Lec 1

# Numpy:

NumPy supports a wide range of functions that allow us to perform operations on arrays in a highly efficient way。

## 1.Construct a dataframe(table):

import numpy as np
data =
{'year':[2018,2019,2020,2018,2109,2020,2018,2019,2020],
'team':['HKU','HKU','HKU','CU','CU','CU','UST','UST','UST'],
'wins':[30,28,32,29,32,26,21,17,19],
'draw':[6,7,4,5,4,7,8,10,8],
'losses':[2,3,2,4,2,5,9,11,11]}
football = pd.DataFrame(data)
Football

| | year | team | wins | draws | losses |
|---|---|---|---|---|---|
| 0 | 2018 | HKU | 30 | 6 | 2 |
| 1 | 2019 | HKU | 28 | 7 | 3 |
| 2 | 2020 | HKU | 32 | 4 | 2 |
| 3 | 2018 | CU | 29 | 5 | 4 |
| 4 | 2019 | CU | 32 | 4 | 2 |
| 5 | 2020 | CU | 26 | 7 | 5 |
| 6 | 2018 | UST | 21 | 8 | 9 |
| 7 | 2019 | UST | 17 | 10 | 11 |
| 8 | 2020 | UST | 19 | 8 | 11 |

## 2.construct a 1-D array:

lista =[1,2,3]
print(f'list a = {lista}.')
arra = np.array(lista)
print(f'array a = {arra}')
type(arra), arra.dtype, arra.ndim, arra.size, arra.shape

```
list a = [1, 2, 3].
array a = [1 2 3]

(numpy.ndarray, dtype('int64'), 1, 3, (3,))
```

**Note: Shape of an array. The 1-D array [1,2,3] has three rows, and its shape is (3,) (I know, its a little bit odd not to say it has three columns, but we want our definition to be consistent with higher dimensional array.**

## 3. An example on constructing a 2-D array

listb =
np.array([[1.1,1.2],[2.1,2.2],[3.1,3.2]])
print(f'list b = {listb}.')
arrb = np.array(listb)
print(f'array b = {arrb}')
type(arrb), arrb.dtype, arrb.ndim, arrb.size, arrb.shape

```
list b = [[1.1 1.2]
 [2.1 2.2]
 [3.1 3.2]].
array b = [[1.1 1.2]
 [2.1 2.2]
 [3.1 3.2]]

(numpy.ndarray, dtype('float64'), 2, 6, (3, 2))
```

## 4. Various ways to initialize an numpy array:

np.zeros((2,3))
np.ones((4,2))
**Note: You can remove the parentheses of the tuples**
a=(2,3)　a=2,3 is the same
np.arange(0,10,2)　不包括**end,** 间隔**2**
np.linspace(0,10,10)　总共**10**个数
np.random.random((4,2))　在[0,1)中

## 5. Reshape

b = a.reshape((3,4))　a保持不变
**Warning: the method "reshape" will only construct and return another array, and the original array (i.e., the argument) will be kept intact.**

## 6. Element-wise operations

a = [1,2,3]
b = [10,10,10]
a+b `[1, 2, 3, 10, 10, 10]`
a = np.array([1,2,3])
b = np.array([10,10,10])
a+b `array([11, 12, 13])`
在np上的操作都是元素级别的，还可以进行+、sqrt、sin、* 等操作
**Just like lists, we can access elements of an array using the "indexing" operator and the "slicing" operator**
A[2][3]=A[2,3]
A[0:2, 2:4]

**Conditions and Boolean Arrays**

cond = A % 2 == 0

```
array([[ True, False,  True, False,  True, False],
       [ True, False,  True, False,  True, False],
       [ True, False,  True, False,  True, False]])
```

**Positive and negative indices:**
p=n+len(a),p从0开始

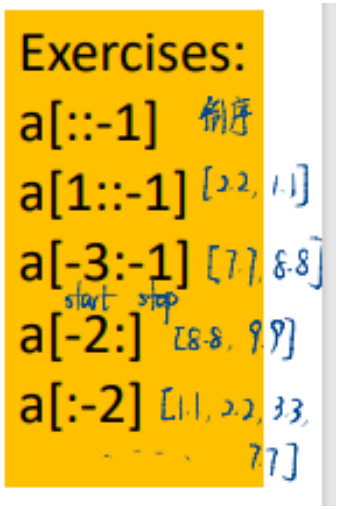
**Default values in a slice a[start:stop:step] or a[start:stop]:**

- With one colon(:) in bracket
  - s[start:stop] # items from start through stop-1
  - s[start:] # items from start through the rest of the string
  - s[:stop] # items from the beginning through stop-1
  - s[:] # a copy of the whole string
- Default values:
  - start = 0
  - stop = len(s)

With two colons(::) in bracket
- s[start:stop:step]

Default values:
- step = 1
- when step > 0, start = 0, stop = len(s)
- when step < 0, start = -1, stop = - len(s) - 1

| 1.1 | 2.2 | 3.3 | 4.4 | 5.5 | 6.6 | 7.7 | 8.8 | 9.9 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |




# Pandas:

Pandas supports two major data structures:
• Series: Similar to Numpy's 1D-array • Dataframes: A 2D table

import pandas as pd

## 1.Series

s = pd.Series([12, -4, 7, 9])

```
0    12
1    -4
2     7
3     9
dtype: int64
```

s = pd.Series([12, -4, 7, 9],
index=['Tom','Peter','Mary','Zoe'])

```
Tom      12
Peter    -4
Mary      7
Zoe       9
dtype: int64
```

**Construct the same series based on a dictionary:**

tdict = {'Tom':12, 'Peter':-4, "Mary":7, "Zoe":9}

```
(dict, {'Tom': 12, 'Peter': -4, 'Mary': 7, 'Zoe': 9})
```

t = pd.Series(tdict)

```
Tom      12
Peter    -4
Mary      7
Zoe       9
dtype: int64
```

**Two ways to access the elements of a Series:**

**(1) using the index as if it is a numpy array**

t[2], t[[1,2]]

```
(7,
 Peter   -4
 Mary     7
 dtype: int64)
```

t[0:2] 同样不含end

```
Tom      12
Peter    -4
dtype: int64
```

**(2) using the index labels**

t['Tom':'Zoe']

```
Tom      12
Peter    -4
Mary      7
Zoe       9
dtype: int64
```

t[['Mary','Peter']]

```
Mary      7
Peter    -4
dtype: int64
```

**Element-wise operations, conditions and Boolean array selection can be applied as if it is a numpy**

t + 2, np.square(t):

```
(Tom      14
 Peter    -2
 Mary      9
 Zoe      11
 dtype: int64, Tom      144
 Peter     16
 Mary      49
 Zoe       81
 dtype: int64)
```

t > 0, t[t>0]:

```
Tom       True
Peter    False
Mary      True
Zoe       True
dtype: bool
```

```
Tom      12
Mary      7
Zoe       9
dtype: int64
```

s = pd.Series([1,0,2,1,2,3])

```
0    1
1    0
2    2
3    1
4    2
5    3
dtype: int64
```

s.unique(), s.sum(), s.mean(), s.max(), s.min()

```
(array([1, 0, 2, 3]), 9, 1.5, 3, 0)
```

s.value_counts()

```
1    2
2    2
0    1
3    1
dtype: int64
```

## 2. Dataframe:

```
data = {'color':['blue','green','yellow','red','white'],
        'object': ['ball','pen','pencil','paper','mug'],
        'price': [1.2,1.0,0.6,0.9,1.7]}
frame = pd.DataFrame(data)
frame
```

data = {'color':['blue','green','yellow','red','white'], 'object': ['ball','pen','pencil','paper','mug'], 'price': [1.2,1.0,0.6,0.9,1.7]}

frame = pd.DataFrame(data)

frame

| | color | object | price |
|---|---|---|---|
| 0 | blue | ball | 1.2 |
| 1 | green | pen | 1.0 |
| 2 | yellow | pencil | 0.6 |
| 3 | red | paper | 0.9 |
| 4 | white | mug | 1.7 |

**Smaller frame by selecting some columns**:

sf = pd.DataFrame(data, columns = ['object','price'])

**Give labels to the indexes**:

d2 = pd.DataFrame(data, index=['one','two','three','four','five'])

**Creating a dataframe from a numpy array:**

```
d3 = pd.DataFrame(np.arange(16).reshape((4,4)),
                  index=['red','blue','yellow','white'],
                  columns=['ball','pen','pencil','paper'])
d3
```

| | ball | pen | pencil | paper |
|---|---|---|---|---|
| red | 0 | 1 | 2 | 3 |
| blue | 4 | 5 | 6 | 7 |
| yellow | 8 | 9 | 10 | 11 |
| white | 12 | 13 | 14 | 15 |

**Some useful functions**:

d3.columns

Index(['ball', 'pen', 'pencil', 'paper'], dtype='object')

d3.index

Index(['red', 'blue', 'yellow', 'white'], dtype='object')

d3.values

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

d3.sum(), d3.mean(), d3.max(), d3.min()均是在列内做操作

d3.sort_values(by='paper',ascending=False) 按某属性降序

d3.iloc[[2,0]] 先第2行再0行,按**index**

d3.loc[['blue','white']] 按 **label**

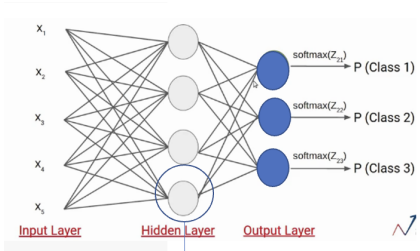
## Reading/Writing different types of files:

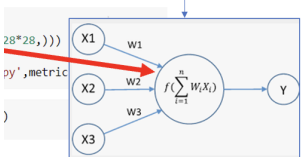
```
from google.colab import files
d3.to_csv('test.csv')
files.download('test.csv')

uploaded = files.upload()
fd = pd.read_csv("df3.csv",index_col=0)
```

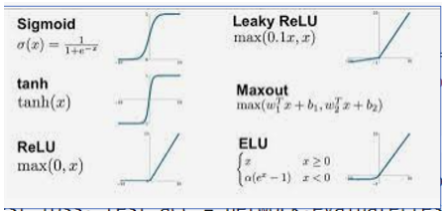
**Lec2:**

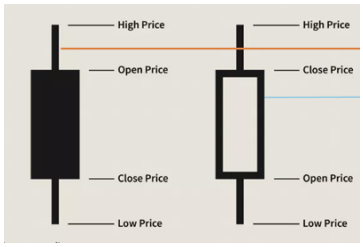
**ML Basics:**

| Unsupervised Learning | | Supervised Learning | | Reinforcement Learning | |
|---|---|---|---|---|---|
| Clustering | Consumer segmentation | Classification | Fraud detection | - | Real-time decisions |
| | Recommendation System | | Image classification | | Robot Navigation |
| Density Estimation | - | Regression | Price prediction | | Skill acquisition |
| | | | Weather forecast | | Game AI |



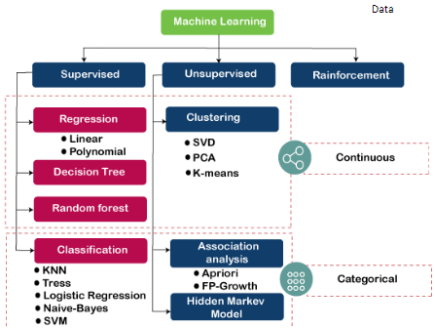




## 7 Steps of ML:

1. Gathering Data: Quality (balanced data) and Quantity of data that being gathered will directly determine how good the predictive model can be.
2. Data Preparation: Data cleaning; transforming the data (feature engineering); Randomize the order; Split data: training vs testing
3. Choosing ML algorithm




4. Training
5.Evaluation
6.Parameter tuning
7.Predicting unseen data

Linear Regression model:预测冰淇淋销售

```
[1]  import matplotlib.pyplot as plt
     import numpy as np
     from sklearn.linear_model import LinearRegression

     x = np.array([10,12,16,20,24,26]).reshape((-1,1))
     y = np.array([4,5,7,10,15, 16])

[3]  model = LinearRegression()
     model.fit(x, y)
     model = LinearRegression().fit(x,y)

[4]  y_pred = model.predict(x)

     plt.figure(figsize=(10,6))
     plt.scatter(x,y)
     plt.scatter(x,y_pred,color='r')
     x_seq = np.linspace(x.min(),x.max(),15).reshape((-1,1))
     plt.plot(x_seq, model.predict(x_seq))
     plt.show()
model.predict([[30]])
```

DT:分类苹果和梨

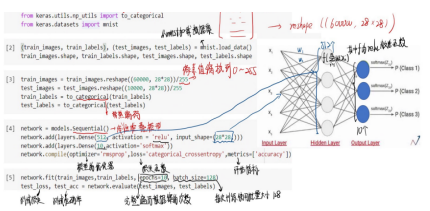
```
from sklearn import tree
import numpy as np

features = ['color', 'width', 'height']
fruitnames=['apple', 'pear']

data =np.array( [[10, 5.3, 4.8], [62, 5.5, 7.8], [85, 5.7, 4.9], [58, 6.8, 5.9],
       [15, 8.6, 7.7], [100, 5.4, 8.2], [22, 5.8, 5.6], [71, 4.5, 6.3]])
labels = np.array([0, 1, 0, 1, 0, 1, 0, 1])

[3] fruit_classifier = tree.DecisionTreeClassifier()

[4] fruit_classifier = fruit_classifier.fit(data, labels)

[5] predit = fruit_classifier.predict(np.array([[101, 6.3, 7.9],[25, 4.5, 3.8],[85, 5.2, 4.0]]))
    predit

    array([1, 0, 0])

[9]
    list(map(lambda i: fruitnames[i], predit))

    ['pear', 'apple', 'apple']

[9]  print(fruitnames[int(fruit_classifier.predict([[101,6.3,7.9]]))])

     pear
```

Kmeans：

```
from pylab import plt
import numpy as np
import pandas as pd

plt.style.use('seaborn')

from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

x, y = make_blobs(n_samples=100, centers=4, random_state=500, cluster_std=1.25)
x, y

model = KMeans(n_clusters=4, random_state = 0)
model.fit(x)

y_ = model.predict(x)

plt.figure(figsize=(10,6))
plt.scatter(x[:,0],x[:,1], c=y, cmap='coolwarm')

plt.figure(figsize=(10,6))
```

Deep learning: TensorFlow and Keras （classifying digits）

```
[1]  import numpy as np
     import matplotlib.pyplot as plt
     from keras import models
     from keras import layers
     from keras.utils.np_utils import to_categorical
     from keras.datasets import mnist

[2]  (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
     train_images.shape, train_labels.shape, test_images.shape, test_labels.shape

[3]  train_images = train_images.reshape((60000, 28*28))/255
     test_images = test_images.reshape((10000, 28*28))/255
     train_labels = to_categorical(train_labels)
     test_labels = to_categorical(test_labels)

[4]  network = models.Sequential()
     network.add(layers.Dense(512, activation = 'relu', input_shape=(28*28,)))
     network.add(layers.Dense(10,activation='softmax'))
     network.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])

[5]  network.fit(train_images,train_labels, epochs=10, batch_size=128)
     test_loss, test_acc = network.evaluate(test_images, test_labels)
```



**Lec3: Financial forecasting by ML**

After importing pandas-datareader in Jupyter notebook, we can use their **datareader method** to obtain prices of stocks in various markets.

Getting general stock example:

```
[1]  !pip install yfinance

[2]  from pandas_datareader import data
     import yfinance as yf
     yf.pdr_override()

     ticker = input("Input stock ticker: ")
     start = input("start date:")
     end = input("end date:")
     df = data.get_data_yahoo(tickers=ticker, start=start, end=end)
     df
```

Candle Stick chart for stock prices:

```
!pip install yfinance
```

```
from pandas_datareader import data
import yfinance as yf
yf.pdr_override()
```

```
mtr = data.get_data_yahoo(tickers='0066.HK', start = "2020-01-01", end = "2020-12-31")
mtr
```

```
import matplotlib.pyplot as plt

mtr['20d']=np.round(mtr['Open'].rolling(20).mean(),2)
mtr['40d']=np.round(mtr['Open'].rolling(40).mean(),2)
plt.figure(figsize=(15,4))
plt.grid(True)
plt.plot(mtr[['Open','20d','40d']])
plt.show()
```

```
import mplfinance as mpf
mpf.plot(mtr,type='candle',style='charles',figratio=(10,6),mav=(20,40), volume=True)
```

```
mpf.plot(mtr[:100], type='candle',style='sas',figratio=(10,6),mav=(20,40),volume=True)
```

## Stock prediction:

> Technical analysis:
mainly focuses on numerical stock data, such as closing price, opening prices, trading volume, ...
> qualitative analysis:
based on external factors, such as political situation, social media, and company profile.

1. Technical analysis:

>>Statistical methods for time series prediction
Example: Autoregressive Integrated Moving Average (ARIMA) model

```
from statsmodels.tsa.arima_model import ARIMA
model=ARIMA(df['Sales'],order=(1,1,1))
model_fit=model.fit()
model_fit.summary()
```

>>Machine Learning method
① LinearRegression model to predict stock price of next day.

```
!pip install yfinance
```

```
from pandas_datareader import data
import yfinance as yf
yf.pdr_override()
```

```
mtr = data.get_data_yahoo('MTR',start="2010-01-01", end="2020-06-30")
mtr
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
today =mtr['Open'].iloc[:len(mtr)-1].reset_index(drop=True)
nextday = mtr['Open'].iloc[1:].reset_index(drop=True)
```

```
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score

today = np.array(today).reshape(-1,1)
nextday = np.array(nextday)
X_train, X_test, y_train, y_test = train_test_split(today,
                                    nextday, test_size=0.2)
```

```
reg=linear_model.LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
```

//

```
df = pd.DataFrame()
df['X_test']=X_test.reshape((1,-1))[0]
df['y_test']=y_test
df['y_pred']=y_pred
df.head(10)
```

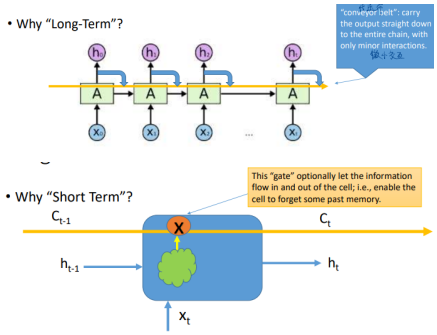
x

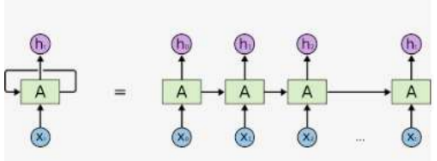
```
plt.figure(figsize=(16,10))
plt.scatter(y_test,y_pred)
```

//

```
res_msr = mean_squared_error(y_test, y_pred)
res_r2_score = r2_score(y_test, y_pred)
print(f'mean_squared_error = {res_msr} and r2_score = {res_r2_score}')
```

## ②🌟Deep learning method(LSTM) for predicting HengSeng index



```
!pip install yfinance
import yfinance as yf
from pandas_datareader import data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
yf.pdr_override()
```

```
df_sse = data.get_data_yahoo(tickers='000001.SS',start='2013-01-01', end='2022-12-31')
df_sse = df_sse['Close']
```

```
df=df_sse
groupA = df[:len(df)-14] # past ten days sequence
groupB = df[14:] # the coming 5th day sequence
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
rdata = scaler.fit_transform(pd.concat([groupA.reset_index(drop=True), groupB.reset_index(drop=True)],axis=1))
training_size=int(len(rdata)*0.8)
train_period,test_period=rdata[:training_size],rdata[training_size:]
```

from sklearn.preprocessing import **MinMaxScaler**
scaler=MinMaxScaler(feature_range=(0,1))
rdata = scaler.fit_transform(pd.concat([groupA.reset_index(drop=True), groupB.reset_index(drop=True)],axis=1))
training_size=int(len(rdata)*0.8)
train_period,test_period=rdata[:training_size],rdata[training_size:]

```
from keras.preprocessing.sequence import TimeseriesGenerator
train = TimeseriesGenerator(train_period[:,0], train_period[:,-1], length=10, batch_size=2000)
test = TimeseriesGenerator(test_period[:,0], test_period[:,-1], length=10, batch_size=1000)
X_train, y_train = list(train)[0][0], list(train)[0][1]
X_test, y_test = list(test)[0][0], list(test)[0][1]
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(10,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

from keras.preprocessing.sequence import TimeseriesGenerator
train = TimeseriesGenerator(train_period[:,0], train_period[:,-1], length=10, batch_size=2000)
test = TimeseriesGenerator(test_period[:,0], test_period[:,-1], length=10, batch_size=1000)
X_train, y_train = list(train)[0][0], list(train)[0][1]
X_test, y_test = list(test)[0][0], list(test)[0][1]

```
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64,verbose=True)
```

```
predict_X_train=model.predict(X_train)
predict_X_test=model.predict(X_test)
```

```
train_predict = scaler.inverse_transform(np.concatenate((np.random.rand(len(predict_X_train),1),predict_X_train),axis=1))
test_predict = scaler.inverse_transform(np.concatenate((np.random.rand(len(predict_X_test),1),predict_X_test),axis=1))
train_predict = train_predict[:,-1]
test_predict = test_predict[:,-1]
```

predict_X_train=model.predict(X_train)
predict_X_test=model.predict(X_test)
train_predict = scaler.inverse_transform(np.concatenate((np.random.rand(len(predict_X_train),1),predict_X_train),axis=1))
test_predict = scaler.inverse_transform(np.concatenate((np.random.rand(len(predict_X_test),1),predict_X_test),axis=1))
train_predict = train_predict[:,-1]
test_predict = test_predict[:,-1]

```
# train plot
# create a array same size with rdata aka actual data
trainPredictPlot=np.empty_like(rdata)
trainPredictPlot = trainPredictPlot[:,0]
# empty the array, waiting for assignment
trainPredictPlot[:]=np.nan # nan is for error supression
# assign train prediction into desginated area:
trainPredictPlot[:len(train_predict)] = train_predict

testPredictPlot=np.empty_like(rdata)
testPredictPlot = testPredictPlot[:,0]
testPredictPlot[:] = np.nan
testPredictPlot[len(rdata)-len(test_predict):] = test_predict

plt.figure(figsize=(10,6))
plt.plot(scaler.inverse_transform(rdata)[:,-1],label="actual stock price")
plt.plot(testPredictPlot,label="test prediction")
plt.plot(trainPredictPlot,label="train prediction")
plt.legend()
plt.title("actual/prediction graph")

plt.show
```

```
plt.figure(figsize=(10,6))
plt.plot(scaler.inverse_transform(rdata)[len(rdata)-len(test_predict):,-1],label="actual")
plt.plot(test_predict,label="test prediction")
plt.legend()
plt.title("test period")
plt.show
```

```
train_accuracy=np.sum(np.absolute(train_predict-df[:len(train_predict)]))
print("train accuracy =",train_accuracy)
```

```
test_accuracy=np.sum(np.absolute(test_predict - scaler.inverse_transform(rdata)[len(rdata)-len(test_predict)-len(rdata):,-1]))
print("test accuracy =",test_accuracy)
```

**Lec 4:**

# Reinforcement Learning:

**Environment:** defines the problems at hand. This can be a computer game or financial market.

**State**:subsumes all relevant parameters that describe the current state of the environment. 游戏屏幕上的像素点和金融市场中的价格。

**Agent:** subsumes all elements of the RL algorithms that interact with the environment and that learns from these interactions. 玩家或者trader

**Action:** an agent can choose one action from a limited set of allowed actions.

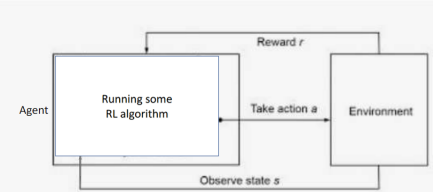
**Step:** Given an action of an agent, the state of the environment is updated, One such update is called a step.

**Reward:** depend on action an agent chooses, a reward or penalty is awarded.

**Target:** what the agent tries to maximize.分数或利润

**Policy:** defines which action an agent takes given a certain state of the environment.

**Episode:** a set of steps from the initial state of the environment until success is achieved or failure is observed.




## *Q-learning:*

1. Q-learning is based on **Bellman's principle** of optimality:
An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision
2. Idea:
• We try to learn the Q-value Q(S, A) for all possible states S, and actions A.
Intuitively, Q(S, A) tells us how good it is if we take action A when we are
current in state S.
• Initially, Q(S,A) is assigned an arbitrary number, say 0, and we will improve it during the learning process.
• Suppose when at state S, we have chosen action A and the action moves us to S'.
• Suppose there are k possible actions A1, A2, ... Ak. Then, after we move to S', the next action can be A1, A2,..., Ak. Which one?
• By Bellman's principle of optimality, we will choose the one MA such that Q(S', A') = max Q(S', Ai)
• Then, we need to update Q(S, A).
> Compare Q(S',A') and Q(S,A) as we have looked a step further.
• If Q(S',A') > Q(S, A), then we need to increase Q(S,A) somewhat
• If Q(S',A') < Q(S, A), we need to decrease Q(S,A) somewhat.

**Q(S, A)=Q(S, A)+**
$\alpha \cdot (reward + \gamma \cdot Q(S', A') - Q(S, A))$
**α is learning rate, γ is a discount factor.**

Taking random actions:

```
import numpy as np
import random
```

```
import gym
env = gym.make('CartPole-v1')
```

```
state = env.reset()
done = False
no_steps = 0
while done == False:
    a = random.randint(0,1)
    state, reward, done, info = env.step(a)
    no_steps = no_steps + 1
print(f"no_steps = {no_steps}")
```

## After q-learning:

```
!pip install gym

from sklearn.preprocessing import KBinsDiscretizer
import gym
import numpy as np
import time, math, random
from typing import Tuple
env = gym.make('CartPole-v1')
```

```
n_bins = (6, 12)
lower_bounds = [env.observation_space.low[2], -math.radians(50)]
upper_bounds = [env.observation_space.high[2], math.radians(50)]

def discretizer(_, __, angle, pole_velocity):
    est = KBinsDiscretizer(n_bins = n_bins, encode = 'ordinal', strategy='uniform')
    est.fit([lower_bounds, upper_bounds])
    return tuple(map(int, est.transform([[angle, pole_velocity]])[0]))
```

```
Q_table = np.zeros(n_bins + (env.action_space.n,))

def policy(state: tuple):
    return np.argmax(Q_table[state])

def new_Q_value(reward: float, state_new: tuple, discount_factor=1):
    future_optimal_value = np.max(Q_table[state_new])
    learned_value = reward + discount_factor * future_optimal_value
    return learned_value

def learning_rate(n:int, min_rate=0.01):
    return max(min_rate, min(1.0, 1.0-math.log10((n+1)/25)))

def exploration_rate(n:int, min_rate=0.1):
    return max(min_rate, min(1, 1.0-math.log10((n+1)/25)))
```

```
n_episodes = 250
for e in range(n_episodes):
    current_state, done = discretizer(*env.reset()), False
    while done == False:
        action = policy(current_state)
        if np.random.random() < exploration_rate(e):
            action = env.action_space.sample()
        obs, reward, done, _ = env.step(action)
        new_state = discretizer(*obs)

        lr = learning_rate(e)
        learnt_value = new_Q_value(reward, new_state)
        old_value = Q_table[current_state][action]
        Q_table[current_state][action] = (1-lr)*old_value + lr*learnt_value

        current_state = new_state
        if e % 3 == 0:
            env.render()
    time.sleep(0.5)
env.close()
```

## RL for Trading:

**Gym-anytrading:**

• For our purpose (stock trading), we use one of gym environment simulation tools, anytrading.
• AnyTrading is a collection of OpenAI Gym environments for reinforcement learning trading algorithms.
• More specially, it provides three Gym environments, namely, TradingEnv, ForexEnv, and StocksEnv, and facilitate many useful methods for developers to implement and test RL-based algorithms for trading in the FOREX and the Stock markets
Environment Properties:
• **Trading Positions:** Short (=0) and Long (=1)
> Long position wants to buy shares when prices are low and profit by sticking with them
while their value is going up, and Short position wants to sell shares with high value and
use this value to buy shares at a lower value, keeping the difference as profit.
> Long position buy action
> Short position sell action.
• prices: "Real" prices over time. Used to calculate rewards and final profit.
• window_size: Number of ticks (previous ticks + current tick) in an observation.
• profit: The amount of units of currency achieved by starting with 1.0 unit (profit = FinalMoney/StartingMoney).

**gymstockEnv.ipynb:**

```
[1] !pip install gym-anytrading gym yfinance
```

```
[2] import gym
    import gym_anytrading

    import numpy as np
    import pandas as pd
    from pylab import plt
```

```
[3] from pandas_datareader import data
    from datetime import datetime
    import yfinance as yf
    yf.pdr_override()

    start = datetime.strptime('2020-01-01','%Y-%m-%d')
    end = datetime.strptime('2021-06-30','%Y-%m-%d')
    df = data.get_data_yahoo('0066.HK', start=start, end=end)
```

```
plt.figure(figsize=(15,6))
plt.plot(df['Close'])
plt.show()
```

```
env = gym.make('stocks-v0', df=df, frame_bound=(5,200), window_size=5)
state = env.reset()
```

```
while True:
    action = env.action_space.sample()
    obser, reward, done, info = env.step(action)
    if done:
        print('info', info)
        break
plt.figure(figsize=(15,6))
plt.cla()
env.render_all()
plt.show()
```

(上文这个用**sample**不太**ok**)

**stable_baselines:**It provides many state-of-the-art RL model building algorithms

**A2C: Advanced Actor Critic RL algorithm**

```
[1] !pip install stable-baselines3 gym gym-anytrading tensorflow
```

```
import gym
import gym_anytrading
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from stable_baselines3.common.vec_env import DummyVecEnv
from stable_baselines3 import A2C
```

```
[3] !pip install yfinance
```

```
[4] from pandas_datareader import data
    from datetime import datetime
    import yfinance as yf
    yf.pdr_override()
```

```
[5] start = datetime.strptime('2020-01-01','%Y-%m-%d')
    end = datetime.strptime('2021-06-30','%Y-%m-%d')
    df = data.get_data_yahoo('MTR', start=start, end=end)
```

```
env_maker = lambda: gym.make('stocks-v0', df=df, frame_bound=(5,100), window_size=5)
env = DummyVecEnv([env_maker])
```

```
model = A2C('MlpPolicy', env, verbose=1)
model.learn(total_timesteps=2000)
```

```
env = gym.make('stocks-v0',df=df, frame_bound=(90,110), window_size=5)
obs = env.reset()
while True:
    obs = obs[np.newaxis, ...]
    action, _states = model.predict(obs)
    obs, rewards, done, info = env.step(action)
    if done:
        print("info", info)
        break
```

```
plt.figure(figsize=(15,6))
plt.cla()
env.render_all()
plt.show()
```

## Lec 5: sentiment analysis

General steps:

• Text Input

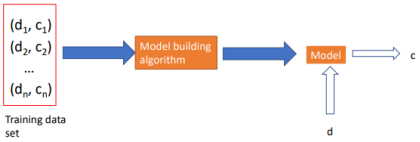
• Tokenization

• Stop word filtering(Examples of stopwords: are, and, am, at, be, but, so, such, then, …)

• Negation handling

• Stemming

• Classification




**NLTK:**

```
import nltk
from nltk.corpus import movie_reviews
import random
```

```
nltk.download('movie_reviews')
```

```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
words_features = list(all_words)[:2000]
```

```
documents = [(list(movie_reviews.words(fileid)),category)
       for category in movie_reviews.categories()
       for fileid in movie_reviews.fileids(category)]
```

```
def document_features(document):
    document_words = set(document)
    features = {}
    for word in words_features:
        features[word] = (word in document_words)
    return features
```

```
random.shuffle(documents)
featuresets = [(document_features(d),c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
model = nltk.NaiveBayesClassifier.train(train_set)
```

```
model.classify(test_set[1][0])
```

```
nltk.classify.accuracy(model, test_set)
```

```
model.show_most_informative_features(5)
```

LSTM, and other traditional models for NLP, fell **short** when it came to handling long-range dependencies (which is essential to translate a long and complex sentence).

• **Transformer** solves this difficult problem by introducing the "technique of attention", which helps us relate different positions of a single sequence in order to compute a representation of the sequence. (ChatGPT)

• It is a deep neural network architecture that aims to solve sequence-to-sequence tasks (NLP is a sequence-to-sequence tasks).

**"en-sentiment"**

**Code**:**(根据题目要求更换'q'和API)**

```
import pandas as pd
import requests
import flair
from google.colab import files
```

```
import pandas as pd
import requests
import flair
```

```
def get_data(tweet):
    data = {
        'id':tweet['id_str'],
        'created_at':tweet['created_at'],
        'text':tweet['full_text']
    }
    return data
```

### Prepare the data

```
params = {'q':'tesla', 'tweet_mode':'extended', 'lang':'en', 'count':'100'}
fin = open('BearerToken.txt','r')
BearToken = fin.readlines()[0].strip()
response = requests.get(
'https://api.twitter.com/1.1/search/tweets.json', params=params,
headers = {'authorization':'Bearer '+ BearToken})

tweets = pd.DataFrame()
for tweet in response.json()['statuses']:
    row = get_data(tweet)
    tweets = tweets.append(row, ignore_index=True)
```

### Get the pre-trained model for SA

```
sentiment_model = flair.models.TextClassifier.load('en-sentiment')
```

```
2022-02-16 11:32:14,605 loading file C:\Users\hfting\.flair\models\sen
```

```
probs = []
sentiments = []

for tweet in tweets['text']:
    sentence = flair.data.Sentence(tweet)
    sentiment_model.predict(sentence)
    probs.append(sentence.labels[0].score)
    sentiments.append(sentence.labels[0].value)

tweets['probability'] = probs
tweets['sentiment'] = sentiments
```

```
tweets
```

```
sentiment_model = flair.models.TextClassifier.load('en-sentiment')
```

```
sentence = flair.data.Sentence("Hello, I am happy")
sentence
```

```
sentiment_model.predict(sentence)
sentence
```

```
sentence.labels
```

```
sentence.labels[0]
```

```
probs = []
sentiments = []

for tweet in tweets['text']:
    sentence = flair.data.Sentence(tweet)
    sentiment_model.predict(sentence)
    probs.append(sentence.labels[0].score)
    sentiments.append(sentence.labels[0].value)

tweets['probability'] = probs
tweets['sentiment'] = sentiments
```

```
tweets
```

```
tweets[['text', 'probability', 'sentiment']]
```

**relate sentiment with stock prices**

```
!pip install flair
!pip install yfinance
```

```
import requests
import pandas as pd
import re
from pylab import plt
```

```
whitespace = re.compile(r"\s+")
web_address = re.compile(r"(?i)https(s):\/\/[a-z0-9.~_\-\/]+")
tesla = re.compile(r"(?i)@Tesla(?=\b)")
user = re.compile(r"(?i)@[a-z0-9_]+")

def clean(tweet):
    tweet = whitespace.sub(' ', tweet)
    tweet = web_address.sub('', tweet)
    tweet = tesla.sub('Tesla', tweet)
    tweet = user.sub('', tweet)
    return tweet
```

```
BearerToken = "AAAAAAAAAAAAAAAAAAAAA"

api = 'https://api.twitter.com/2/tweets/search/recent'
headers = {'authorization': f'Bearer {BearerToken}'}
params = {
    'query': '(TESLA OR tsla OR Musk) (lang:en)',
    'max_results': '10',
    'tweet.fields':'created_at,lang'
}
```

```
from datetime import datetime, timedelta
dtformat = '%Y-%m-%dT%H:%M:%S.000Z'

def go_back(now, mins):
    return now - timedelta(minutes = mins)
```

```
df = pd.DataFrame()
now = datetime.now() - timedelta(days=1)
last_week = now - timedelta(days=10)
while True:
    if now <= last_week:
        break
    pre60 = go_back(now, 60)
    params['start_time'] = pre60.strftime(dtformat)
    params['end_time'] = now.strftime(dtformat)
    response = requests.get(api, params = params, headers =headers)
    tweetdict = response.json()
    if 'data' in tweetdict:
      for tweet in response.json()['data']:
          row = {'created_at':tweet['created_at'][5:10],
                 'text':clean(tweet['text'])}
          df = df.append(row, ignore_index=True)
    now = pre60
```

```
import flair
sentiment_model = flair.models.TextClassifier.load('en-sentiment')
wsentiments = []
for tweet in df['text']:
    sentence = flair.data.Sentence(tweet)
    sentiment_model.predict(sentence)
    prob = sentence.labels[0].score
    sentiment = 1 if sentence.labels[0].value == 'POSITIVE' else -1
    wsentiments.append(prob * sentiment)
df['wsentiments'] = wsentiments
df_sent = df.groupby('created_at')['wsentiments'].mean()
```

```
[20]
    import yfinance as yf
    yf.pdr_override()

    from pandas_datareader import data

    start = datetime.now() - timedelta(days=10)
    end = datetime.now() - timedelta(days=1)
    stockprice = data.get_data_yahoo('TSLA', start=start, end=end)
```

```
[28] indexlist = stockprice.index.to_list()
     for i in range(len(indexlist)):
       indexlist[i] = str(indexlist[i])[5:10]
```

```
[14] stockprice.index = indexlist
```

```
fig = plt.figure(figsize=(18,5))
ax1 = fig.add_subplot(121)
ax1.plot(df_sent)
ax1.title.set_text('Sentiment')
ax2 = fig.add_subplot(122)
ax2.title.set_text('Stock price')
ax2.plot(stockprice['Close'])
```

## Lec 6:Fault detection

• Areas where fraud detection and prevention are applied include

insurance claims, money laundering, electronic payments, and bank transactions, both online and offline.

• The challenge is to quickly identify and separate anomalous transactions from those that are legitimate, without impacting on customer experience.

• Note that companies are also suffering because of lost sales when genuine transactions are declined by fraud management systems.

## How can Machine learning help?

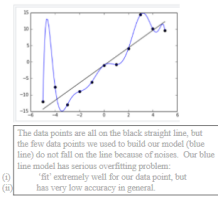
• Before Computer: Banks and financial institutions analyzed their customers' behavioral patterns for any signs of abnormality, and designed checking rules to "flag" abnormal transactions manually.

• After Computer: Computers gave them the ability to identify and flag abnormal transactions in real-time.

• After ML: Machine learning tools 'learn' new patterns, without the need for human intervention. This allows models to adapt over time to uncover previously unknown patterns or identify new tactics that might be employed by fraudsters.

Key for ML's success:

- High accuracy
  - small number of false negatives
  - small number of false positives
- Avoid overfitting
  - We build the ML model based on some dataset
  - The model may have very high accuracy for this dataset, but very low accuracy in general.



The data points are all on the black straight line, but the few data points we used to build our model (blue line) do not fall on the line because of noises. Our blue line model has serious overfitting problem:
(i) "fit" extremely well for our data point, but
(ii) has very low accuracy in general.

## code:

• Based on the following transaction file: `tranrecords.csv`

• The csv if for the "comma-separated-values" format

```
1599.5483533432368,C560169,Paid_UnReconciled,Finance,2015,Aug,1
1.1614787867721998,C817105,Paid_UnReconciled,Purchasing,2018,Aug,0
11.885565040510526,C476755,Paid_Reconciled,Finance,2017,Apr,0
13.886409044014,E335726,Paid_Reconciled,R&D,2011,Aug,0
21.369041770353846,A818773,Paid_UnReconciled,R&D,2015,Jun,0
0.3106347611010127564,C141146,Paid_UnReconciled,R&D,2018,Jan,0
3.3556584009813974,E388954,Paid_Reconciled,Production,2012,Apr,0
2.1404702211954084,E453358,Paid_Reconciled,Finance,2017,Aug,0
22.156267234047004,C641107,Paid_Reconciled,Finance,2013,Apr,0
0.7910295964163332,:,Paid_UnReconciled,Purchasing,2016,Sep,0
151.33482871940614,C805852,Paid_UnReconciled,Marketing,2014,Aug,1
1.0224137650926193,E977677,Paid_Reconciled,R&D,2015,Nov,0
0.040415567013301,C320147,Paid_Reconciled,Purchasing,2014,Sep,0
7.825946900244843,C156772,Paid_UnReconciled,Finance,2019,Apr,0
3.6136035346377477,E908822,Paid_Reconciled,Finance,2018,Apr,0
6.585972233373637,C529086,Paid_Reconciled,Finance,:,Aug,0
2.968845717305337,E524327,Paid_Reconciled,Finance,2010,Apr,0
0.8460743446295853,E761698,Paid_Reconciled,Finance,2010,Jun,0
```

1.reading cvs file ':' represents missing values

```
import pandas as pd
df = pd.read_csv("tranrecords.csv", na_values=':')
df.info()
```

2. Know your data: Get some idea of missing values

```
In [6]: df.any()
Out[6]: Amount          True
        TranNo          True
        Status          True
        Department      True
        Fiscal Year     True
        Month           True
        RedFlag         True
        dtype: bool

In [8]: df.isnull().sum()
Out[8]: Amount          0
        TranNo         44
        Status         41
        Department     14
        Fiscal Year    55
        Month          22
        RedFlag         0
        dtype: int64
```

df.isnull()

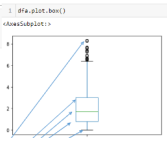
3.Cleaning your data: Handling the missing values

•To drop all the rows with missing values by executing df.dropna()

•To replace, for every missing value, by the mean of the column the missing value belongs. df.fillna(df.mean())

4.Cleaning your data: detecting outliers

```
dfa = df['Amount']
dfa.min(), dfa.quantile(q=0.25), dfa.quantile(q=0.5),
    dfa.quantile(q=0.75), dfa.max()

(0.0013588035896111,
 1.1768322150657688,
 4.672200947259141,
 19.919945218582576,
 4060.3644385505363)
```



• What exactly are outliers?

Based on the 5-number summary: xmin, Q1 , Q2 , Q3 , xmax

• Define the interquartile range IQR to be the value Q3 – Q1

• An outlier is the one with value either

  • <= (Q1 – 1.5 * IQR), or

  • >= (Q3 + 1.5 * IQR)

**outliers are important because they give signals for possible fraud transactions**

5.Preparing the data set: convert to numeric values

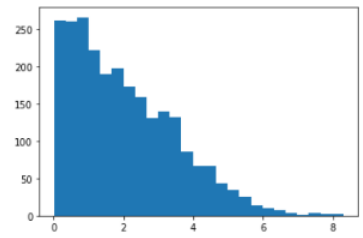
```
d = {"Jan":1, "Feb":2, "Mar":3, "Apr":4, "May":5,
     "Jun":6, "Jul":7, "Aug":8,
    "Sep":9, "Oct":10, "Nov":11, "Dec":12}
df['Month'] = df['Month'].map(d)
df['Month'].head(10)
```

```
0     1.0
1     9.0
2     5.0
3     5.0
4    10.0
5     5.0
6     1.0
7     4.0
8     5.0
9     4.0
Name: Month, dtype: float64
```

Preparing the data set: distributions of data:

```
pic = plt.hist(df['Amount'],bins=25)
```

```
df['Amount'] = df['Amount'].apply(lambda x: np.log(x+1))
pic = plt.hist(df['Amount'], bins=25)
```

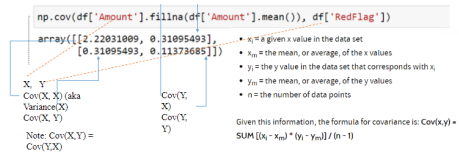



We use log-transformation here.

6.Choosing the columns (i.e.,"features") for ML

We compute, for each column, the compute the covariance of this column and the output column (i.e., the RedFlag column).

```
np.cov(df['Amount'].fillna(df['Amount'].mean()), df['RedFlag'])

array([[2.22031009, 0.31095493],
       [0.31095493, 0.11373685]])
```




7.How to handle rare classes (imbalanced)

  • Navie Random over-sample

  • ROSE: Random Over-Sample Examples

  • SMOTE: Synthetic Minority Oversample Technique

  • ADASYN: Adaptive Synthetic Method

8.We use the following ML methods from scikit-learn to construct classifiers

• GuassianNB: To learn and construct a Guassian Naive Bayes classifier

• LogisticRegression: To learn and construct Logistic Regression classifier

• DecisionTreeClassifier: To learn and construct a Decision Tree classifier

• RandomForestClassifier: To learn and construct a Random Forest classifier

• SGDClassifier: To learn and construct a Stochastic Gradient Descent (SGD) classifier

```
from sklearn.preprocessing import MinMaxScaler

data = pd.DataFrame(df)
scaler = MinMaxScaler()
numerical = ['Amount','Month','Status']
data[numerical] = scaler.fit_transform(data[numerical])
```

```
from sklearn.model_selection import train_test_split

data = data.dropna()
X_train, X_test, y_train, y_test = train_test_split(data[['Amount','Month','Status']],
                                                    data['RedFlag'], test_size=0.2, random_state=0)
print(f"Training set has {X_train.shape[0]} samples")
print(f"Testing set has {X_test.shape[0]} samples")
```

```
Training set has 1863 samples
Testing set has 466 samples
```

```
summary = pd.DataFrame(columns=['Learner', 'Train Time', 'Pred Time', 'Acc score', 'F1 score',
                                'Precision', 'Recall'])

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score, precision_score, \
        recall_score, classification_report, confusion_matrix

NB = GaussianNB()
start = time()
NB.fit(X_train, y_train)
mid = time()
pred = NB.predict(X_test)
end = time()

pred_res = pd.DataFrame(pred)
pred_res = pred_res.set_index(y_test.index)

summary = summary.append({'Learner':'GaussianNB','Train Time':mid-start, 'Pred Time':end-mid,
              'Acc score':accuracy_score(y_test, pred),
              'F1 score':f1_score(y_test,pred,average='macro'),
              'Precision':precision_score(y_test,pred,average='macro'),
              'Recall': recall_score(y_test,pred,average='macro')}, ignore_index=True)
summary
```

9.**Definitions for Precision and Recall**

Given a transaction, we say that it is a

- **true positive:** ML predicts positive, and it is indeed redflag in our DS
- **true negative**: ML predicts negative, and it is indeed no-redflag
- **false positive:** ML predicts positive, but it is no-redflag
- **false negative:** ML predicts negative, but it is redflag

$$\text{precision} = \frac{tp}{tp + fp}, \text{ and}$$

$$\text{recall} = \frac{tp}{tp + fn}$$

$F1$-score is the *harmonic mean* of precision and recall, i..e,

$$F1\text{-score} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

F1-score is better than Precision when there is a large class imbalance (like our example here).

```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(random_state=0)
start = time()
LR.fit(X_train, y_train)
mid = time()
pred = LR.predict(X_test)
end = time()

summary = summary.append({'Learner':'LogisticReg','Train Time':mid-start, 'Pred Time':end-mid,
              'Acc score':accuracy_score(y_test, pred),
              'F1 score':f1_score(y_test,pred,average='macro'),
              'Precision':precision_score(y_test,pred,average='macro'),
              'Recall': recall_score(y_test,pred,average='macro')}, ignore_index=True)
summary
```

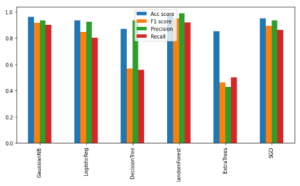
```
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier(max_features=0.2, max_depth=2,
                            min_samples_split=2, random_state=0)
start = time()
DT.fit(X_train, y_train)
mid = time()
pred = DT.predict(X_test)
end = time()
summary = summary.append({'Learner':'DecisionTree','Train Time':mid-start, 'Pred Time':end-mid,
              'Acc score':accuracy_score(y_test, pred),
              'F1 score':f1_score(y_test,pred,average='macro'),
              'Precision':precision_score(y_test,pred,average='macro'),
              'Recall': recall_score(y_test,pred,average='macro')}, ignore_index=True)
summary
```

```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(max_depth=2)
start = time()
RF.fit(X_train, y_train)
mid = time()
pred = RF.predict(X_test)
end = time()

summary = summary.append({'Learner':'RandomForest','Train Time':mid-start, 'Pred Time':end-mid,
              'Acc score':accuracy_score(y_test, pred),
              'F1 score':f1_score(y_test,pred,average='macro'),
              'Precision':precision_score(y_test,pred,average='macro'),
              'Recall': recall_score(y_test,pred,average='macro')}, ignore_index=True)
summary
```

```
from sklearn.ensemble import ExtraTreesClassifier

ET = ExtraTreesClassifier(max_depth=2)
start = time()
ET.fit(X_train, y_train)
mid = time()
pred = ET.predict(X_test)
end = time()

summary = summary.append({'Learner':'ExtraTrees','Train Time':mid-start, 'Pred Time':end-mid,
              'Acc score':accuracy_score(y_test, pred),
              'F1 score':f1_score(y_test,pred,average='macro'),
              'Precision':precision_score(y_test,pred,average='macro'),
              'Recall': recall_score(y_test,pred,average='macro')}, ignore_index=True)
summary
```

```
from sklearn.linear_model import SGDClassifier

SGD = SGDClassifier(loss='hinge', penalty="l2")
start = time()
SGD.fit(X_train, y_train)
mid = time()
pred = SGD.predict(X_test)
end = time()

summary = summary.append({'Learner':'SGD','Train Time':mid-start, 'Pred Time':end-mid,
              'Acc score':accuracy_score(y_test, pred),
              'F1 score':f1_score(y_test,pred,average='macro'),
              'Precision':precision_score(y_test,pred,average='macro'),
              'Recall': recall_score(y_test,pred,average='macro')}, ignore_index=True)
summary
```

```
summary[['Learner','Acc score','F1 score','Precision','Recall']].plot(kind='bar', x='Learner', figsize=(10,5))
```




## Lec 7:option pricing

**Option Pricing (European):** (不考**American**)

Objective of option pricing: The premium must be fair so that the buyer and seller will have an equal chance to win (i.e., gain profit).

**Two types of options:**

•Call option: allow the holder to buy the stock at the strike price on the expiry date.

•Put option: allow the holder to sell the stock at the strike price on the expiry date.

•Buyer has the right not to exercise the contract, and in such cases, he loses nothing, except for the premium he paid when he bought the option.

•Seller has the obligation to execute the contract if the buyer decides to do so.

**Black-scholes-merton** achieve this objective:

$$C = S \times N(d_1) - Ke^{-rT} \times N(d_2)$$

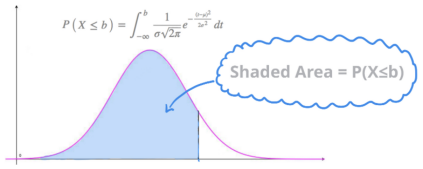
where $d_1 = \frac{\ln(\frac{S}{K}) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}$, $d_2 = d_1 - \sigma\sqrt{T}$, and

- $C$ = option price
- $N$ = Cumulative Distribution Function (CDF) of the normal distribution
- $S$ = stock price of today (the date the contract is signed)
- $K$ = strike price
- $r$ = risk-free rate
- $T$ = time to maturity = (expiry date - today)/365
- $\sigma$ = volatility of the stock (i.e., the standard deviation of the stock's move over a set time of period). This $\sigma$ depends on the stock's move in the future, but we can only use the stock's move in the pass to "estimate" $\sigma$.
- $e$ = 2.7182718... the Euler constant

N(d1) & N(d2):

N(): Normal distribution with mean=0 and standard deviation = 1

$$N(d) = \Pr(X \leq d)$$




How to find N value? Example: 1.14, 先看x轴找到1.1, 在看小数点后第二位是什么 (纵轴), 锁定概率P.

| $x$ | 0 | 0.01 | 0.02 | 0.03 | 0.04 | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.5 | 0.50399 | 0.50798 | 0.51197 | 0.51595 | 0.51994 | 0.52392 | 0.5279 | 0.53188 | 0.53586 |
| 0.1 | 0.53983 | 0.5438 | 0.54776 | 0.55172 | 0.55567 | 0.55962 | 0.56356 | 0.56749 | 0.57142 | 0.57535 |
| 0.2 | 0.57926 | 0.58317 | 0.58706 | 0.59095 | 0.59483 | 0.59871 | 0.60257 | 0.60642 | 0.61026 | 0.61409 |
| 0.3 | 0.61791 | 0.62172 | 0.62552 | 0.6293 | 0.63307 | 0.63683 | 0.64058 | 0.64431 | 0.64803 | 0.65173 |
| 0.4 | 0.65542 | 0.6591 | 0.66276 | 0.6664 | 0.67003 | 0.67364 | 0.67724 | 0.68082 | 0.68439 | 0.68793 |
| 0.5 | 0.69146 | 0.69497 | 0.69847 | 0.70194 | 0.7054 | 0.70884 | 0.71226 | 0.71566 | 0.71904 | 0.7224 |
| 0.6 | 0.72575 | 0.72907 | 0.73237 | 0.73565 | 0.73891 | 0.74215 | 0.74537 | 0.74857 | 0.75175 | 0.7549 |
| 0.7 | 0.75804 | 0.76115 | 0.76424 | 0.7673 | 0.77035 | 0.77337 | 0.77637 | 0.77935 | 0.7823 | 0.78524 |
| 0.8 | 0.78814 | 0.79103 | 0.79389 | 0.79673 | 0.79955 | 0.80234 | 0.80511 | 0.80785 | 0.81057 | 0.81327 |
| 0.9 | 0.81594 | 0.81859 | 0.82121 | 0.82381 | 0.82639 | 0.82894 | 0.83147 | 0.83398 | 0.83646 | 0.83891 |
| 1 | 0.84134 | 0.84375 | 0.84614 | 0.84849 | 0.85083 | 0.85314 | 0.85543 | 0.85769 | 0.85993 | 0.86214 |
| 1.1 | 0.86433 | 0.8665 | 0.86864 | 0.87076 | 0.87286 | 0.87493 | 0.87698 | 0.879 | 0.881 | 0.88298 |
| 1.2 | 0.88493 | 0.88686 | 0.88877 | 0.89065 | 0.89251 | 0.89435 | 0.89617 | 0.89796 | 0.89973 | 0.90147 |
| 1.3 | 0.9032 | 0.9049 | 0.90658 | 0.90824 | 0.90988 | 0.91149 | 0.91308 | 0.91466 | 0.91621 | 0.91774 |
| 1.4 | 0.91924 | 0.92073 | 0.9222 | 0.92364 | 0.92507 | 0.92647 | 0.92785 | 0.92922 | 0.93056 | 0.93189 |
| 1.5 | 0.93319 | 0.93448 | 0.93574 | 0.93699 | 0.93822 | 0.93943 | 0.94062 | 0.94179 | 0.94295 | 0.94408 |
| 1.6 | 0.9452 | 0.9463 | 0.94738 | 0.94845 | 0.9495 | 0.95053 | 0.95154 | 0.95254 | 0.95352 | 0.95449 |
| 1.7 | 0.95543 | 0.95637 | 0.95728 | 0.95818 | 0.95907 | 0.95994 | 0.9608 | 0.96164 | 0.96246 | 0.96327 |
| 1.8 | 0.96407 | 0.96485 | 0.96562 | 0.96638 | 0.96712 | 0.96784 | 0.96856 | 0.96926 | 0.96995 | 0.97062 |
| 1.9 | 0.97128 | 0.97193 | 0.97257 | 0.9732 | 0.97381 | 0.97441 | 0.975 | 0.97558 | 0.97615 | 0.9767 |

N(1.14)

Example calculate:

- Stock Price $S = \$62$.
- Strik Price $K = \$60$.
- Time to Expiration $T - t = 40$ days, which is a fraction of $40/365 = 0.10959$ of a year.
- Volatility $\sigma = 32\% = 0.32$.
- Risk-Free rate $r = 4\% = 0.04$ (daily compounding).

Pluggin in the Black-Scholes equation, we get

$$d_1 = \frac{\ln(62/60) + [0.04 + 0.5(0.32)^2](40/365)}{0.32\sqrt{\frac{40}{365}}} = 0.40$$

$$d_2 = 0.404 - 0.32\sqrt{40/365} = 0.30$$

$$N(0.40) = 0.6554, \quad N(0.30) = 0.6179$$

Now the price of the option is

$$C = (62)(0.6554) - [60/e^{0.04(40/365)}](0.6179) = \$3.72.$$

- Limitation of Black-Scholes Model
  - The log-normal assumption does not capture extreme movements such as stock market crashes.
  - Volatility and interest rate are not constant throughout the option's life in practice
  - The model is very sensitive to the value of volatility which is difficult to estimate
  - The model normally uses historic volatility for the option price for a future period
  - Application to non-traded assets is questionable (e.g. employee stock options)

关于**paper《Option Pricing with Deep Learning》2022-8** 最后表格题

**Conclusion**:

Even with a naïve estimation of volatility, we were able to achieve performance much superior to Black-Scholes using the same set of features. By learning from historical options data, we were able to evade financial assumptions to view options pricing as a function that is well approximated by a neural network. We've found even better performance by forecasting bid and ask prices separately instead of the equilibrium, so future studies using historical data should consider this alternative problem formulation. With more

time, we would like to continue searching for optimal hyperparameters for our LSTM approach. We could also train models on the reverse problem to find the volatility implied by a given option price. This would allow us to plot the volatility surface and compare it with the volatility surfaces from the Heston model or GARCH family models. Additionally, we could do a deeper error analysis and examine pricing bias to see if our models perform better or worse given particular features (near the money, time until expiry, etc.), as well as estimating the partial derivatives of this model to draw curves of the risk metrics or Greeks and compare with Black-Scholes Greeks curves.

```
from keras.models import Sequential
from keras.layers import Dense, Activation, LeakyReLU, BatchNormalization
from keras import backend
from keras.callbacks import TensorBoard
from keras.optimizers import Adam
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
Using TensorFlow backend.
```

```
# Hyperparams
n_units = 400
layers = 4
n_batch = 4096
n_epochs = 10
```

```
df = pd.read_csv('../options-df-sigma.csv')
df = df.dropna(axis=0)
df = df.drop(columns=['date', 'exdate', 'impl_volatility', 'volume', 'open_interest'])
df.strike_price = df.strike_price / 1000
call_df = df[df.cp_flag == 'C'].drop(['cp_flag'], axis=1)
put_df = df[df.cp_flag == 'P'].drop(['cp_flag'], axis=1)
```

```
call_df.head()
```

| | strike_price | best_bid | best_offer | date_ndiff | treasury_rate | closing_price | sigma_20 |
|---|---|---|---|---|---|---|---|
| 0 | 600.0 | 28.000 | 29.000 | 47 | 5.17 | 624.22 | 0.007761 |
| 10 | 475.0 | 152.875 | 153.875 | 145 | 5.12 | 624.22 | 0.007761 |
| 13 | 600.0 | 52.375 | 53.375 | 327 | 5.05 | 624.22 | 0.007761 |
| 17 | 610.0 | 20.000 | 20.750 | 47 | 5.17 | 624.22 | 0.007761 |
| 18 | 675.0 | 34.000 | 35.000 | 691 | 5.10 | 624.22 | 0.007761 |

```
call_X_train, call_X_test, call_y_train, call_y_test = train_test_split(call_df.drop(['best_bid', 'best_offer'], axis=1),
                                                                        (call_df.best_bid + call_df.best_offer) / 2,
                                                                        test_size=0.01, random_state=42)
```

```
model = Sequential()
model.add(Dense(n_units, input_dim=call_X_train.shape[1]))
model.add(LeakyReLU())

for _ in range(layers - 1):
    model.add(Dense(n_units))
    model.add(BatchNormalization())
    model.add(LeakyReLU())

model.add(Dense(1, activation='relu'))

model.compile(loss='mse', optimizer=Adam())
```

```
history = model.fit(call_X_train, call_y_train,
                    batch_size=n_batch, epochs=n_epochs,
                    validation_split = 0.01,
                    callbacks=[TensorBoard()],
                    verbose=1)
```

```
call_y_pred = model.predict(call_X_test)
```

```
diff = (call_y_test.values - call_y_pred.reshape(call_y_pred.shape[0]))
```

```
np.mean(np.square(diff))
```

# Assignment1-partB

```
import os
import gym as gym
from stable_baselines3 import PPO
import gym
from stable_baselines3.common.callbacks import CheckpointCallback

log_dir ="cachefile/"
os.makedirs(log_dir, exist_ok=True)
env = gym.make("PongDeterministic-v4",render_mode="human")
checkpointcallback = CheckpointCallback(save_freq=10000, save_path=log_dir)
model = PPO("CnnPolicy", env, verbose=1, tensorboard_log=log_dir,learning_rate=0.0005, n_steps=4096, gamma=0.99)
model.learn(total_timesteps=100_0000,callback=checkpointcallback)
# model = PPO.load("F:/PycharmProjects/machine_learning_pong/cachefile/rl_model_660000_steps.zip")
# model.set_env(env)
# model.learn(total_timesteps=340000,callback=checkpointcallback)
```

log_dir ="cachefile/"
os.makedirs(log_dir, exist_ok=True)
env = gym.make("PongDeterministic-v4",render_mode="human")
checkpointcallback = CheckpointCallback(save_freq=10000, save_path=log_dir)
model = PPO("CnnPolicy", env, verbose=1, tensorboard_log=log_dir,learning_rate=0.0005, n_steps=4096, gamma=0.99)
model.learn(total_timesteps=100_0000,callback=checkpointcallback)
# model = PPO.load("F:/PycharmProjects/machine_learning_pong/cachefile/rl_model_660000_steps.zip")
# model.set_env(env)
#model.learn(total_timesteps=340000,callback=checkpointcallback)

## 2022-8 paper

(a) (8%) Write a python program that inputs three integers r, c, s (one integer at a line), and then constructs two numpy 2D arrays A and B, both have r rows and c columns, and A contains the integer sequence s, s+1, … starting from left to right, and top to bottom, and B contains the square-root of the elements of A. For example, given the input:

```
import numpy as np
import pandas as pd

r = int(input(""))
c = int(input(""))
s = int(input(""))

aarray = np.arange(s, s+r*c, 1)
aarray = aarray.reshape([r,c])
barray = np.sqrt(aarray)

barray + aarray  # output sum
```

(b) (7%) Give a single Python statement that creates and assigns the following DataFrame to the variable `table`:

```
table
```

| | year | team | wins | draws | losses |
|---|---|---|---|---|---|
| 0 | 2018 | HKU | 30 | 6 | 2 |
| 1 | 2019 | HKU | 28 | 7 | 3 |
| 2 | 2020 | HKU | 32 | 4 | 2 |
| 3 | 2018 | CU | 29 | 5 | 4 |
| 4 | 2019 | CU | 32 | 4 | 2 |
| 5 | 2020 | CU | 26 | 7 | 5 |
| 6 | 2018 | UST | 21 | 8 | 9 |
| 7 | 2019 | UST | 17 | 10 | 11 |
| 8 | 2020 | UST | 19 | 8 | 11 |

a = pd.DataFrame({"year":[2018, 2019, 2020, 2018,2019,2020,2018,2019,2020], "team":["HKU", "HKU", "HKU", "CU","CU","CU", "UST", "UST", "UST"], "wins":[30, 28, 32, 29, 32, 26, 21, 17, 19], "draws":[6,7,4,5,4,7,8,10,8], "losses":[2,3,2,4, 2, 5, 9, 11 , 11]})

(c) (5%) Apply the `groupby` method to `table` to get the data structure `teams_wins`, which groups the wins in terms of teams such that

- `teams_wins.groups` gives

```
{'CU': [3, 4, 5], 'HKU': [0, 1, 2], 'UST': [6, 7, 8]}
```

, and

- `teams_wins.sum()` gives

```
team
CU     87
HKU    90
UST    57
Name: wins, dtype: int64
```

team_wins = a[["team","wins"]].groupby("team")
print(team_wins.groups)
print(team_wins.sum())

(d) (5%) Make use of `teams_wins` to write a single Python statement that gives the team with the largest number of wins, i.e.

```
team
HKU    90
Name: wins, dtype: int64
```

sums = team_wins.sum()
sums.loc[sums.idxmax()]

**Question 2:**

Modify the program to train a new LSTM model to predict the average Heng Seng Index of the following three days.

```
import datetime
import yfinance as yf
from pandas_datareader import data
yf.pdr_override()

# start = datetime.strptime('2004-01-02' ,'%Y-%m-%d')
# end = datetime.strptime( '2009-6-30' , "%Y-%m-%d")
df = data.get_data_yahoo('^HSI',start="2004-01-02", end="2009-6-30")
df = df['Close']
```

```
[**********100%**********]  1 of 1 completed
```

```
from sklearn. preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
rdata = scaler.fit_transform(np.array(df).reshape(-1,1))
training_size=int(len(rdata)*0.65)
train_period, test_period=rdata[:training_size],rdata[training_size: ]

from keras.preprocessing.sequence import TimeseriesGenerator
threeDaysLater = np.zeros([len(train_period)-3, 1])
for i in range(len(train_period)-3):
  threeDaysLater[i][0] = np.sum(train_period[i:i+3])/3
train_period = train_period[:-3]

threeDaysLaterTest = np.zeros([len(test_period)-3, 1])
for i in range(len(test_period)-3):
  threeDaysLaterTest[i][0] = np.sum(test_period[i:i+3])/3
test_period = test_period[:-3]

train = TimeseriesGenerator(train_period, threeDaysLater, length=100, batch_size=1000)
test = TimeseriesGenerator(test_period, threeDaysLaterTest, length=100, batch_size=1000)
X_train, y_train =list(train)[0][0], list(train)[0][1]
X_test, y_test = list(test)[0][0], list(test)[0][1]
```

## 2023-paper

**Q4(**加粗部分更改 Compare the popularity of “Lenovo” and “Dell” by writing a python program：

```
import datetime
import os
import flair
from flair.models import TextClassifier
from flair.data import Sentence

# Load the sentiment analysis model
flair_sentiment_model = TextClassifier.load("en-sentiment")

# Function to get the average sentiment score of a list of sentences
def get_average_sentiment_score(sentences):
    total_score = 0
    for sentence in sentences:

flair_sentiment_model.predict(sentence)
        total_score += sentence.labels[0].score
    if len(sentences) > 0:
        return total_score / len(sentences)
    else:
        return

# Load the Bearer Token from the file
with open('BearerToken.txt', 'r') as file:
```

```
    bearer_token = file.readline().strip()

# Define the date range (May 7, 2023, to May 16, 2023)
start_date = datetime.datetime(2023, 5, 7)
end_date = datetime.datetime(2023, 5, 16)
date_range = [start_date + datetime.timedelta(days=x) for x in range((end_date - start_date).days + 1)]

# Analyze tweets for each day in the date range
for date in date_range:
    formatted_date = date.strftime("%Y-%m-%d")
    query = f'Lenovo OR Dell since:{formatted_date} until:{formatted_date}'
    command = f'curl -X GET "https://api.twitter.com/2/tweets/search/recent?query={query}&tweet.fields=created_at" -H "Authorization: Bearer {bearer_token}"'
    response = os.popen(command).read()
    tweets = response.split('\n')
    sentiment_scores = []
    for tweet in tweets:
        if tweet != "":
            sentence = Sentence(tweet)
            sentiment_scores.append(sentence)
    average_sentiment_score = get_average_sentiment_score(sentiment_scores)
    print(f"Date: {formatted_date}, Average Sentiment Score: {average_sentiment_score}")
```

## Q-learning for cart-pole:

```
import gym
import numpy as np
# 创建CartPole环境
env = gym.make('CartPole-v0')
# 设置随机种子
np.random.seed(0)
# 初始化Q表
Q = np.zeros((env.observation_space.n, env.action_space.n))
# 设置超参数
num_episodes = 1000
max_steps = 200
learning_rate = 0.1
discount_factor = 0.99
exploration_rate = 0.1

# Q-learning算法
for episode in range(num_episodes):
    state = env.reset()
    total_reward = 0
    for step in range(max_steps):
        # 选择动作
        if np.random.uniform(0, 1) < exploration_rate:
            action = env.action_space.sample()  # 随机选择动作
        else:
            action = np.argmax(Q[state, :])  # 根据Q值选择动作
        # 执行动作
        next_state, reward, done, _ = env.step(action)
        # 更新Q值
        Q[state, action] += learning_rate * (reward + discount_factor * np.max(Q[next_state, :]) - Q[state, action])
        total_reward += reward
        state = next_state
        if done:
            break
    # 输出每个回合的总奖励
    print("Episode:", episode, "Total Reward:", total_reward)
# 使用学习到的Q表玩游戏
state = env.reset()
done = False
total_reward = 0
while not done:
    action = np.argmax(Q[state, :])
    state, reward, done, _ = env.step(action)
    total_reward += reward
# 输出最终的总奖励
print("Final Total Reward:", total_reward)
```

2023-程序设计（lec2 ※）

Design a system that applies machine learning techniques to predict fraud transactions based on the file tranrecords.cvs. You should describe all the steps needed to implement such a system.

Data cleaning: Read the "tranrecords.cvs" file and handle missing values. Depending on the given data format, handle missing entries using an appropriate method (such as filling in the mean, median, or removing missing values).

Exploratory data analysis (EDA): Analyze data and explore the distribution and correlation of different feature columns. Use visualization tools and statistical methods to understand the characteristics and structure of your data.

Feature engineering: Based on the understanding of the data and the results of EDA, select feature columns that are useful for predicting fraudulent transactions. Feature extraction, transformation, or combination may be required to improve model performance.

Model selection and training: Select an appropriate machine learning model for fraudulent transaction prediction. Common models include logistic regression, support vector machines, decision trees, random forests, neural networks, etc. Based on the characteristics of the data and the needs of the problem, select the most appropriate model and use the training data to train the model.

Data preparation: Divide the data into training set and test set. Typically a large portion of the data is used for training and a small portion is used to evaluate the model's performance. Ensure that the partitioning of data is random and maintain the distribution characteristics of the data.

Model training and testing: Use the training set to train the selected model, and use the test set to evaluate the model. Tune and improve the model based on the model's performance indicators (such as accuracy, precision, recall, etc.).

System performance evaluation: Evaluate the overall performance of the system, including model accuracy and efficiency. Methods such as cross-validation, confusion matrix, and ROC curve can be used to evaluate the performance of the model.

Results Interpretation and Deployment: Interpret the model's predictions and make further interpretations and adjustments as needed. Finally, the trained model is deployed into practical applications for real-time prediction and monitoring of fraudulent transactions.

Some ideas to solve overfitting:

**Increase training data**
**Cross-validation**
**Regularization**
**Feature selection**
**Early stopping**
**Ensemble methods**
**Simpler models**
**Data preprocessing**
**Hyperparameter tuning**

```
    n = len(nums)
    max_ending_here = min_ending_here = max_so_far = nums[0]
    # To store the start and end indices of the maximum product subsequence
    start = end = s = 0
    result = [0] * n
    result[0] = nums[0]
    for i in range(1, n):
        # Temporary variables to hold the previous values
        temp_max = max_ending_here
        temp_min = min_ending_here
        # Update the max and min products for the current number
        max_ending_here = max(nums[i], nums[i] * temp_max, nums[i] * temp_min)
        min_ending_here = min(nums[i], nums[i] * temp_max, nums[i] * temp_min)
        # Reset the starting index of the maximum product subsequence
        if max_ending_here == nums[i]:
            s = i
        # If the current max product is greater than the max
        #so far, update the start and end indices
        if max_ending_here > max_so_far:
            max_so_far = max_ending_here
            start = s
            end = i
        result[i] = max_ending_here
    # Extract the maximum product subsequence
    max_product_subsequence = nums[start:end+1]
    return result, max_product_subsequence
# Define the input array (assuming the image has given this array)
input_array = [0.5, 1.5, 30, 10, 5, 0.4, 10]
# Call the function and print the result
max_products, max_product_subsequence = max_subarray_product(input_array)
max_products, max_product_subsequence
```

bellman考题