**Import**

import pandas as pd

import numpy as np

import scipy as sp

import cv2

import os

import itertools

import zipfile

import shutil

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import tensorflow as tf

import sklearn

from sklearn import datasets

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn import linear\_model

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVC

from sklearn.svm import SVR

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Conv2D, Activation, Flatten, MaxPool2D

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

**Skleran**

* *.fit(x,y)*
* *.score(x\_test, y\_test)*
* *.predict(value)*

**Tensorflow**

* *model.compile(optimizer, loss, metrics)*
* *model.fit(x,y,epoch….)*
* *model.fit\_generator(train\_generator, steps\_per\_epoch, epochs, validation\_data, validation\_steps, verbose)*
* *model.evaluate(X\_test, Y\_test)*
* *model.predict()*

**Load**

from sklearn import datasets

# load iris dataset

iris = datasets.load\_iris()

x=iris.data

y=iris.target

* ***dir(iris\_data)*** → ['DESCR', 'data', 'feature\_names', 'filename', 'target', 'target\_names']
* ***print(iris\_data['DESCR'])*** → DESCR is a description of the dataset
* ***print(iris\_data['filename'])*** → filename is the name of the source file where the data is stored.
* ***print(iris\_data['feature\_names'])*** → feature\_names is the name of the feature columns.['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
* print(iris\_data['target\_names']) → target\_names, despite the name, is not the names of the target columns. There is only one column of targets.
* print(iris\_data['target']) → target and see that it contains zeros, ones, and twos. These correspond to the target names 'setosa', 'versicolor', and 'virginica'.

**TFDS**

import tensorflow as tf

import tensorflow\_datasets as tfds

# EXTRACT

dataset = tfds.load(name="mnist", split="train")

# TRANSFORM

dataset.shuffle(100)

# # TRANSFORM

# dataset = dataset.shuffle(NUM\_SAMPLES) # buffer size

# dataset = dataset.repeat(NUM\_EPOCHS)

# dataset = dataset.map(lambda x: ...)

# dataset = dataset.batch(BATCH\_SIZE)

# LOAD

for data in dataset.take(1):

image = data["image"].numpy().squeeze()

label = data["label"].numpy()

print("Label: {}".format(label))

plt.imshow(image, cmap=plt.cm.binary)

plt.show()

import tensorflow as tf

import tensorflow\_datasets as tfds

# Construct a tf.data.Dataset from MNIST

dataset = tfds.load(name="mnist")

# Inspecting shapes and datatypes

print(dataset)

# Checking if the dataset is an instance of tf.data.Dataset

print(isinstance(dataset['train'], tf.data.Dataset))

# See available datasets

print(tfds.list\_builders())

mnist, info = tfds.load(name="mnist", with\_info=True)

info

# print('URLs: ', info.urls)

print('Image features: ', info.features['image'])

print('Label features: ', info.features['label'])

print('Number of training examples ', info.splits['train'].num\_examples)

print('Number of training examples ', info.splits['test'].num\_examples)

Image features: Image(shape=(28, 28, 1), dtype=tf.uint8)

Label features: ClassLabel(shape=(), dtype=tf.int64, num\_classes=10)

Number of training examples 60000

Number of training examples 10000

# as\_supervised=True : A handy feature of loading a dataset is

# to specify it a supervised. If you do this

# with as supervised equals true, then your dataset

# will be preformatted into tuples of data and label

# as you can see here.

dataset = tfds.load('mnist', as\_supervised=True)

# Inspecting shapes of batch

for image, label in dataset['train'].take(1):

print(image.shape, label.shape)

(training\_images, training\_labels), (test\_images, test\_labels) = tfds.as\_numpy(tfds.load('fashion\_mnist',

split = ['train', 'test'],

batch\_size=-1,

as\_supervised=True))

import tensorflow as tf

import tensorflow\_datasets as tfds

data = tfds.load('horses\_or\_humans', split='train', as\_supervised=True)

val\_data = tfds.load('horses\_or\_humans', split='test', as\_supervised=True)

train\_batches = data.shuffle(100).batch(32)

validation\_batches = val\_data.batch(32)

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(16, (3,3), activation='relu',

input\_shape=(300, 300, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dense(1, activation='sigmoid')

])

model.compile(optimizer='Adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(train\_batches, epochs=10, validation\_data=validation\_batches, validation\_steps=1)

import tensorflow as tf

import tensorflow\_datasets as tfds

def my\_one\_hot(feature, label):

return feature, tf.one\_hot(label, depth=3)

data = tfds.load('rock\_paper\_scissors', split='train', as\_supervised=True)

val\_data = tfds.load('rock\_paper\_scissors', split='test', as\_supervised=True)

data = data.map(my\_one\_hot)

val\_data = val\_data.map(my\_one\_hot)

train\_batches = data.shuffle(100).batch(10)

validation\_batches = val\_data.batch(32)

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input\_shape=(300, 300, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dense(3, activation='softmax')

])

model.summary()

model.compile(loss = 'categorical\_crossentropy', optimizer='Adam', metrics=['accuracy'])

history = model.fit(train\_batches, epochs=10, validation\_data=validation\_batches, validation\_steps=1)

model.save("test2.h5")

**Keras Datasets**

import tensorflow as tf

data = tf.keras.datasets.fashion\_mnist

(training\_images, training\_labels), (test\_images, test\_labels) = data.load\_data()

import tensorflow as tf

class myCallback(tf.keras.callbacks.Callback):

def on\_epoch\_end(self, epoch, logs={}):

if(logs.get('accuracy')>0.6):

print("\nReached 60% accuracy so cancelling training!")

self.model.stop\_training = True

mnist = tf.keras.datasets.fashion\_mnist

(x\_train, y\_train),(x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

callbacks = myCallback()

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(512, activation=tf.nn.relu),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

model.compile(optimizer=tf.optimizers.Adam(),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=10, callbacks=[callbacks])

**Describe**

* read\_csv()
* .read\_csv(url, sep=",")
* Head()
* tile()
* Info()
* Dtypes
* Describe()
* s.Unique()
* df.unique()
* np.unique()
* .Shape, .shape[0], .shape[1]
* df.target.value\_counts()
* len()
* Df\_banking.columns
* Df\_banking.index
* series.value\_counts(dropna=False)

**Convert data**

* pd.to\_datetime(df\_employee['tanggal\_lahir'])
* df.astype(np.float32)

**Null Values**

* Isnull()
* Isna()
* notna()
* Dropna()
* Fillna()
* isnull().any()

**Category**

* Label Encoder
* One Hot Encoding
* df['gender'].replace(['Female', 'Male'], [0,1], inplace=True)
* # pandas.get\_dummies(data, prefix=None, prefix\_sep='\_', dummy\_na=False, columns=None, sparse=False, drop\_first=False, dtype=None)
* pd.get\_dummies(data)
* pd.get\_dummies(data[“”])
* dataset = pd.get\_dummies(dataset, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
* c = pd.get\_dummies(df['slope'], prefix = "slope")
* df.name[df.name == 'orange'] = 0

df.name[df.name == 'grapefruit'] = 1

* tf.one\_hot(label, depth=3)

**Standarisasi**

* StandardScaler (scaler.fit(X), scaled\_data = scaler.transform(X), mx.fit\_transform(df\_boston[['lstat', 'rm', 'ptratio', 'indus', 'tax', 'rad']]))
* MinMaxScaler

**Ubah Column**

* Df.columns = list
* df.rename(columns=dict)

**Drop Column**

* # DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')
* iris.drop('Id',axis=1,inplace=True)
* # df.drop(labels, axis, index, columns, inplace)
* data = df.drop(columns=['User ID'])
* x\_data = df.drop(['target'], axis = 1)
* df = df.drop(columns = ['cp', 'thal', 'slope'])
* X = df.drop(['CustomerID', 'gender'], axis=1)

**Concat**

* df = pd.concat([“”, “”], axis = 1)
* pd.concat([df, df/s], axis=1)
* df+df

**Train test Split**

* x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2)
* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33, random\_state = 0)
* train\_data, test\_data = train\_test\_split(data, test\_size=0.2, random\_state=12)
* = train\_test\_split(news, label, test\_size=0.2 ,random\_state=42, shuffle = True)

**Catatan:**

Input shape : brp dimensi yang dimasukkan (Baris )

array([0.22222222, 0.625 , 0.06779661, 0.04166667],

Input shape : [1] untuk satu row

array([[0.22222222, 0.625 , 0.06779661, 0.04166667],

[0.16666667, 0.41666667, 0.06779661, 0.04166667],

[0.11111111, 0.5 , 0.05084746, 0.04166667],

[0.08333333, 0.45833333, 0.08474576, 0.04166667],

[0.19444444, 0.66666667, 0.06779661, 0.04166667],

Input shape : (4,) untuk satu row