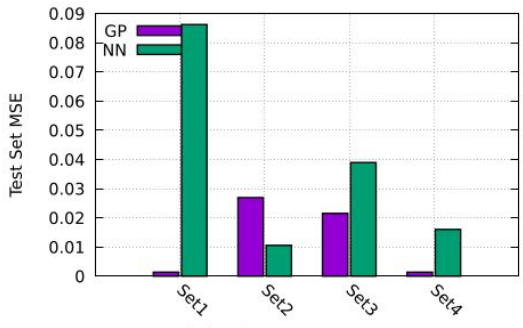
E values as a
nitial
quantum
g_{10} (k) values
ng to a
ere removed
ining set and
he test set.
anels
to processes

respectively

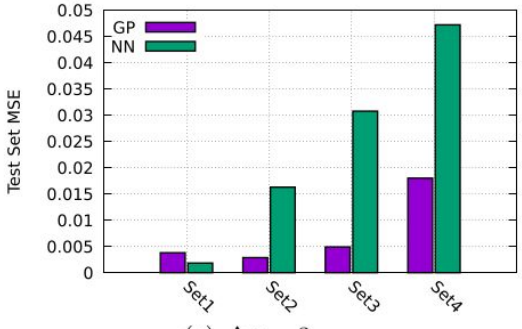
N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients



(a) $\Delta v = 1$



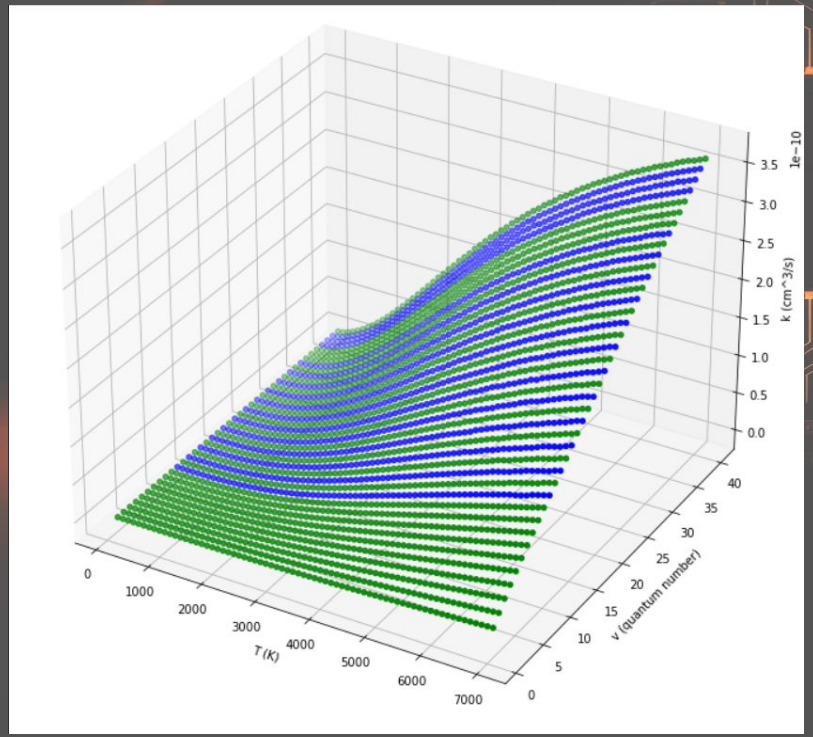
(b) $\Delta v = 2$



(c) $\Delta v = 3$

The test set MSE values for the two models obtained by removing an increasing number of systematically selected points, corresponding to specific v values, from the training set, i.e., Set1, removed v = [2; 4; 6; 8; 10; 14; 18; 22; 26; 30; 35], Set2, removed v = [1; 3; 5; 7; 9; 12; 16; 20; 24; 28; 32; 40], Set3, removed v = [2; 3; 5; 6; 8; 9; 12; 14; 18; 20; 24; 26; 30; 32], Set4, removed v = [1; 2; 4; 5; 7; 8; 10; 12; 16; 18; 22; 24; 28; 30; 35; 40]. The three panels correspond to processes (5) with Δv = 1, 2, 3, respectively

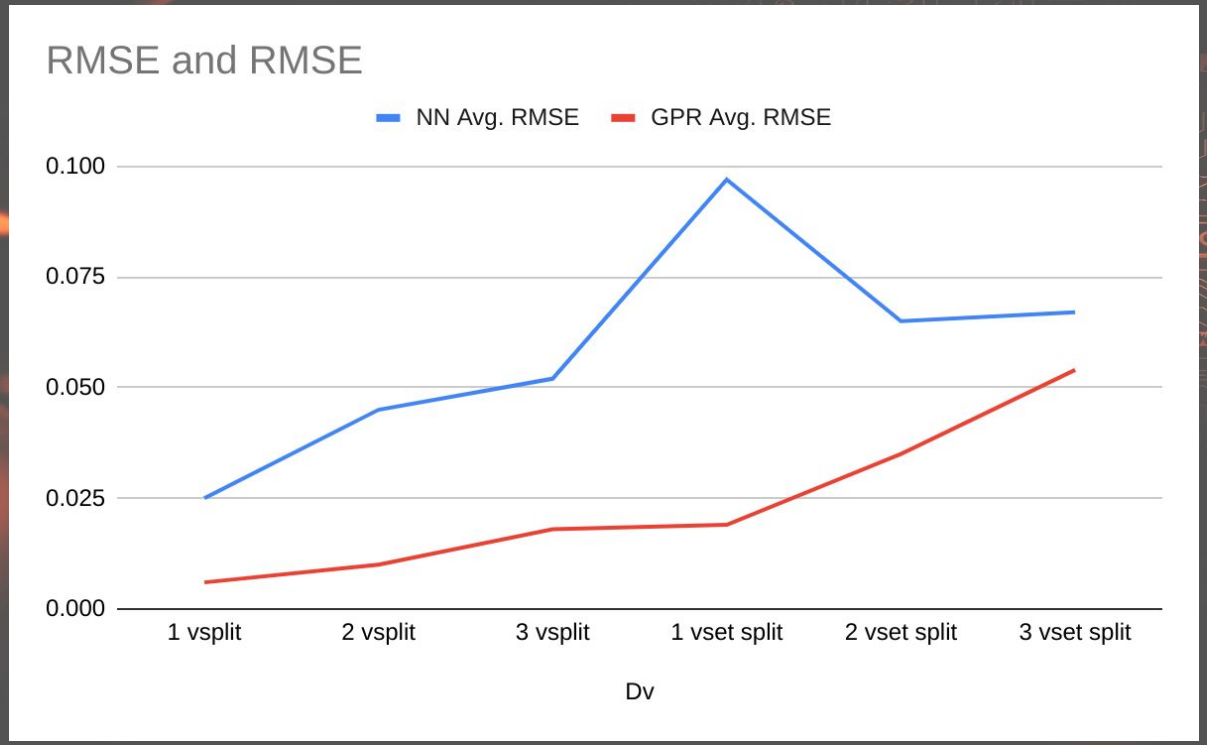
N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients



GPR $\Delta v = 1$

- Blue [points are the predicted ones, while the green points are the training set

N_2-H_2 Inelastic Collisions mixed quantum-classical rate coefficients



Preliminary new results after a deeper grid search of better hyperparameters

NN [64; 64; 64] batch 10 epochs 100

GPT Mattern Kernel v = 2

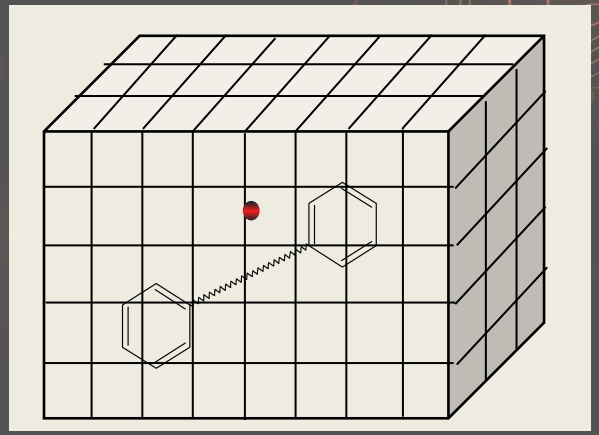
GRID MIF and PLS

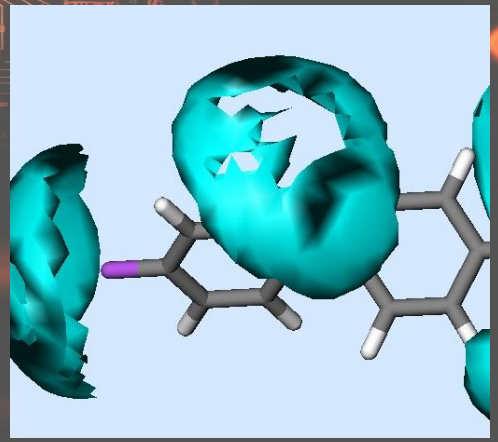
Machine Learning and the GRID Force-Fields

- **GRID program**: a computational procedure for determining energetically favourable binding sites on molecules for functional groups of known structure through the use of PROBES.
  - The PROBE is moved through a grid of points superimposed on the target molecule (to each atoms of the target and AtomType is assigned) . Its interaction energy with the target molecule is computed by an empirical energy function

$$E_{XYZ} = \Sigma[E_{LJ}] + \Sigma[E_{HB}] + \Sigma[E_Q] + [S] \dots$$

E_{LJ} = Lennard-Jones potential E_{HR} = hydrogen bonding interaction energy E_Q = electrostatic function S= entropic term





Machine Learning and the GRID Force-Fields

We build PLS models, each model is related to a specific AT, to improve the quality of the Hydrogen-Bonding term E_{HB} that is the product of three terms terms:

- E_r based on the distance between the target and the probe given in kcal/mol
- The other two, both ranging in the interval 0–1. They are dimensionless functions of the angles t and p made by the hydrogen bond (HB) at the target and the probe atoms respectively

TECHOLUG

$$E_{HB} = E_r * E_t * E_p.$$

Er assumes relative values in case of interaction with a HB acceptor or donor complementary probe and is parametrized by two values: **Emin is the strongest hydrogen-bond attraction energy at the optimum position (Emin),** and half of the straight-line distance between donor and acceptor atom pairs which corresponds to the strongest hydrogen-bond attraction energy (Rmin).

Emin → dEmin

Sara Tortorella, Emanuele Carosati, Giovanni Bocci, Simon Cross, Gabriele Cruciani, Loriano Storchi, "Combining Machine Learning and Quantum Mechanics Yields More Chemically-Aware Molecular Descriptors for Medicinal Chemistry Applications", Journal of Computational Chemistry, DOI: 10.1002/jcc.26737 (2021)

Machine Learning and the GRID Force-Fields

The dataset is made of 66463 drug-like molecules

- We used GAMESS-US B3LYP/SVP (necessity of having a versatile basis set and method) to compute the Electrostatic Potential (EP) for each atom
- EP is converted to the so called dEmin value using linear equation derived so that for each AT all the resulting dEmin values always fall within an acceptable range

$$dEmin_{BH} = m_{BH} * EP + q_{BH}.$$

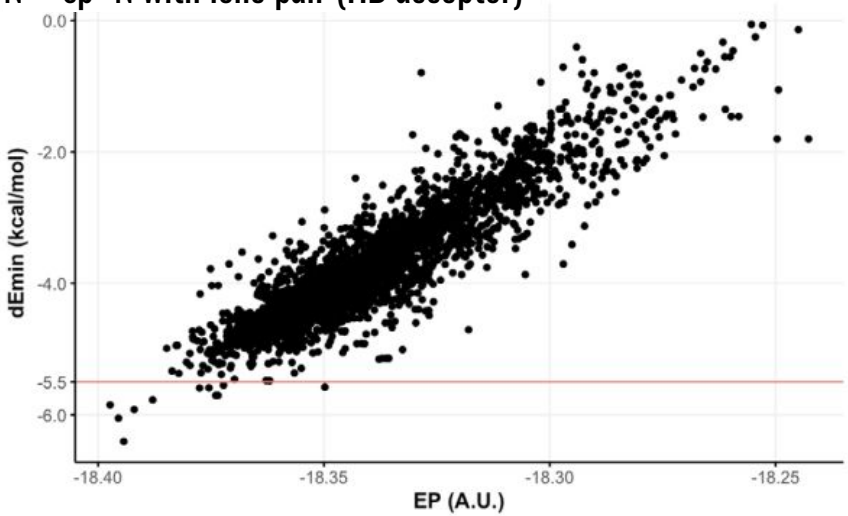
$$dEmin_{AH} = -m_{AH} * EP - q_{AH}.$$

22 equations, each one for each AtomType

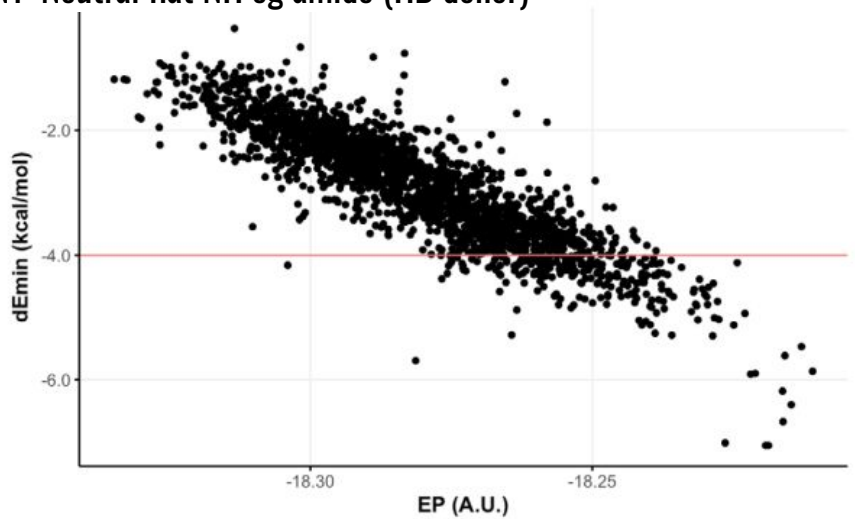
The dEmin is our label

Machine Learning and the GRID Force-Fields

N:= sp^2 N with lone pair (HB acceptor)



N1 Neutral flat NH eg amide (HB donor)



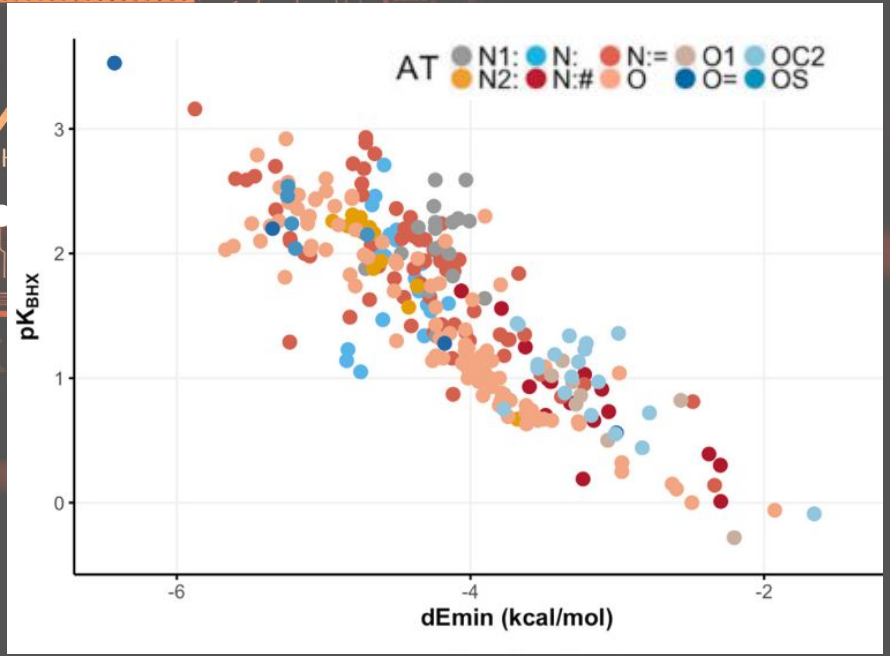
The red lines represent values of the traditional, static Emin of the GRID force field, namely -5.5 for N:= and -4.0 for N1 atom types. dEmin, dynamic Emin

Machine Learning and the GRID Force-Fields

Does chemically sound to use the dEmin in the the E_{HB} term ?

We decided to test the correlation of the proposed dEmin to those experimental hydrogen-bonding (HB) properties.

dEmin versus H-bond basicity scale for the Kenny dataset (279 atoms, R – Pearson = 0.85).



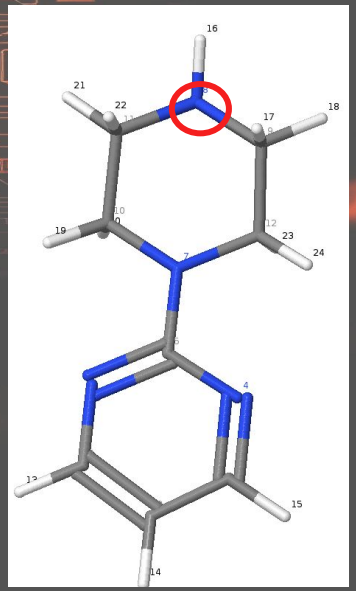
Machine Learning and the GRID Force-Fields

We have a good label, now we need to select the feature (descriptor) to use in the model

The molecular environment is described by a tree-structured molecular fingerprint with a length of 10 bond distances

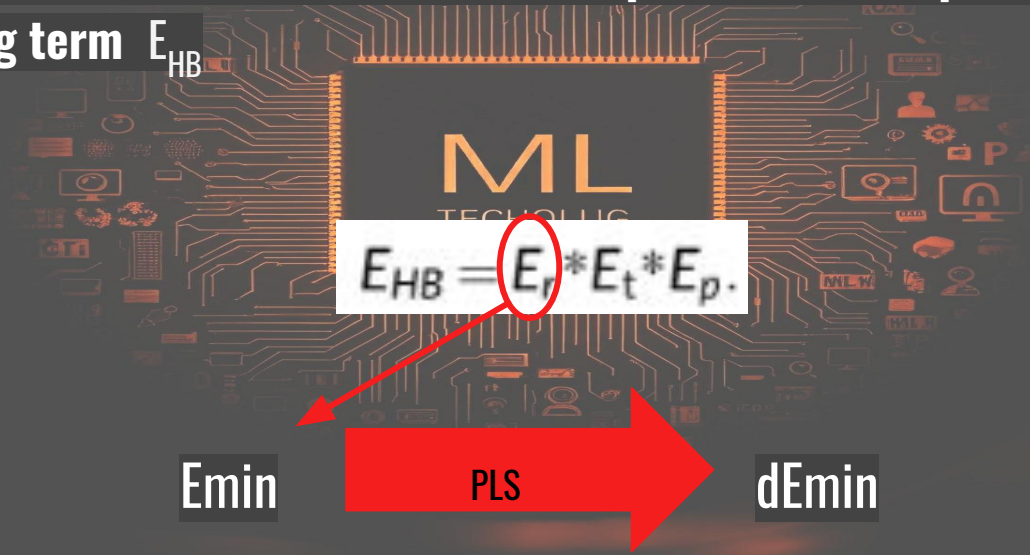
ML
TECHOLUG

```
0   1    8 N_3H 122
1   2    9 C.3 326   11 C.3 326
2   2   12 C.3 629   10 C.3 629
3   1    7 N.3_ar 1016
4   1    5 C.ar+ 1250
5   2    4 NPYM 1706    6 NPYM 1706
6   2    3 C.ar+ 1856    1 C.ar+ 1856
```



Machine Learning and the GRID Force-Fields

We build PLS models, each model is related to a specific AT, to improve the quality of the Hydrogen-Bonding term E_{HB}



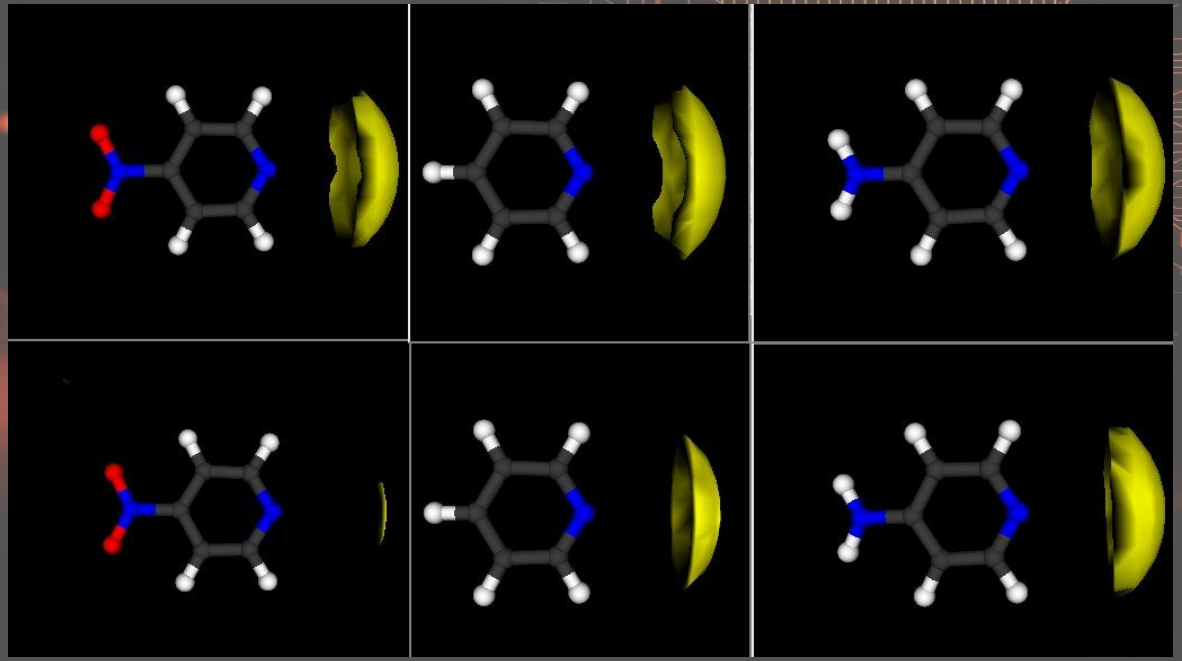
Machine Learning and the GRID Force-Fields

Using this approach, 22 PLS models were built relating atomic environment to dEmin for the HB GRID atom types (some of the models results are reported validated using leave-one-out crossvalidation)

AT	Description	H-bond type	Atoms	LV	R^2	Q^2	SDEC (kcal/Mol)	SDEP (kcal/Mol)
N:	sp3 (tertiary) nitrogen, accepting one H-bond	A	6954	9	0.92	0.88	0.56	0.41
N1:	sp3 (secondary) nitrogen, donating one hydrogen and accepting one H-bond	A	3941	8	0.91	0.84	0.24	0.49
		D	4776	7	0.96	0.92	0.30	0.53
N2:	sp3 (primary)nitrogen, donating up to two hydrogen and accepting one H-bond	A	3618	8	0.84	0.71	0.26	0.38
		D	4895	7	0.95	0.92	0.30	0.41
ON	oxygen of nitro or nitroso group, accepting up to two H-bond	A	4907	8	0.82	0.69	0.26	0.38
N:=	sp2 (aromatic) nitrogen, accepting one H-bond	A	27,140	12	0.91	0.89	0.35	0.47
N::	sp2 nitrogen with two lone pairs and one double bond	A	472	4	0.89	0.59	0.23	0.12
N:#	sp nitrogen	A	15,798	10	0.72	0.66	0.29	0.32

Machine Learning and the GRID Force-Fields

More chemically aware force-field



The energy values of the isocontour surfaces chosen for H-bond donating probe (“N1,” probe) was 4.0 kcal/Mol

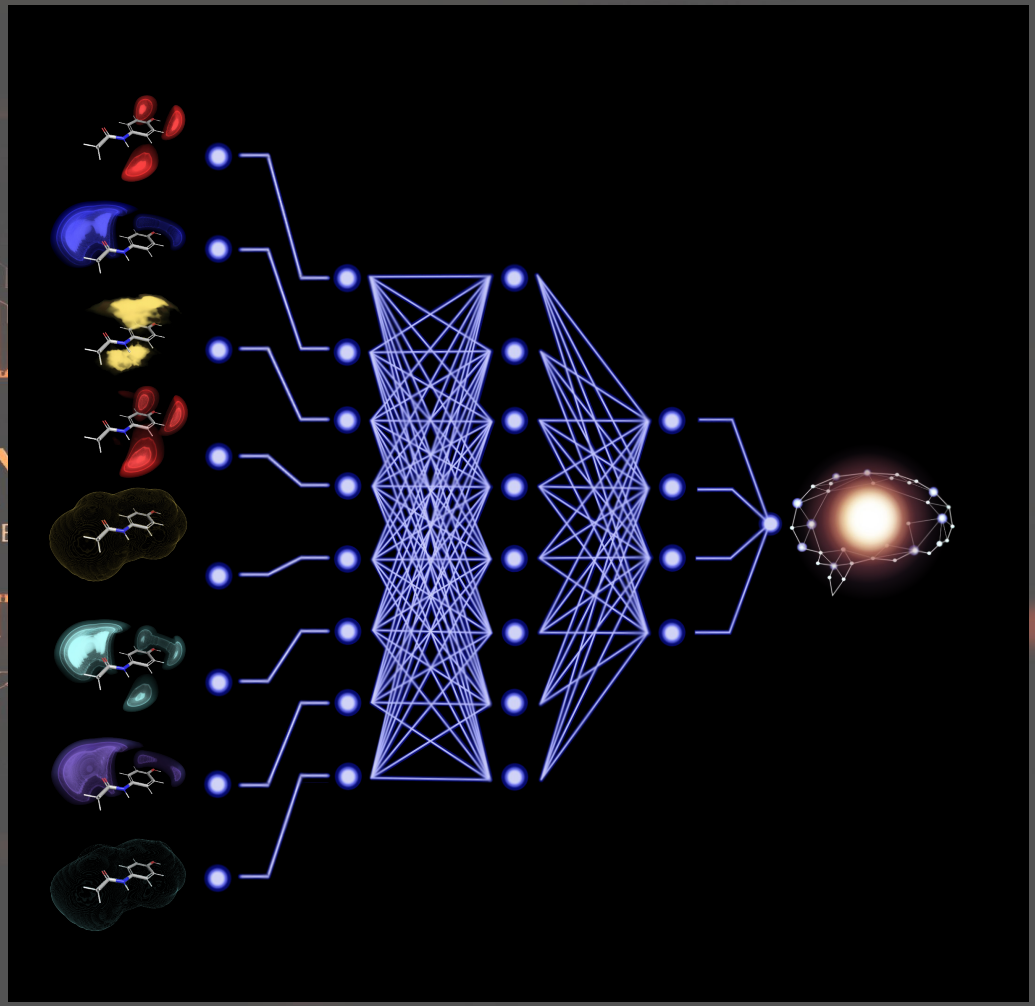
GRID MIF and PLS

DeepGRID

Two ingredients are needed:

- Deep Learning techniques (i.e., CNN)
- GRID MIFs

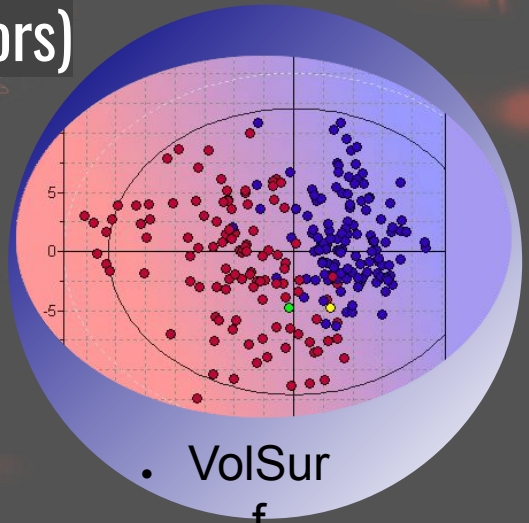
Loriano Storchi, Gabriele Cruciani, Simon Cross, "DeepGRID: Deep Learning using GRID descriptors for BBB prediction", Journal of Chemical Information and Modeling, DOI: 10.1021/acs.jcim.3c00768 (2023)



DATASET AND LABEL

Test Case: Blood Brain Barrier Permeation

- A model exists within VolSurf (PLS) – we have a baseline
- We can investigate a number of modelling approaches: DeepGRID, Random Forest & PLS (using VS descriptors)
- There are some larger publicly available datasets eg. LightBBB (7000 cpds)



Dataset Preparation

- **VS-IgBB-332** dataset In-house dataset used to build the original VolSurf model
- **Light-IgBB-416** dataset A subset of the 2105 dataset which had experimental logBB values
- **Light-BBclass-2105** dataset - Classification Generated from the Shaker/Parakkal LightBBB dataset of 7000+ structures
  - After filtering by InChI to remove duplicates 4285 compounds remained (-40%!)
  - Given that such a large proportion of the dataset contained duplicates we filtered also by Druglikeness to give 3464 compounds
  - 70% of the dataset removed due to duplicate InChI strings or diastereoisomerism

Dataset Splitting

- For each dataset, subsets of compounds were randomly selected:
  - Training Set: 60% - used to train the models
  - Validation Set: 20% - used to select the best hyperparameters or to train the CNN
  - Test Set: 20% - used as a final performance check
- The same sets were used for each model

FEATURES

DeepGRID Approach

GRAID descriptors calculated (normalised GRID MIFs, 8 channels)

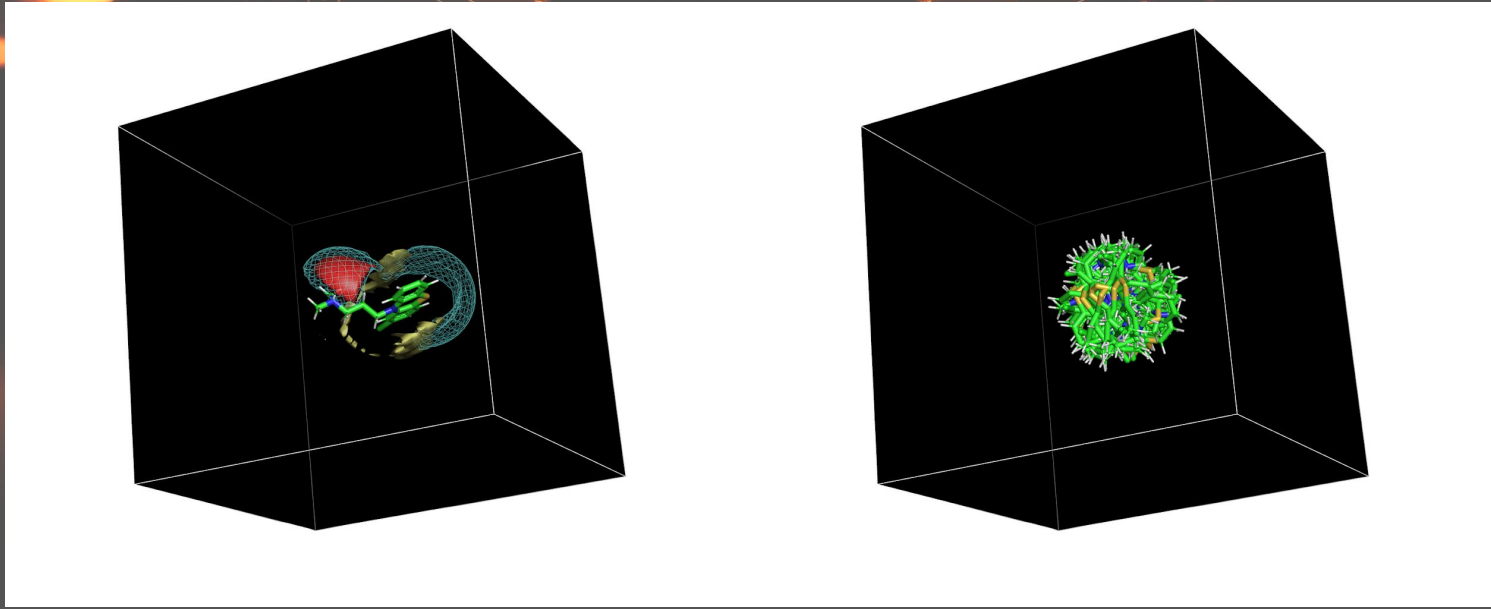
Descriptors fed into a Deep Learning CNN model

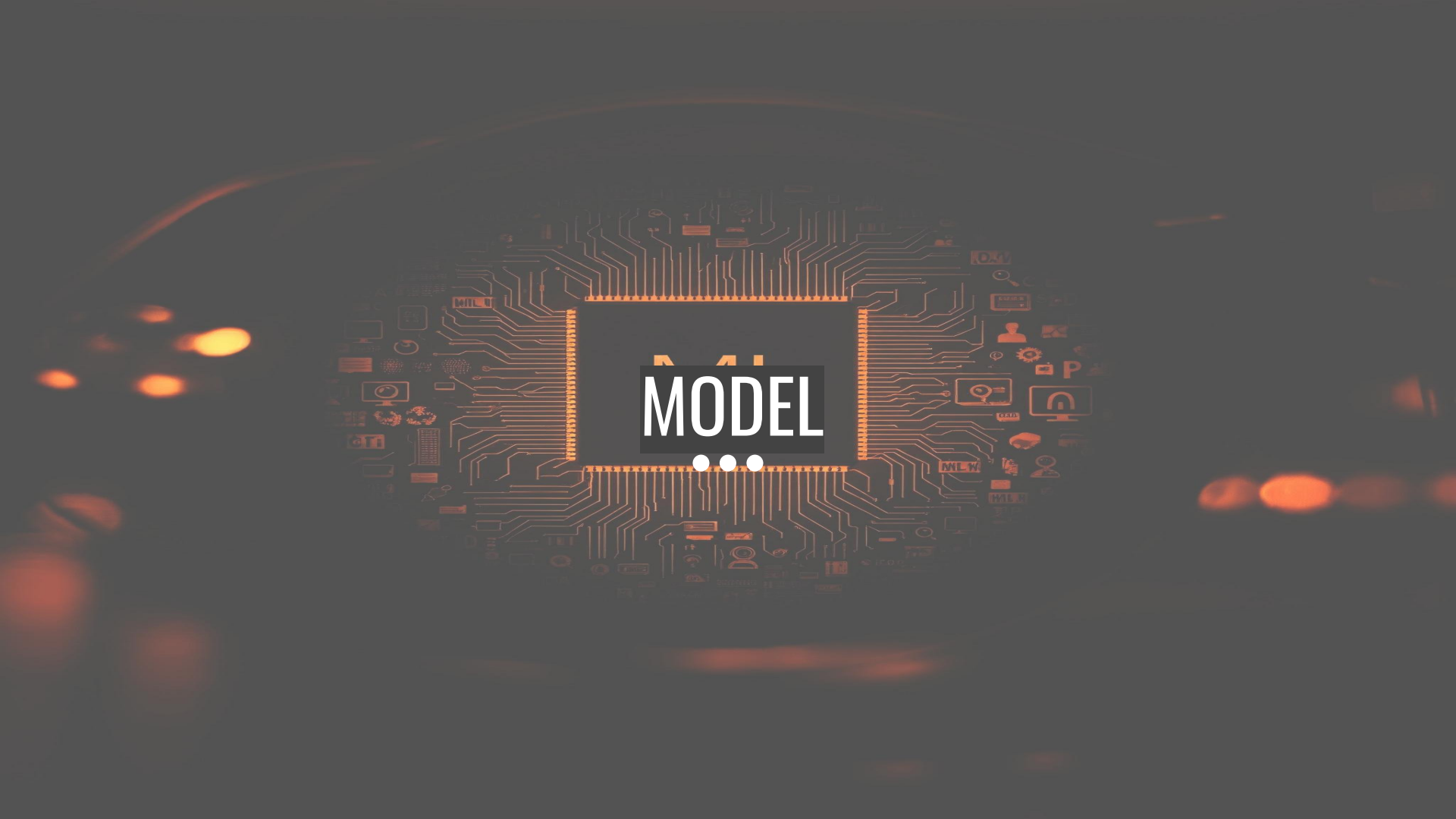
Note: in this case the training and validation sets were mixed so that different viewpoints of the same molecule were in training/validation, to allow the model to learn from the viewpoints

DeepGRID is alignment independent

Each molecule conformation centred within a grid cage 0,0,0 to 30,30,30

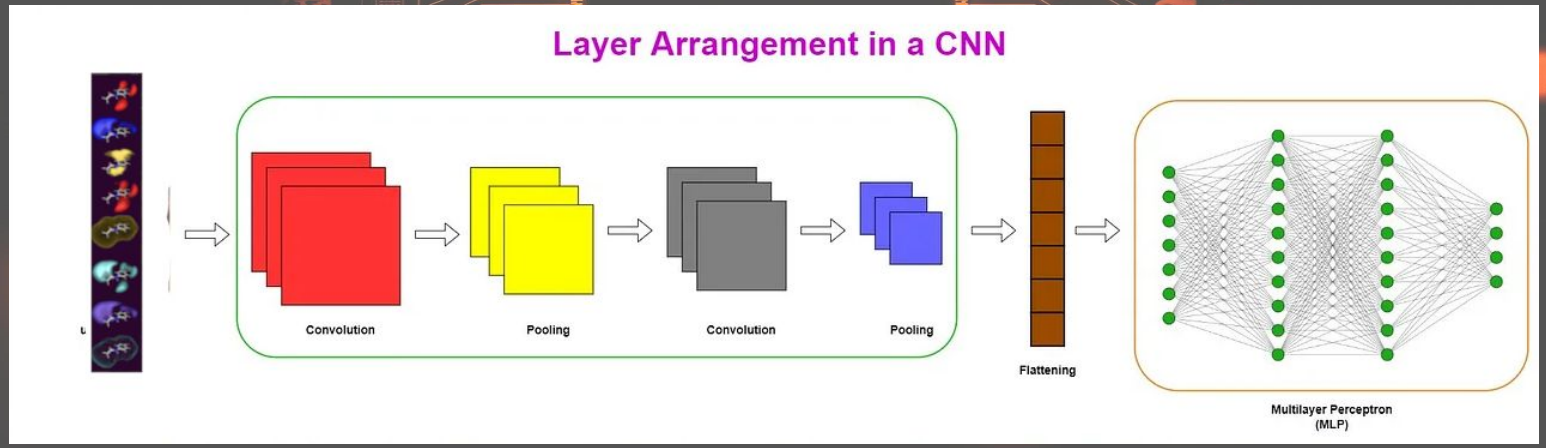
27 ‘Viewpoints’ generated by rotating the molecule around each axis

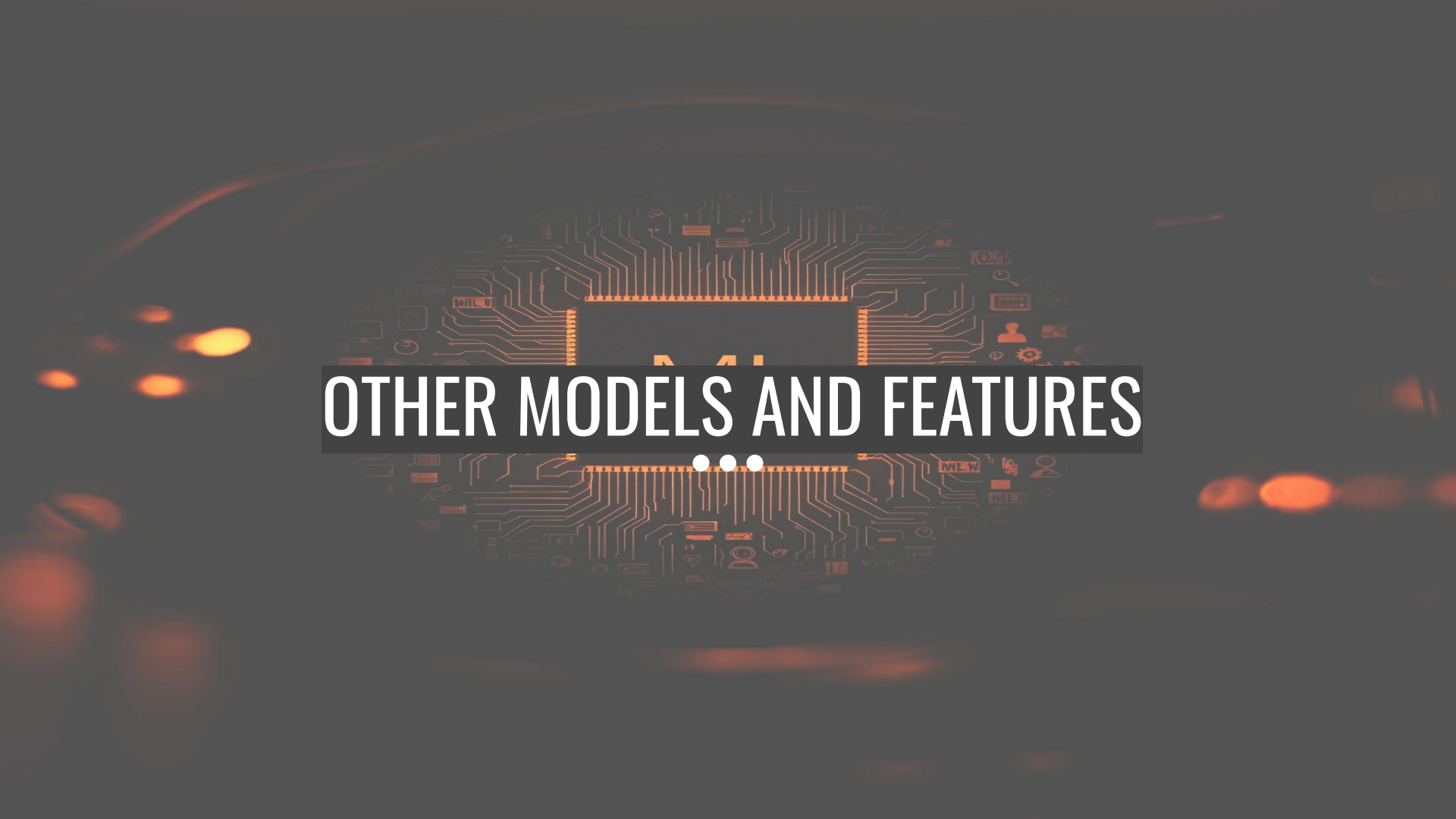



MODEL

DeepGRID Model

- 3 convolutional layers, drop out and max pooling
  - extracting features and reducing the dimensionality
- Flattening layer
- 3 dense layers and drop out before the final dense layer




OTHER MODELS AND FEATURES
...

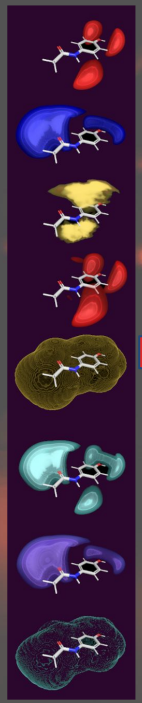
DeepGRID Hyperparameters optimization

Volsurf Descriptors

Descriptors	Probes*			Description
	OH2	DRY	O	
V	X			Molecular volume
S	X			Molecular surface
POL				Polarizability
MW				Molar mass
HB1–HB8			X	Hydrogen bonding
A				Amphiphilic moment
BV	X		X	Best volumes
W1–W8	X			Hydrophilic regions
ID1–ID8		X		Hydrophobic integy moment
Cw1–Cw8	X			Capacity factor
D1–D8		X		Hydrophobic regions
CP				Critical packing
LOG P				logarithm of partition coefficient
DIFF				Diffusivity

* Blank, other ways of calculation. For deatails, see reference Cruciani et al. (2000).

DeepGRID vs RF and PLS models



Quite some time was needed to develop the VS descriptors

Volsurf3 Descriptors

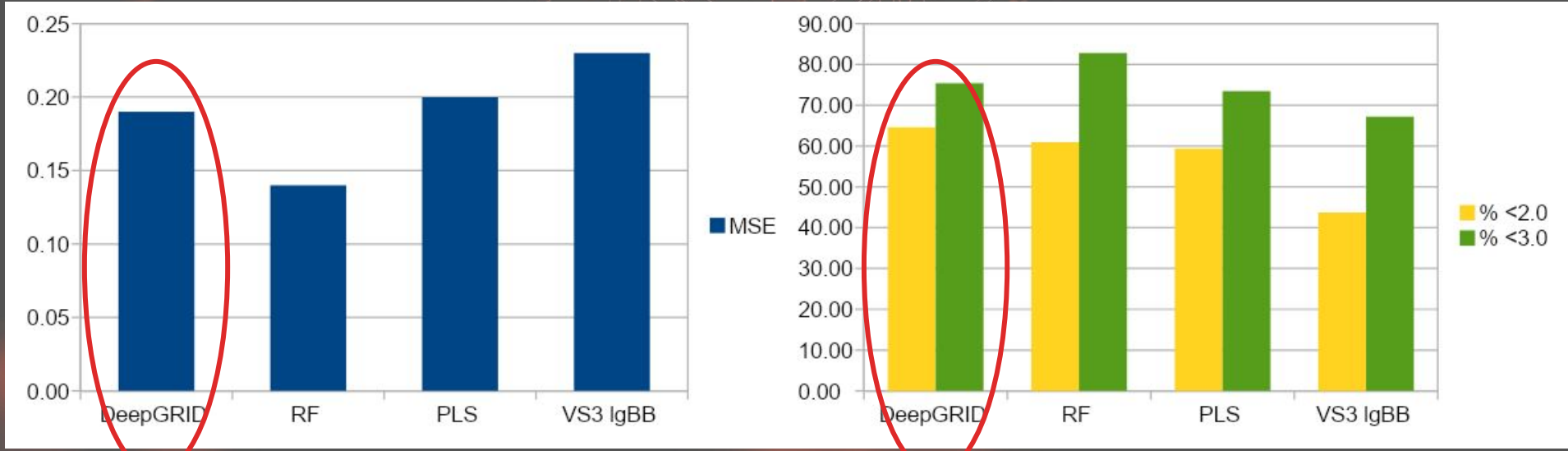
Descriptors	Probes*			Description
	OH2	DRY	O	
V	X			Molecular volume
S	X			Molecular surface
POL				Polarizability
MW				Molar mass
HB1–HB8			X	Hydrogen bonding
A				Amphiphilic moment
BV	X		X	Best volumes
W1–W8	X			Hydrophilic regions
ID1–ID8		X		Hydrophobic integy moment
Cw1–Cw8	X			Capacity factor
D1–D8		X		Hydrophobic regions
CP				Critical packing
LOG P				logarithm of partition coefficient
DIFF				Diffusivity

* Blank, other ways of calculation. For deatails, see reference Cruciani et al. (2000).

Extracted features used by the dense layers

RESULTS

VS-IgBB-332 Dataset

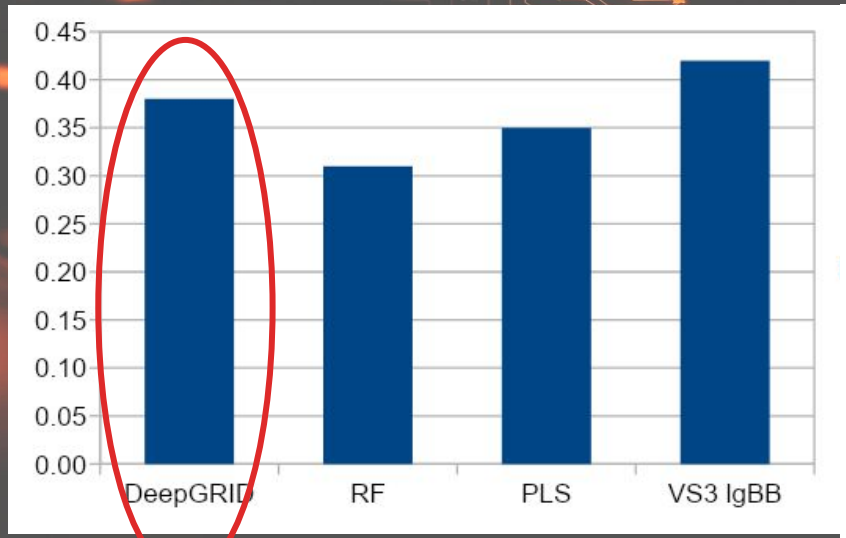


Lower is better

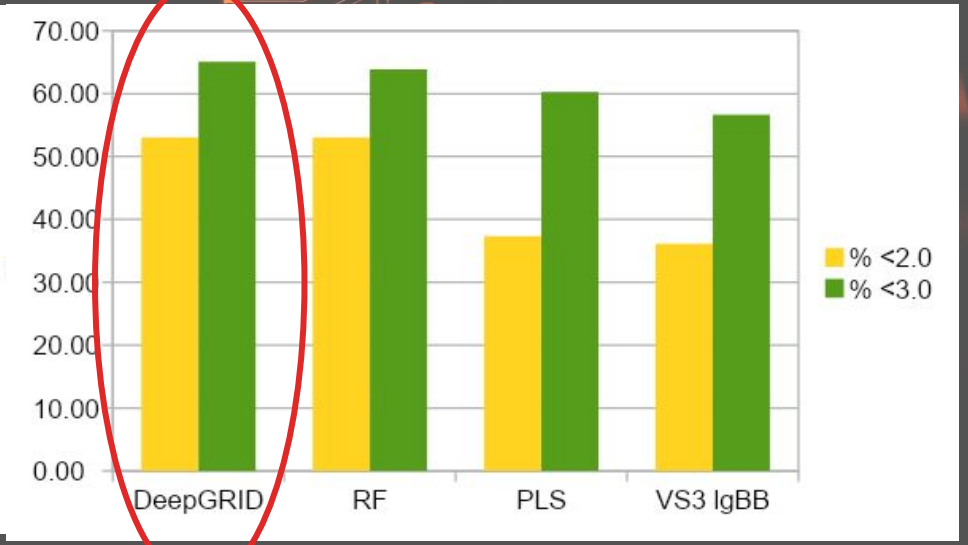
Higher is better

Light-IgBB-416 dataset is more diverse

More diverse → more difficult→ all approaches give less accurate models



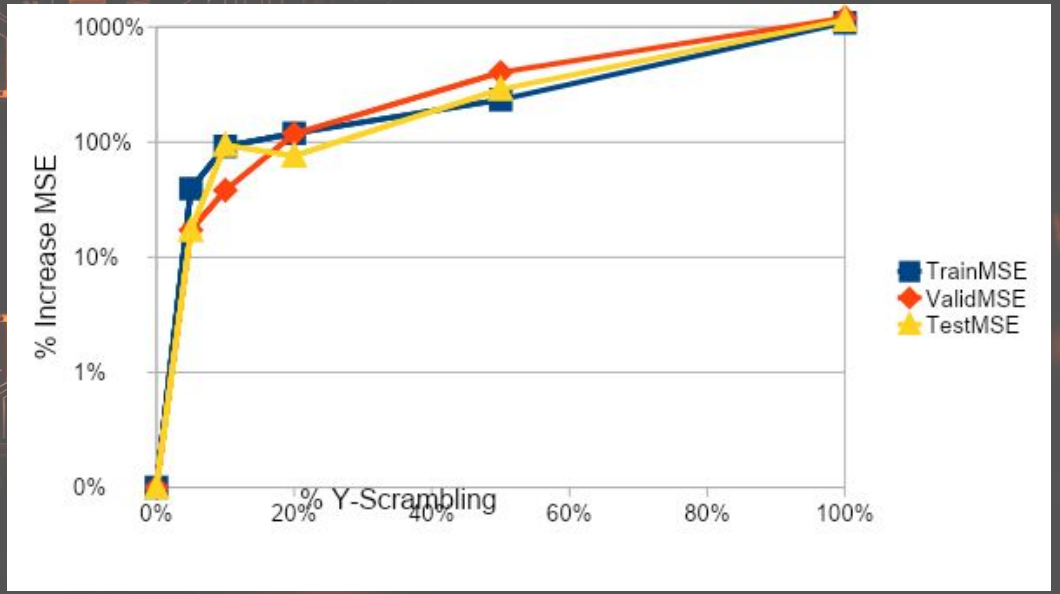
Lower is better



Higher is better

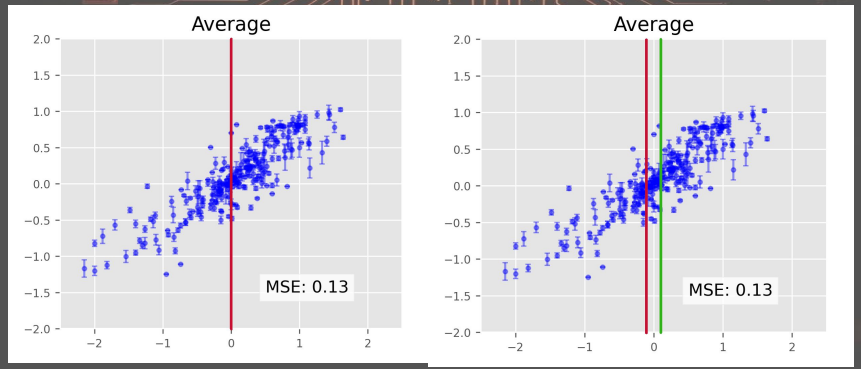
DeepGRID gives a robust model

- Y-Scrambling the data affects the model, ie. It is not overfitting
- At 5% scrambling the Test MSE is only 17% worse, hence the approach is relatively robust to erroneous data



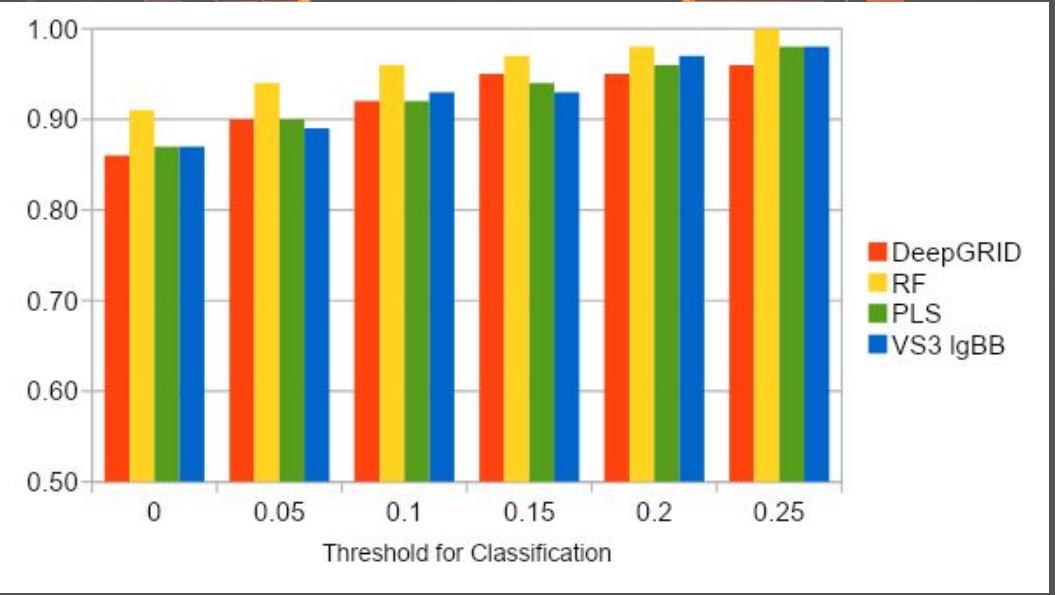
Regression → Classification

- The regression models for described can also be used for classification (BBB +/-)
- Compounds with experimental lgBB close to 0.0 may be ambiguous and misclassified
- In this case we measured the ROC AUC at varying thresholds on the **Test**



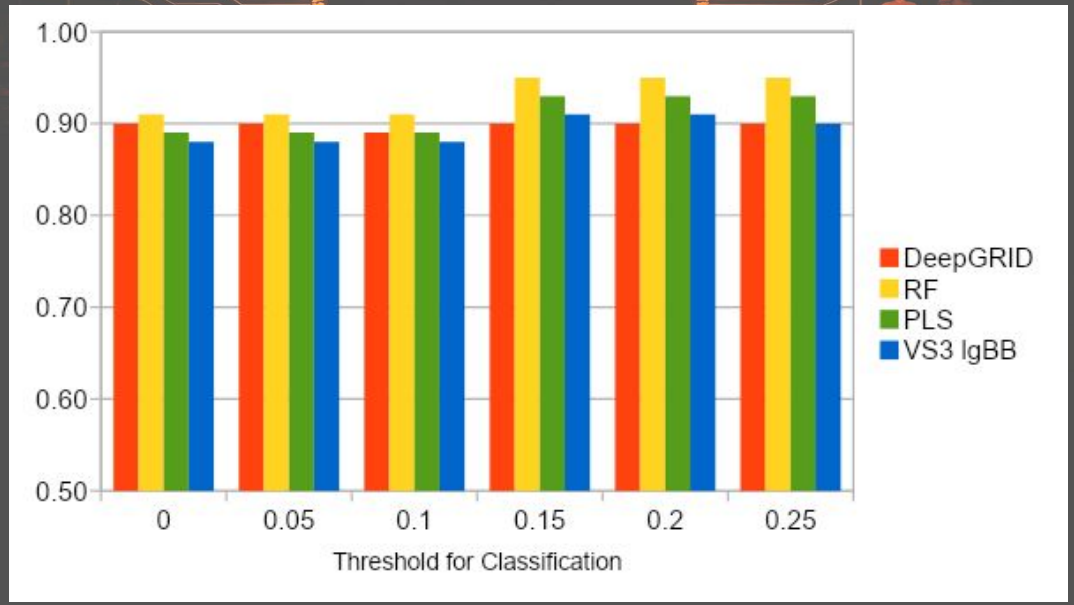
Classification: VS-IgBB-332 model

- At a minimal threshold of 0.1, all models predict with >90% accuracy
- The RF model is slightly better



Classification: Light-IgBB-416 model

- At minimal threshold of 0.1, all models predict with ~90% accuracy
- All models are fairly equal

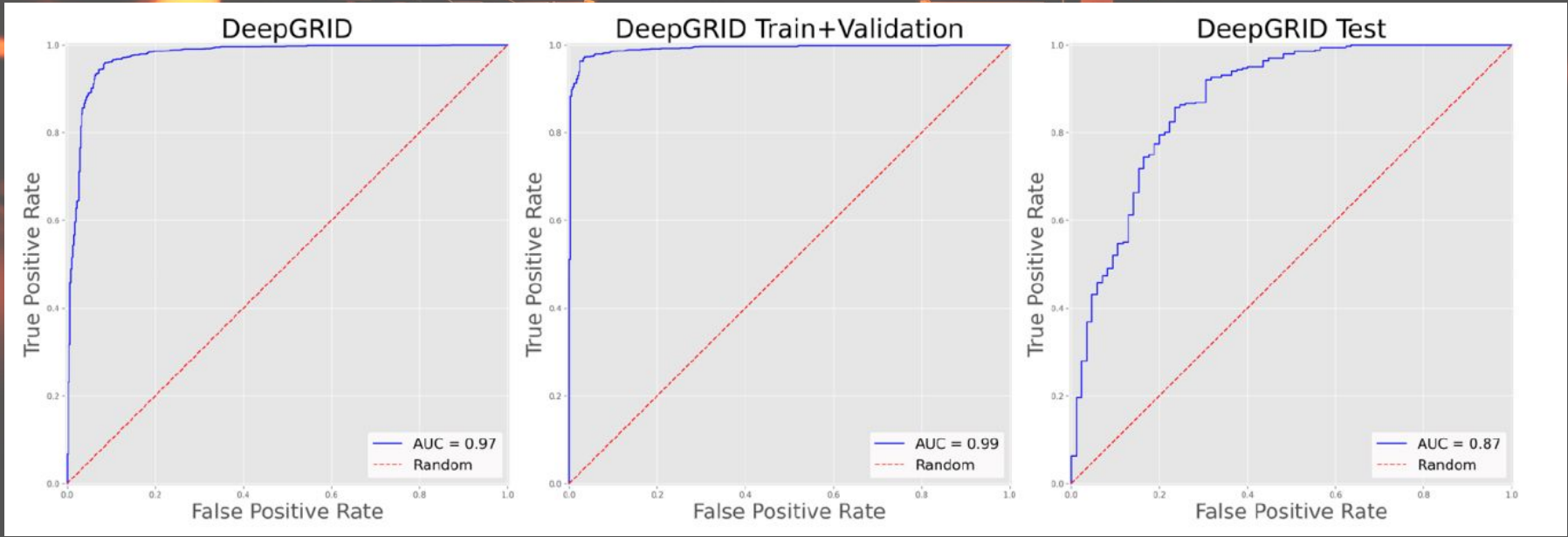


Classification Models - Light-IgBB-2105 dataset

- New classification models were built using DeepGRID and Random Forest (with hyperparameter optimization)
- Initial attempts with DeepGRID kept stalling during learning
- Potentially due to data imbalance?
- The BBB- cpds were artificially augmented to bring the balance to 0.5:1
  - successful learnin

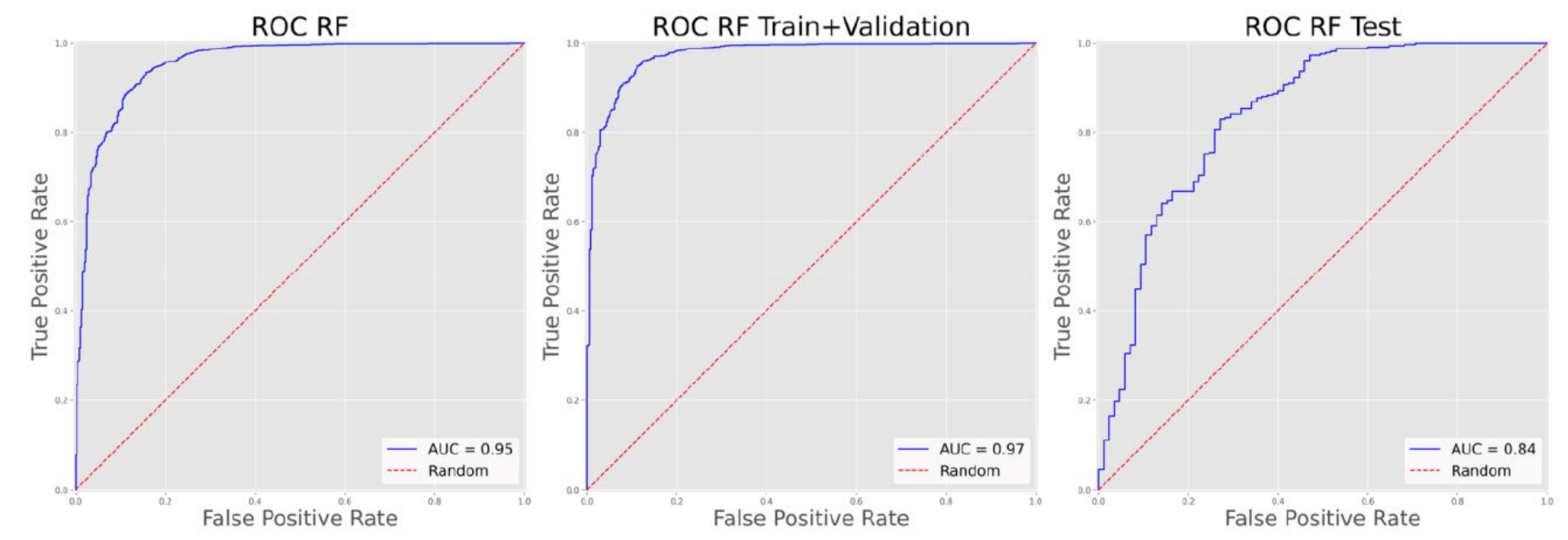
DeepGRID Classification Models - Light-IgBB-2105 dataset

AUC Full Set: **0.97** Test Set: **0.87**



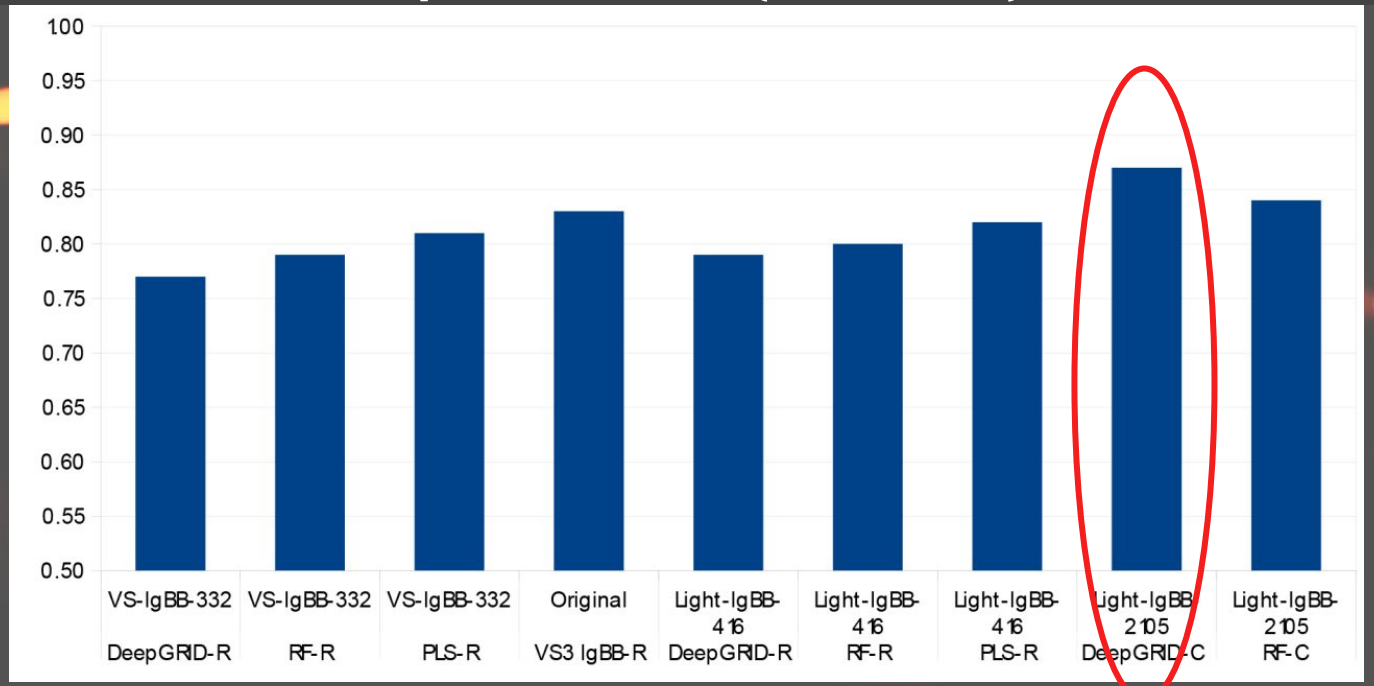
RF Classification Models - Light-IgBB-2105 dataset

AUC Full Set: **0.95** Test Set: **0.84**



DeepGRID model the best for classification

All models classification performance (ROC-AUC) on the 2105 dataset



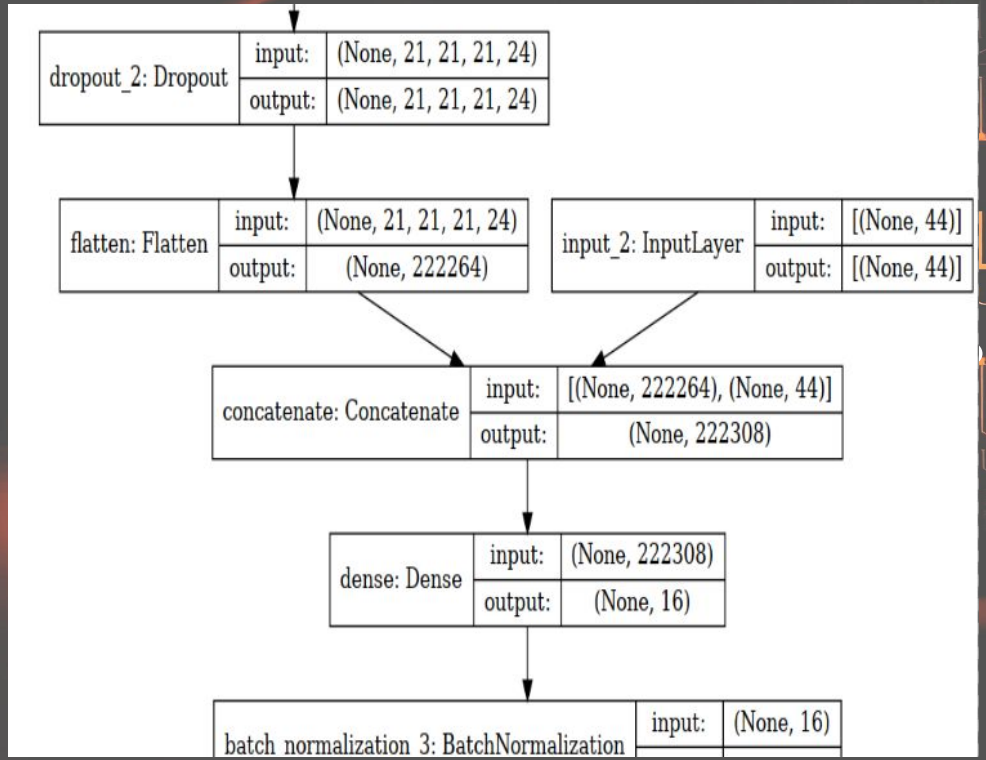
DeepGRID: mixing VS descriptors and MIF

Clearance mechanism classification fro drugs two classes :

- Metabolic Clearance: This is the most complex mechanism, involving the biotransformation of drugs into more hydrophilic metabolites to facilitate excretion.(643 compounds)
- Renal Clearance: This mechanism involves the direct excretion of drugs in the urine, typically for small, hydrophilic compounds. (329 compounds)

I am using augmentation techniques

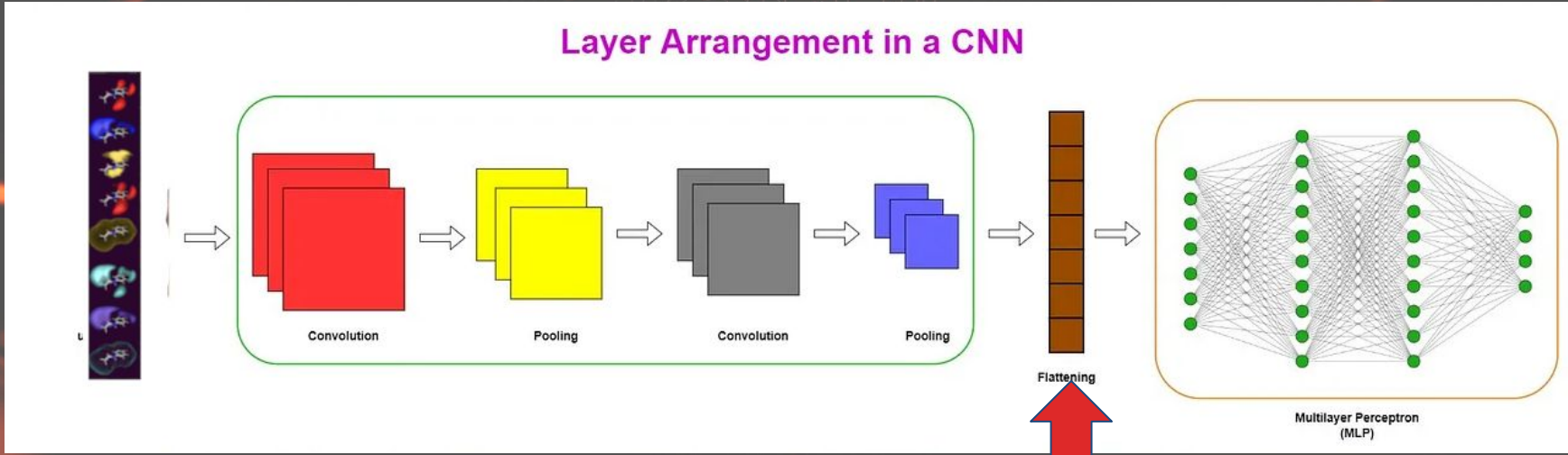
DeepGRID: mixing VS descriptors and MIF



Two model are cocananated :

- Model 1 is the CNN model
- Model 2 is a simple input layer thata is getting the VS descriptors just before the flattening layer

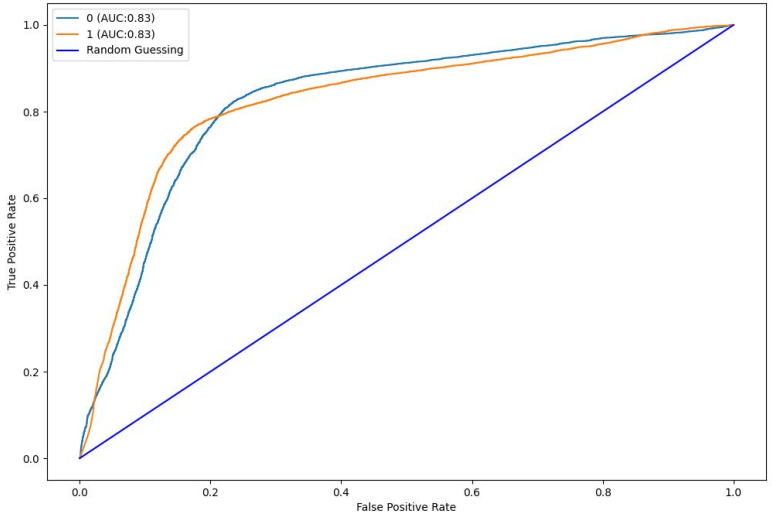
DeepGRID: mixing VS descriptors and MIF

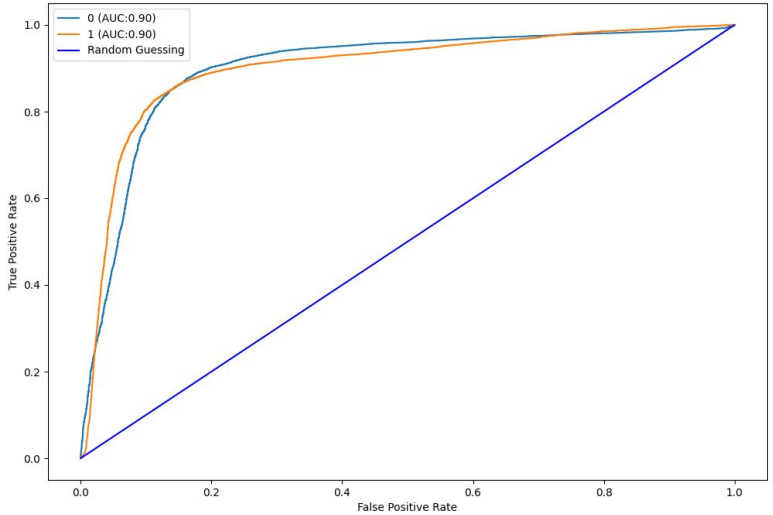


VS descriptors are appended together with the output of the Convolutional layers in the flatten layer

DeepGRID: mixing VS descriptors and MIF

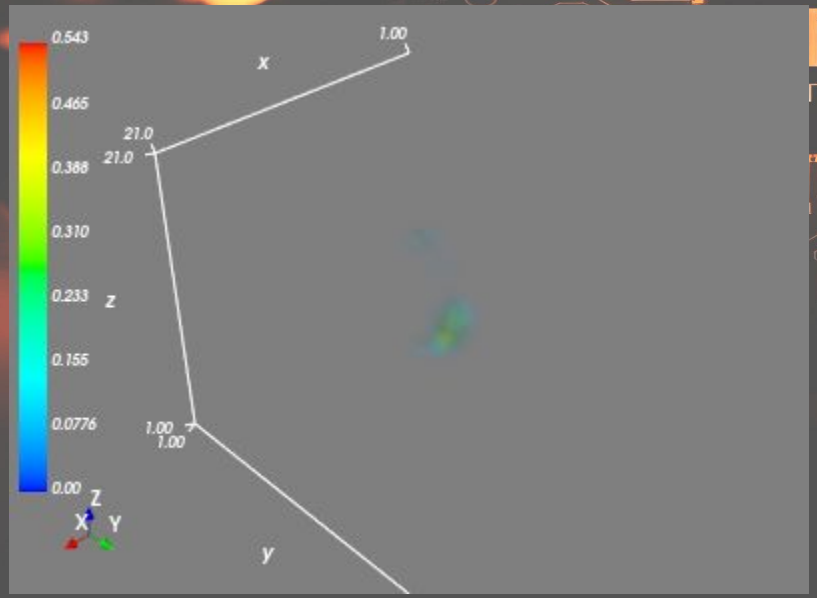
AUC Test Set without VS: **0.83 With VS : 0.90**

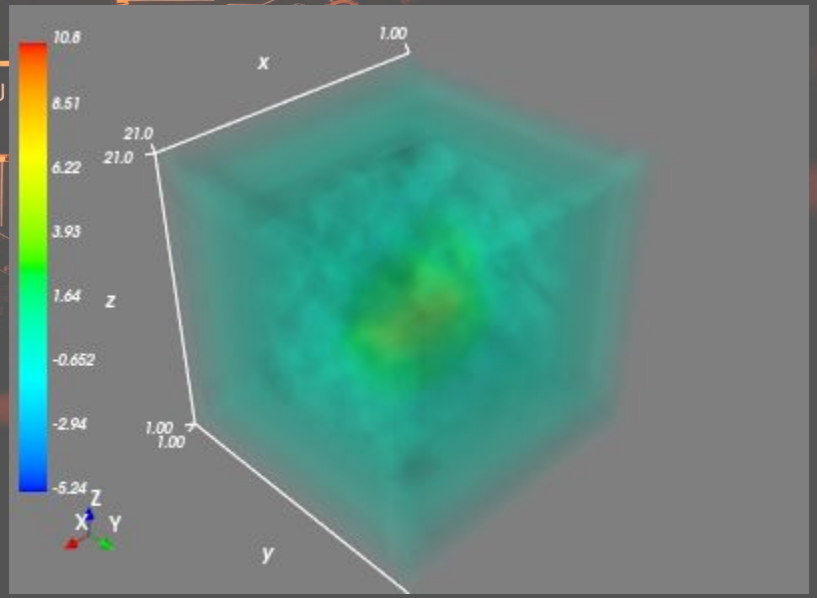


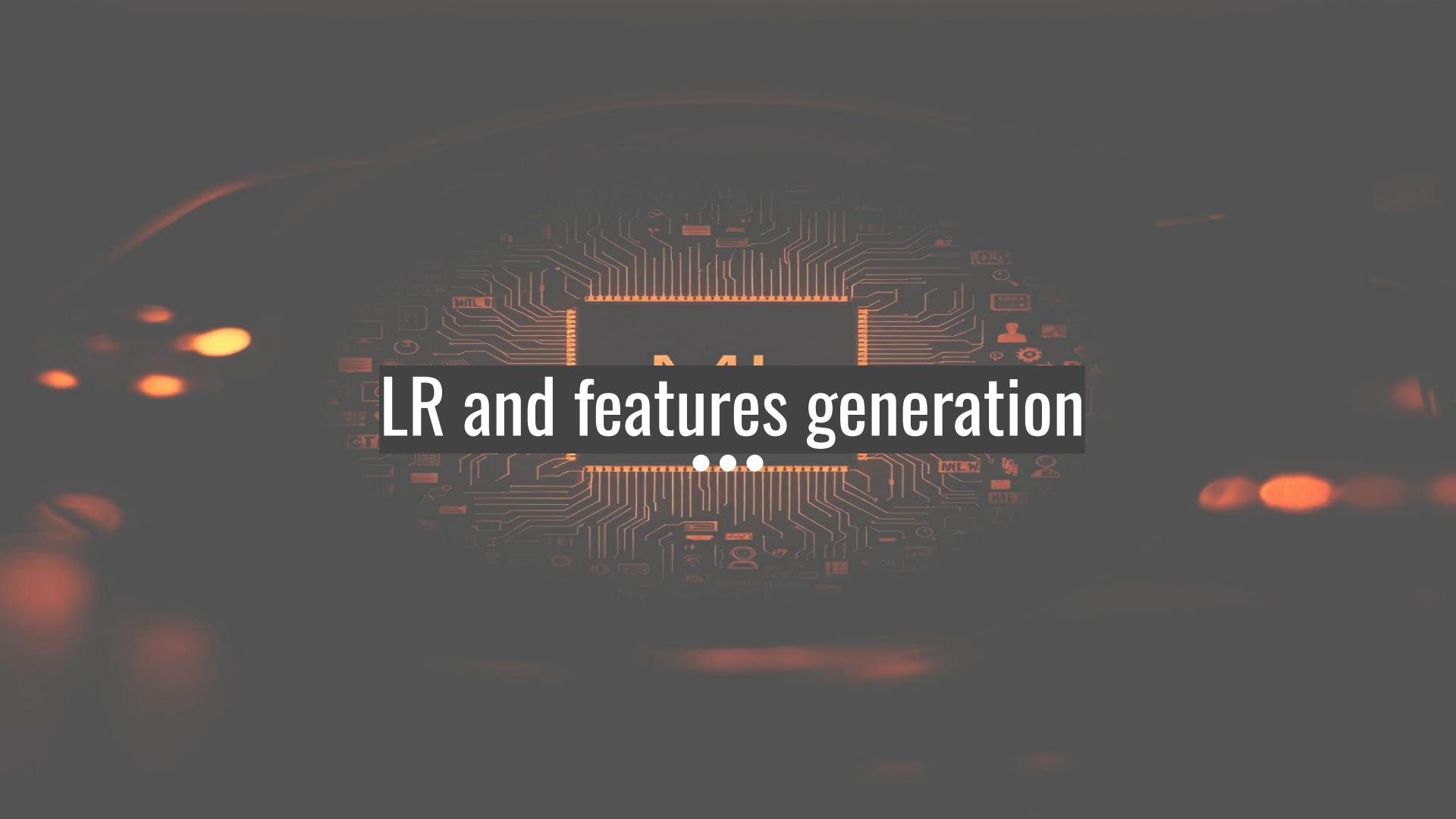


DeepGRID: try to understand how the CNN works

It is possible, although quite tricky, to dump the features as extracted by the Convolutional layers:

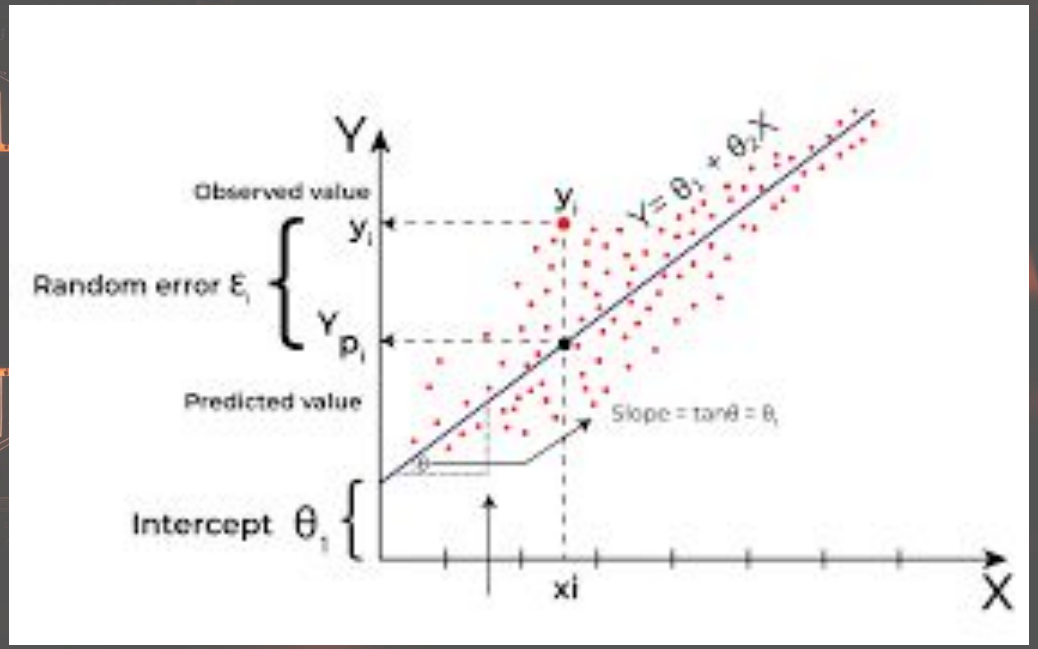





LR and features generation
...

Linear Regression

- Linear Regression models predict a dependent variable (Y) based on independent variables (X).
- The relationship between the variables is assumed to be linear.
- Models are relatively simple and easy to interpret.
- Common applications include predicting sales, energy consumption, and other continuous values.
- A key assumption is that the errors are normally distributed.



A Formula search

Methods, such as random forest (RF) or neural network (NN), are very efficient 36 but not always transparent, partially blurring the comprehension of the role played by the input variables in the final results

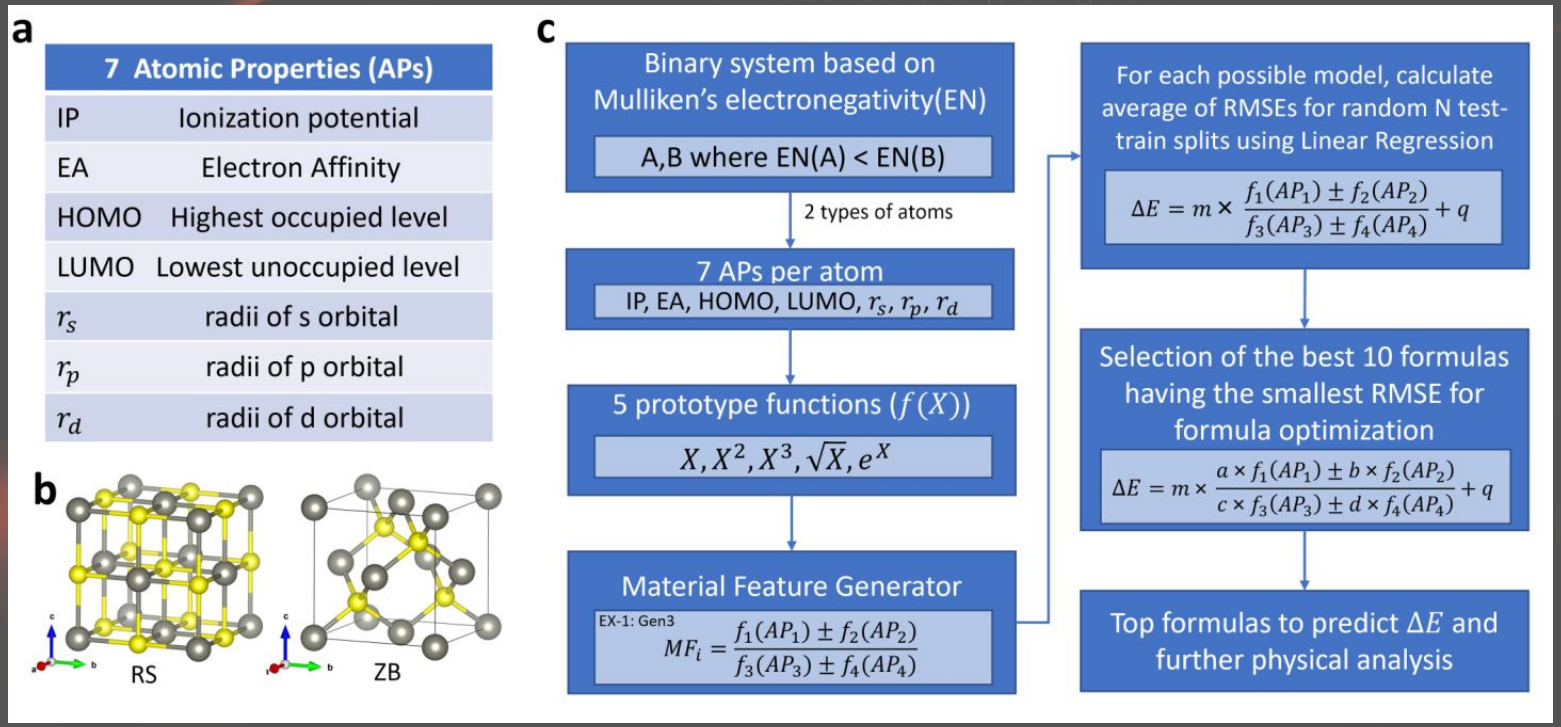
- Improvements toward the interpretability of such “black-box” ML models have been made through additional methodologies, such as model-agnostic methods (i.e., permutation feature importance)
- A ML-based approach to build sets of features (or descriptors) starting from a given set of basic variables (e.g., atomic properties), subsequently used to construct LR models (or formulas)

Inspired by the original work of Ghiringhelli et al. prediction of the difference in energy between RS [rocksalt\ and ZB; (zinc blende) from that optimization, a classification of the most stable crystal structure semiconductor AB binary compounds **(full dataset is made of 82 compounds)**

TECHOLUG

Udaykumar Gajera, Loriano Storchi, Danila Amoroso, Francesco Delodovici, Silvia Picozzi "Towards machine learning for microscopic mechanisms:a formula search for crystal structure stability based on atomic properties" Journal of Applied Physics, DOI: 10.1063/5.0088177 (2022)

A Formula search



(a) Basic atomic properties (APs) used to construct the material features. (b) Crystal structures of RS and ZB (plot made using the VESTA tool). 62 Gray (yellow) spheres represent A (B) atoms. (c) Workflow for formula construction, machine-learning methodology, validation, and MF selection.

A Formula search

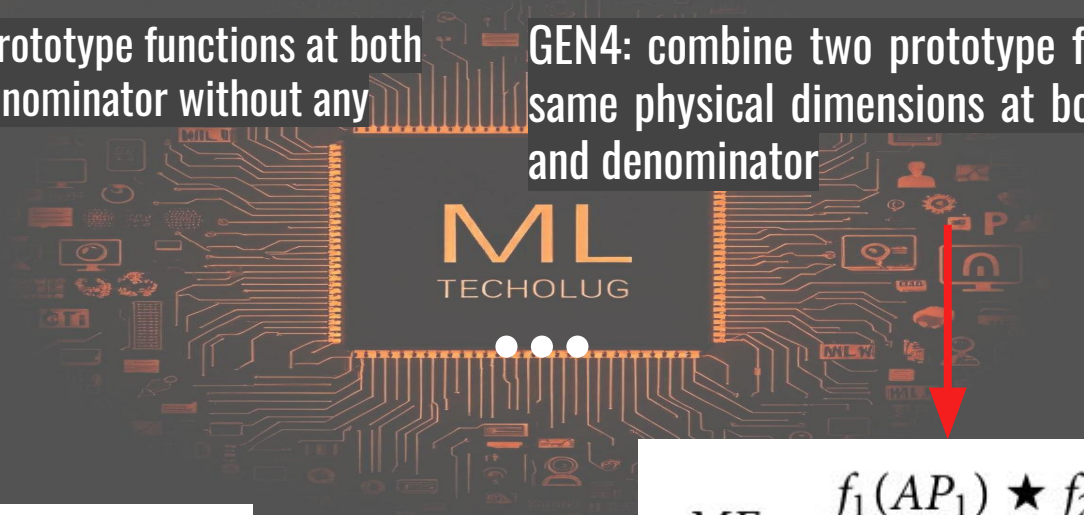
GEN1: combine two prototype functions in the numerator, forcing them to belong to the same kind of APs, which is both “spatial”-like or both “energy”-like; one prototype function is at the denominator with the only constraint to be non-zero

GEN2: combine two prototype functions with the same kind of APs at the numerator and a single prototype function at the denominator with an argument of a different kind with respect o the numerator ones. For instance, if AP_1 in $f_1(AP_1)$ and AP_2 in $f_2(AP_2)$ are “energy” terms (i.e., EA or HOMO), then AP_3 must be a “spatial” term (i.e., r_p)

$$MF = \frac{f_1(AP_1) \pm f_2(AP_2)}{f_3(AP_3)}.$$

A Formula search

GEN3: combine two prototype functions at both the numerator and denominator without any constraints,



$$MF = \frac{f_1(AP_1) \pm f_2(AP_2)}{f_3(AP_3) \pm f_4(AP_4)}.$$

GEN4: combine two prototype functions with the same physical dimensions at both the numerator and denominator

$$MF = \frac{f_1(AP_1) \bigstar f_2(AP_2)}{f_3(AP_3) \bigstar f_4(AP_4)},$$

$$\bigstar = + - \times \div.$$

A Formula search

$$\Delta E = m \times \frac{a \times f_1(AP_1) \bigstar b \times f_2(AP_2)}{c \times f_3(AP_3) \bigstar d \times f_4(AP_4)} + q,$$

GRID search, for each set of weight coefficients generated during the grid search, we also run the linear regression. Thus, we are performing a proper formula optimization, as at each step of the grid search, we are updating both the weight coefficients as well as the slope and intercept coming from the LR

Formula	avg (RMSE)	RMSE	R^2	Success rate (%)	Generator type
$0.127 \times \frac{0.800 \times EA(B) - 1.000 \times IP(B)}{1.110 \times r_p(A)^2} - 0.352$	0.1457	0.1419	0.89	89	1D descriptor[55]
$-1.870 \times \frac{0.801 \times \sqrt{r_p(B)} - 0.606 \times \exp[r_p(A)]}{1.010 \times r_p(A)^3} - 0.968$	0.1191	0.1143	0.93	91	*GEN*1
$0.477 \times \frac{0.876 \times \sqrt{\lvert HOMO(B)\rvert} + 0.468 \times \sqrt{\lvert LUMO(B)\rvert}}{1.110 \times r_p(A)^2} - 0.372$	0.1340	0.1296	0.91	91	*GEN*2
$1.609 \times \frac{0.642 \times r_p(B) + 0.502 \times \sqrt{\lvert r_d(A)\rvert}}{1.170 \times r_p(A)^3 + 1.170 \times r_p(B)^3} - 0.309$	0.0991	0.0961	0.95	94	*GEN*3
$1.207 \times \frac{0.878 \times r_s(B) + 0.200 \times r_p(A)}{0.512 \times r_p(B)^3 + 0.610 \times r_p(A)^3} - 0.359$	0.1045	0.1016	0.94	99	*GEN*4

1D formulas after the optimization step, along with related statistics. Notation as in Table I. RMSEs are in eV.

A Formula search

Generator	Total Number of generated formulas	Elapsed time (s) for 1D formula construction	Elapsed time (s) for formula optimization
GEN1	106400	5117.32	180.84
GEN2	67840	3338.93	181.54
GEN3	1091200	51821.54	420.52
GEN4	278106	13237.39	418.62

Time needed to generate the best 1D formula and perform its optimization. All the calculations have been performed in a PC equipped with an Intel Core i5-8500 processor and 16 GiB of RAM.

A Scoring Function

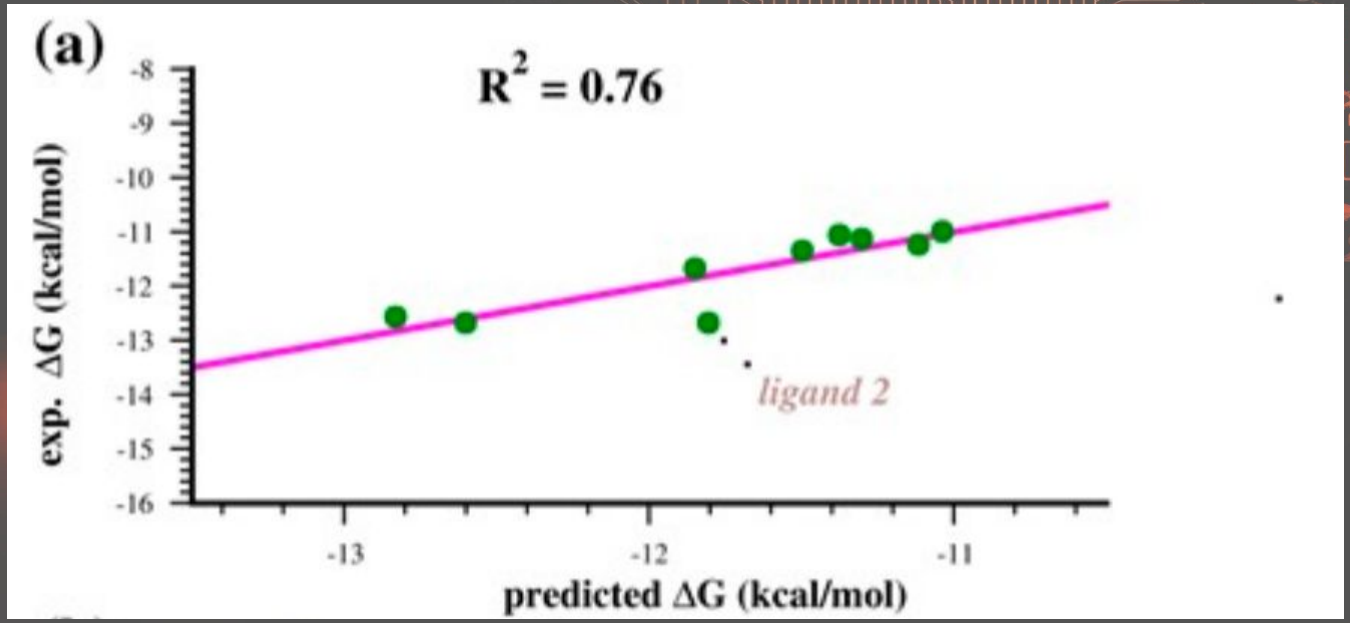
Predicting ligand-metalloenzyme binding affinity, focusing on human Carbonic Anhydrase II (hCA II) inhibitors. It combines fragment molecular orbital (FMO) and GRID approaches,

- FMO Calculations: FMO2 calculations were performed on reduced ligand-receptor complexes to assess binding energies and pair interaction energies.
- GRID Calculations: GRID was used to calculate hydrophobic interaction fields and quantify hydrophobic interactions.
- Dataset: A set of benzenesulfonamide ligands of hCA II was selected as a case study.

Roberto Paciotti, Nazzareno Re,Loriano Storchi, "Combining the Fragment Molecular Orbital and GRID Approaches for the Prediction of Ligandâ€“Metalloenzyme Binding Affinity: The Case Study of hCA II Inhibitors", Molecules, DOI: 10.3390/molecules29153600 (2024)

A Scoring Function

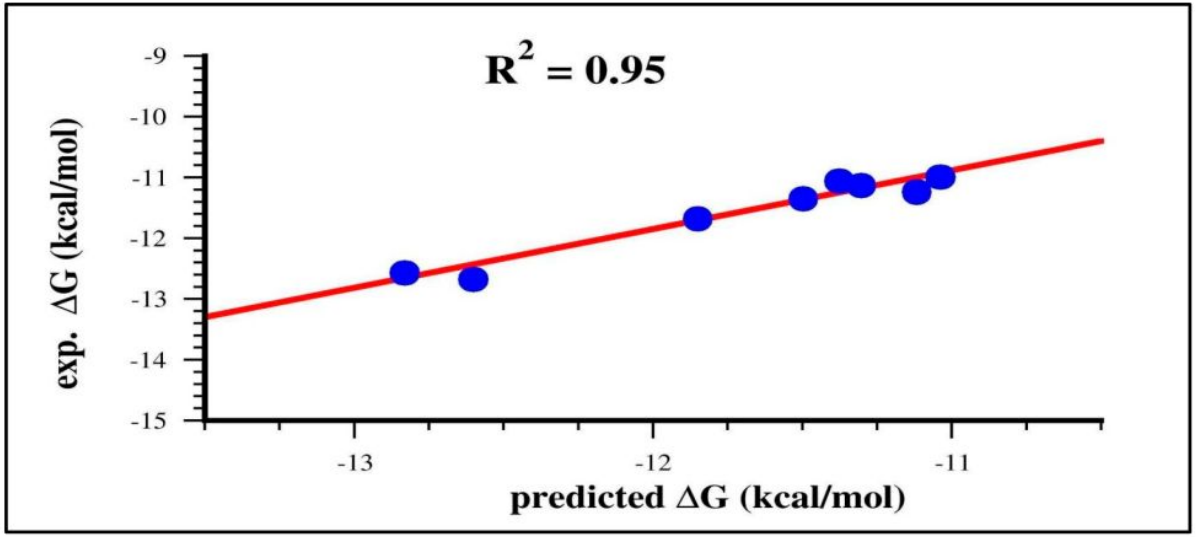
$$\Delta G = -7.4\{[0.7(\log P)^3 - 0.5(e^{\text{HIE-E}})]/[0.5(\text{F2LE})^3 - 0.4(\text{HIE-E})^5]\} - 13$$



A portion of the Ligand 2 structure connected to the benzenesulfonamide is polar compared to other ligands, which determines, in principle, a better interaction with water molecules. Thus, we hypothesize that the its binding pose in the experimental conditions assumed in the measurement of the Ki could be influenced by surrounding water molecules and be slightly different from that observed in the crystal structure.

A Scoring Function

$$\Delta G = -7.4\{[0.7(\log P)^3 - 0.5(e^{\text{HIE-E}})]/[0.5(\text{F2LE})^3 - 0.4(\text{HIE-E})^5]\} - 13$$



To improve the binding affinity of the benzenesulfonamide there should be a certain balance between electrostatic and hydrophobic interactions in order to minimize the denominator and maximize maximize the binding affinity

Interpretable ML

Random Forrest and Permutation Feature Importance

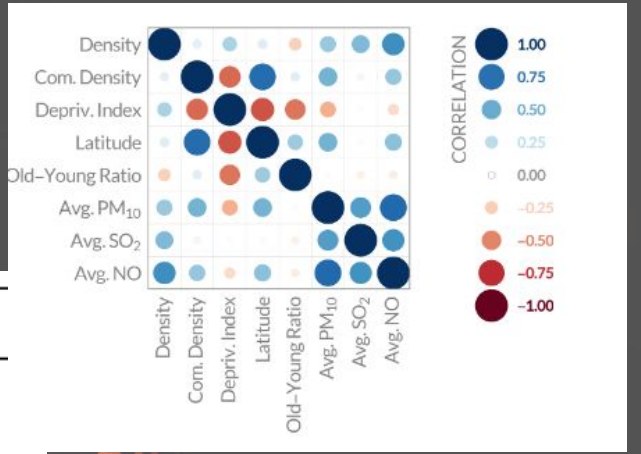
Use the RF model not for prediction purpose but to detect how much a feature is important respect to the others. Two ingredients:

- The permutation feature importance is defined to be **the decrease in a model score when a single feature value is randomly shuffled**. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature
- Random forests or random decision forests is **an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time**

Leonardo Aragao, Elisabetta Ronchieri, Giuseppe Ambrosio5, Diego Ciangottini, Sara Cutini, Cristina Duma, Pasquale Lubrano, Barbara Martelli, Davide Salomoni, Giusy Sergi, Daniele Spiga, Fabrizio Stracci, Loriano Storchi "Air quality changes during the COVID-19 pandemic guided by robust virus-spreading data in Italy",to Air Quality, Atmosphere & Health, DOI: 10.1007/s11869-023-01495-x (2024)

Features

Feature name	Description
Population Density	Population divided by province's area.
Commuting Density	Percentage of commuters over population [8].
Deprivation Index	Represents the multidimensionality of the social and material deprivation concept [29] (calculated for the year 2012).
Latitude	North–south geographic coordinate regarding the province's capital.
Old-Young Ratio	Number of individuals aged 20 or less over the ones aged 65 and over.
Avg. PM_{10}	Average concentration of PM_{10} during the whole study period.
Avg. NO	Average concentration of NO during the whole study period.
Avg. SO_2	Average concentration of SO_2 during the whole study period.

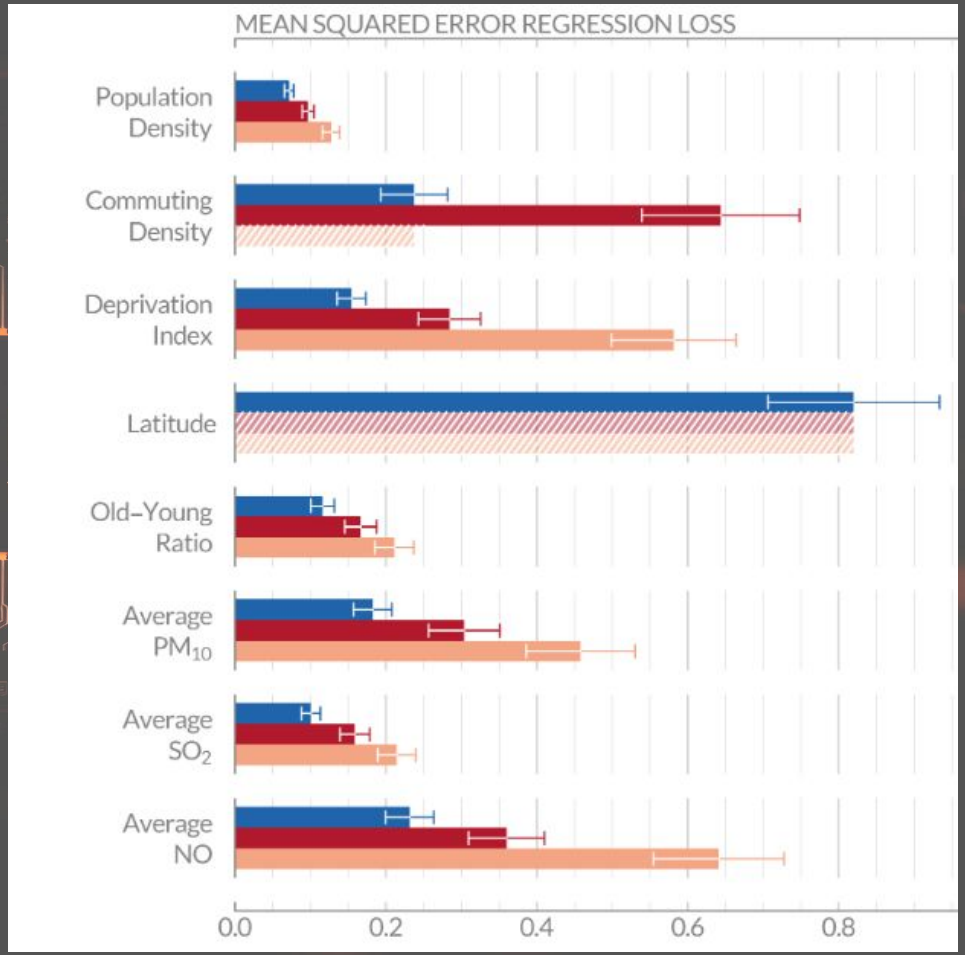


Results

104 Italian provinces analysed applying the Permutation Feature Importance Analysis to a set of different Random Forest models

The role of the pollutants seems not the most important

Details	RMSE	R^2
All features	0.320	0.950
Latitude Removed	0.341	0.943
Latitude and Comm. Density removed	0.362	0.936



EXERCISE 1

Exercise 1

Starting from Ex1 code:

1. Add the calculation of the MSE for the test set snd the scatter plot predicted VS true values. Similarly add the calculation of the accuracy and of the confusion matrix for the test set in the case of the classification model
   a. extract also the features importance from the RF model
2. What happen if we remove the highly correlated fatures ?
3. What happen if we usa the principal components instead of the raw features ? (90% or 80% of the variance)

EXERCISE 2

Exercise 2

Starting from Ex2 code:

1. Try to improve the NN model , how to cure overfitting ?
    a. Data Strategies
        i. More Data: The most effective solution! More data provides a more representative sample and reduces the chance of learning noise.
        ii. Data Augmentation: Artificially increase your dataset size by creating variations of existing samples. For images, this could include rotations, flips, crops, etc. For text, you might use synonyms or paraphrasing.
        iii. Feature Selection: Carefully choose relevant features. Remove redundant or irrelevant ones that might contribute to overfitting.

Exercise 2

Starting from Ex2 code:

1. Try to improve the NN model , how to cure overfitting ? (try to use categorical)
2. Architectural Changes
    a. Simpler Model: Reduce the complexity of your network. Try fewer layers, fewer neurons per layer, or a less complex architecture.
    b. Dropout: Randomly drop neurons during training. This forces the network to learn more robust features and prevents reliance on any single neuron.
    c. Regularization: Add penalty terms to your loss function that discourage large weights. Common types include L1 and L2 regularization. L2 regularization: Adds a penalty proportional to the square of the weights. These penalties encourage the network to keep the weights small, effectively shrinking them towards zero. This leads to a simpler model that is less likely to overfit.

Exercise 2

Starting from Ex2 code:

1. Try to improve the NN model , how to cure overfitting ?
2. Training Process
    a. Early Stopping: Monitor your model's performance on a validation set during training. Stop training when validation performance starts to degrade.
    b. Reduce Learning Rate: A smaller learning rate allows the model to make finer adjustments to the weights and avoid "jumping around" in the loss landscape.