# Curiosity project - Group 06 - Heart Disease

**Submitters:**
Tal Carmi, ID: 039161203
Anna Mosenzon, ID: 200320836
Rami Skolozub, ID: 316736396

Problem description:
A classification problem to distinguish between presence or absence of heart disease based on dataset contains 13 features and 1 Label column (there is a heart disease or there is no heart disease.
The target variable: Heart Disease – Yes\No distributes as follows:
54% of the observations have heart disease and 46% have no heart disease.

Part III:

8. In this part we are using Reinforcement learning in order to apply curious feature selection and calculate the Q-matrix which will provide us the best policy of feature selection for our data set.
Although our data set is relatively small (13 features and 303 observations) applying exhaustive search algorithm on bigger data sets (~>10-15 features) in order to find the optimal set of features, it would be costly in a computation time aspect and will not be efficient - $O(N_{data} \cdot 2^{N_{features}})$. Another motivation to use curious feature selection over using exhaustive search algorithm is to reduce model overfitting and increase generalization.
Here we apply RL model (published by Moran & Gordon, 2019), the model is based on learning using curious trial and error given the set of features our learner was provided with. The learner will calculate the reward based on our reward definition (section 8.b). The complexity of the model is $O(N_{data} \cdot N_{features})$. We don't know what our best set of features and our algorithm will find it. The algorithm we use maximizes the future accumulated reward signal in the long run. Our curiosity loop includes an agent, learner, and reward. In each loop our learner selects an action (will be explained later based on what algorithm we defined to select the action). We used the NN from the second assignment to predict the best next step (next feature to insert). The reward impacted by the learner prediction error and the actor will select the action that will maximize its learning (where the error is the highest).

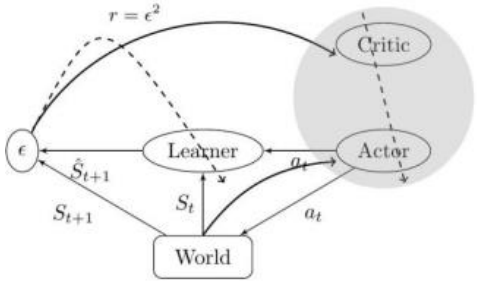



**Figure 12:** The curiosity loop - Forward model.

**Problem formulation as curiosity problem:**

a. The States:
Our states describe the system possible states and has all the information we need to know on the "world", since it's a Markov process, we don't have any other memory unless we save it into our Q function. **The possible States are any subset of the available features** (13 feature columns in the "zero state") where each feature

can be selected only once: ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']. Details about each column feature can be found in the appendix.

**In each iteration the state changes and the algorithm can select features only from the available set of features list.**

For example, ['age', 'sex', 'fbs','age'] is not feasible outcome from our state definition since age feature is selected twice.

In the beginning of the algorithm our state space includes all our feature list and for the next iteration the next state will be selected by the algorithm only from the available features that haven't been selected yet. Each time the algorithm eliminates from the available features the feature that was selected in the previous step.

```
for iter in range(iterations):
  print(f'started iteration #{iter} of {iterations} iterations')
  x_train_episode, y_train_episode, x_validation_episode, y_validation_episode=chooseEpisodeData(x_train,y_train,80,20)
  last_accuracy=0.5
  current_state='zero'
  current_features=[]
  available_features=features_list[:]
  exploration_p=getExplorationP(iter, iterations)
  #print(f'exploration_p: {exploration_p}')
  alpha=getAlpha(iter, iterations)
  while len(current_features)<len(features_list):
    next_state=''
    if random.random()<0.9:#exploration_p:
      #print(f'exploration **************************************************')
      next_state=available_features.pop(random.randint(0,len(available_features)-1))
    else:
      next_state_index=get_best_next_feature(current_state, Q_arr, available_features, feature_to_Index_dict)
      next_best_reward=Q_arr[feature_to_Index_dict[current_state]][next_state_index]
      if next_best_reward<0:
        break
      next_state=available_features.pop(next_state_index)

    current_features.append(next_state)
    #print(f'current_features: {current_features}')
    #print(f'available_features: {available_features}')
```

Getting to some state is dependent on the previous state and the action the actor performed.

We initialized our Q-matrix with "0" for all states.

The Actions:

**The possible actions in each step is adding a specific feature or not adding.**

**Each action the algorithm does, changes the available feature state space.**

The set of actions the algorithm can perform is determined by the states space which we defined above.

The algorithm decide which next feature will give the highest positive gain if all the features will give negative reward, the algorithm will decide to stop adding features.

Each action of adding a new specific feature, changes the new state.

We created a function that find the next best feature given the current state :

```
def get_best_next_feature(current_state, Q_arr, available_features, fea
ture_to_Index_dict):
  max_index=-1
  maxQ=-1000
  for i in range(len(available_features)):
    next_feature=available_features[i]
```

```
    if next_feature!=current_state:
      next_feature_Q_score=Q_arr[feature_to_Index_dict[current_state]][
feature_to_Index_dict[next_feature]]
      if maxQ<next_feature_Q_score:
        if maxQ<Q_arr[feature_to_Index_dict[current_state]][feature_to_
Index_dict[next_feature]]:
          max_index=i
          maxQ=Q_arr[feature_to_Index_dict[current_state]][feature_to_I
ndex_dict[next_feature]]
  return max_index
```

b. Reward definition:
   The reward as the benefit the algorithm will get from a given action it will perform**.** In our case, we defined it **as the delta in the prediction accuracy given the action that was done** (adding specific feature in the state we were), meaning, the difference between the validation accuracy at the current step comparing to the validation accuracy at the previous step:
   **Reward** = current_validation_accuracy-last_validation_accuracy
c. Closing the "curiosity loop":
   Our learner (we used our NN model from the second assignment) tries to learn which action should the algorithm do so that it will learn the most.
   We defined in our actor a trade-off of exploration-exploitation.
   The parameter we used to determine that is epsilon. The epsilon is defined according to how long we are training the model, the higher the number of iterations is – the smaller the epsilon, so that in the beginning of the process we would want to perform exploration and the longer the process progresses epsilon gets smaller and more exploration will be performed.

```
 def getExplorationP(i, iterations):
    if i<iterations/4:
      return 0.1
    if i<iterations/2:
      return 0.05
    if i<iterations*0.75:
      return 0.01
    return 0.005
```

a.
   In addition, we defined an alpha parameter which defines the learning rate of the model. Alpha is defined according to how long we are training the model, the higher the number of iterations is – the smaller the alpha, so that in the beginning of the process we would want to give more weight for the learning and as the process progresses alpha gets smaller and updates will become smaller.

```
def getAlpha(i, iterations):
  if i<iterations/4:
    return 0.9
  if i<iterations/2:
    return 0.5
  if i<iterations*0.75:
    return 0.3
```

```
  return 0.1
```

In our case, our process is a Markov process (all required information exist in the current state only) so we are getting an optimal policy that solves our problem and this policy is deterministic. Given the state and action the algorithm performs and

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

the reward it receives from this action from now on, we compute the Q-matrix for set of state and action. The Q-matrix update steps computation is done using the Bellman equation:

```
def calc_updated_Q(current_state, next_state, Q_arr, reward, alpha, dis
count_factor, features_list, feature_to_Index_dict):
  current_state_index = feature_to_Index_dict[current_state]
  next_state_index = feature_to_Index_dict[next_state]

  current_state_next_state_Q=Q_arr[current_state_index][next_state_inde
x]
  best_next_state_Q_index=get_best_next_feature(current_state, Q_arr, f
eatures_list, feature_to_Index_dict)
  Q_arr[current_state_index][next_state_index]=current_state_next_state
_Q+alpha*(reward+discount_factor*Q_arr[current_state_index][best_next_s
tate_Q_index]-current_state_next_state_Q)
```

Once we got the final Q-matrix, we will follow the optimal path - optimal feature selection order which is the optimal policy - $\pi$.
The optimal policy is constructed by selecting the feature (action) with the highest value, setting this feature as the next state and repeat this operation till there aren't any available features left.
Since our process is finite, the algorithm finds the optimal policy.
we coded a function called "find_best_path" that executes this optimal policy and results in the best accuracy set given the Q-matrix that was calculated in the previous step:

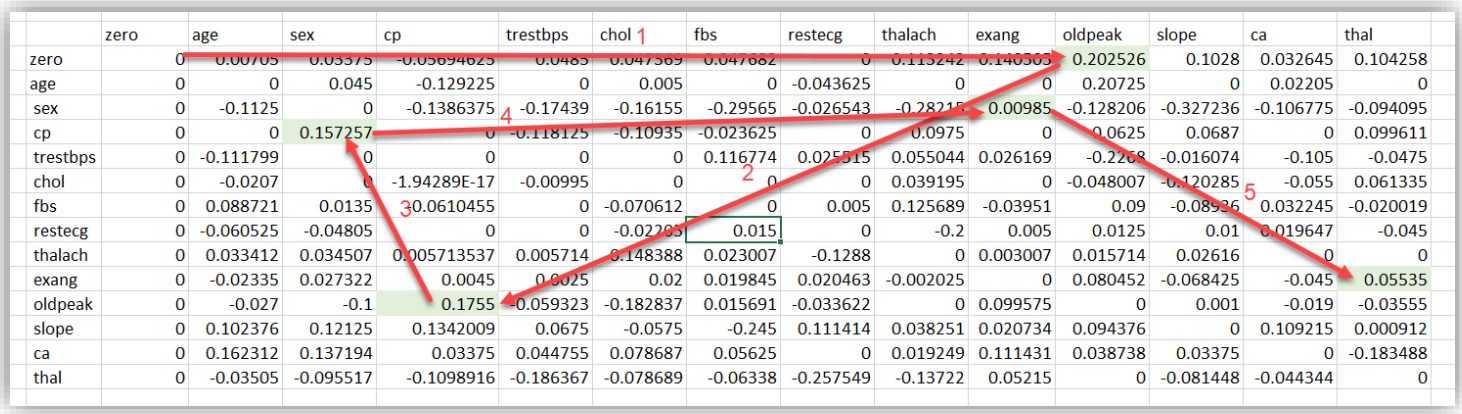
```
def find_best_path(Q_arr,features_list,feature_to_Index_dict):
  available_features=features_list[:]
  best_features=[]
  current_state='zero'
  current_state_index=feature_to_Index_dict[current_state]
  next_state=-1
  counter=0
  while next_state==-1 or len(best_features)<=len(features_list):
    next_state_features_arr_index=get_best_next_feature(current_state,
Q_arr, available_features, feature_to_Index_dict)
    next_state=available_features[next_state_features_arr_index]
    next_state_index=feature_to_Index_dict[next_state]
```

```
    if Q_arr[current_state_index][next_state_index]>0:
      best_features.append(available_features.pop(next_state_features_a
rr_index))
      current_state=next_state
      current_state_index=feature_to_Index_dict[current_state]
    else:
      break
    counter+=1
    if counter>20:
      break

  return best_features
```

9. Computing the solution:
   The detailed results can be found in the appendix.

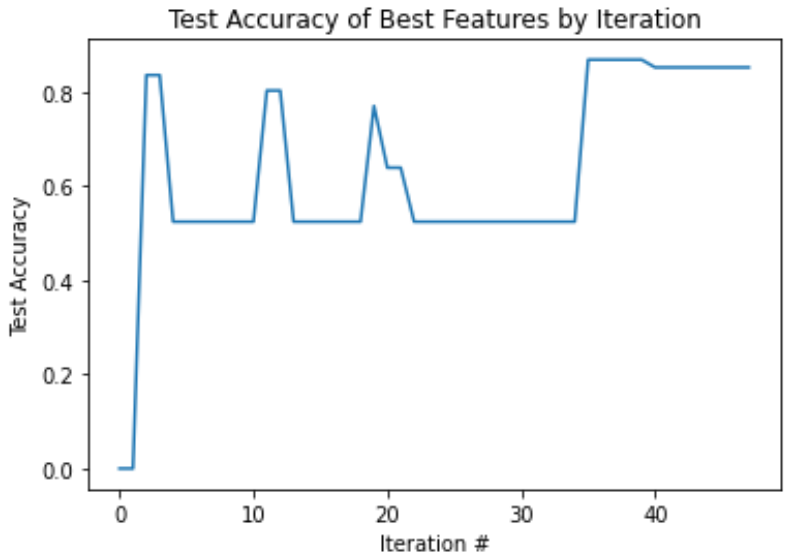
   The final path in the Q-matrix is:



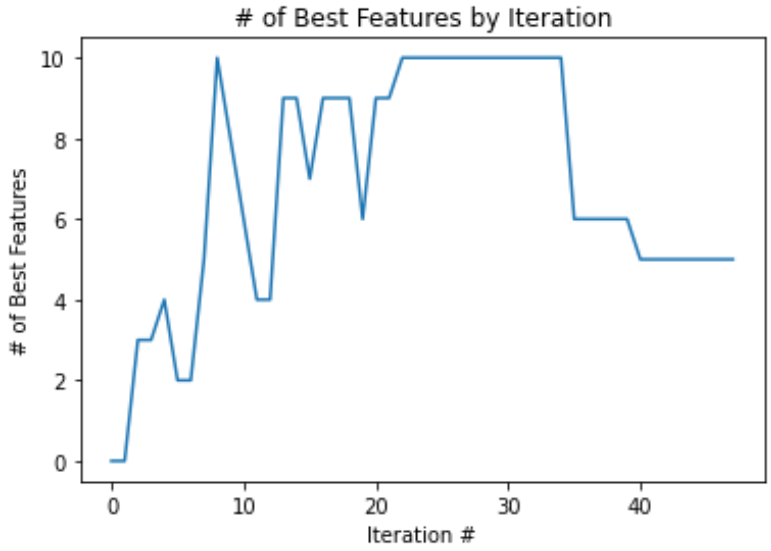
| | zero | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| zero | 0 | 0.00705 | 0.03375 | -0.05894625 | 0.0483 | 0.047369 | 0.047682 | 0 | 0.113242 | 0.140303 | 0.202526 | 0.1028 | 0.032645 | 0.104258 |
| age | 0 | 0 | 0.045 | -0.129225 | 0 | 0.005 | 0 | -0.043625 | 0 | 0 | 0.20725 | 0 | 0.02205 | 0 |
| sex | 0 | -0.1125 | 0 | -0.1386375 | -0.17439 | -0.16155 | -0.29565 | -0.026543 | -0.28215 | 0.00985 | -0.128206 | -0.327236 | -0.106775 | -0.094095 |
| cp | 0 | 0 | 0.157257 | 0 | -0.118125 | -0.10935 | -0.023625 | 0 | 0.0975 | 0 | 0.0625 | 0.0687 | 0 | 0.099611 |
| trestbps | 0 | -0.111799 | 0 | 0 | 0 | 0 | 0.116774 | 0.025515 | 0.055044 | 0.026169 | -0.2268 | -0.016074 | -0.105 | -0.0475 |
| chol | 0 | -0.0207 | 0 | -1.94289E-17 | -0.00995 | 0 | 0 | 0 | 0.039195 | 0 | -0.048007 | -0.120285 | -0.055 | 0.061335 |
| fbs | 0 | 0.088721 | 0.0135 | 0.0610455 | 0 | -0.070612 | 0 | 0.005 | 0.125689 | -0.03951 | 0.09 | -0.08936 | 0.032245 | -0.020019 |
| restecg | 0 | -0.060525 | -0.04805 | 0 | 0 | -0.02265 | 0.015 | 0 | -0.2 | 0.005 | 0.0125 | 0.01 | 0.019647 | -0.045 |
| thalach | 0 | 0.033412 | 0.034507 | 0.005713537 | 0.005714 | 0.148388 | 0.023007 | -0.1288 | 0 | 0.003007 | 0.015714 | 0.02616 | 0 | 0 |
| exang | 0 | -0.02335 | 0.027322 | 0.0045 | 0.0025 | 0.02 | 0.019845 | 0.020463 | -0.002025 | 0 | 0.080452 | -0.068425 | -0.045 | 0.05535 |
| oldpeak | 0 | -0.027 | -0.1 | 0.1755 | -0.059323 | -0.182837 | 0.015691 | -0.033622 | 0 | 0.099575 | 0 | 0.001 | -0.019 | -0.03555 |
| slope | 0 | 0.102376 | 0.12125 | 0.1342009 | 0.0675 | -0.0575 | -0.245 | 0.111414 | 0.038251 | 0.020734 | 0.094376 | 0 | 0.109215 | 0.000912 |
| ca | 0 | 0.162312 | 0.137194 | 0.03375 | 0.044755 | 0.078687 | 0.05625 | 0 | 0.019249 | 0.111431 | 0.038738 | 0.03375 | 0 | -0.183488 |
| thal | 0 | -0.03505 | -0.095517 | -0.1098916 | -0.186367 | -0.078689 | -0.06338 | -0.257549 | -0.13722 | 0.05215 | 0 | -0.081448 | -0.044344 | 0 |



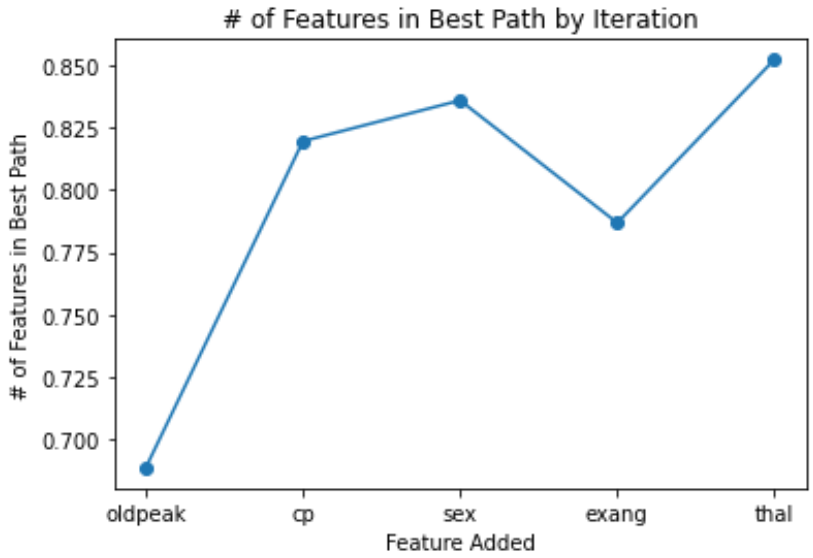
which is the following set of features: ['oldpeak', 'cp', 'sex', 'exang', 'thal'], only 5 features selected out of 13 providing test accuracy of: 0.8524590163934426. comparing to random 5 feature selection (the same number as the algorithm selected) we will get test accuracy of: 0.5245901639344263.
The results can be represented in the following graph:




We can see that the algorithm gets better accuracy, after that the accuracy is deteriorating and the low accuracy stays the same for a few iterations, and then gets better again. It can be explained by the Q matrix that according to the values that are given in the Q matrix at that point there is no motivation to select different set of features so the accuracy stays exactly the same up until the Q matrix changes significantly and then we can see a change in the accuracy.




Another result we can see is the number of features selected in each iteration. The algorithm tries to add features but when the process progress the number of features decreases so that eventually it will get to 5 features only that gives the best Q value.




each addition of the features provided better test accuracy for the first three features, the fourth feature (exang) resulted in a decrease in the test accuracy but it is part of the optimum path since adding the fifth feature (thal) results in a higher accuracy than before the selection of the last two features. This result demonstrates the strength of the curious algorithm.

In our results, we noticed that running the algorithm several times on the same data-set did not produce the same converged policy, we assume that it is related to the fact that our dataset is very small and to the randomization affect this algorithm have.

10. Bonus:
    a. SN/VTA, Nucleus Accumbens, Prefrontal cortex and hippocampus are the areas in the brain that can solve this problem. In addition, recognition of information gap is related to the Anterior Cingulate cortex. Those areas have functional connectivity with DMN in support curiosity-enhanced memory and are responsible for investigation and error prediction which are required for solving the problem.
    b. A possible learning will be, first try to guess based on value of 1 feature, that is based on the error update the prior believe (which is no longer 50-50%).
    then add a second feature and make the same process. The learning will be based on the error.
    *If we don't have any features, just the output, a person would use his 5 senses to try to determine if a patient have a heart disease or not.

# Appendix:

## Details about the feature in the data set:

| Column Name | Description | Comment | Data Type |
|---|---|---|---|
| age | Age in years | | numeric |
| sex | The person's sex | 1 = male, 0 = female | categorical |
| cp | The chest pain experienced (4 values) | Value 0: typical angina, Value 1: atypical angina, Value 2: non-anginal pain, Value 3: asymptomatic | categorical |
| trestbps | The person's resting blood pressure | mm Hg on admission to the hospital | numeric |
| chol | The person's cholesterol measurement in mg/dl | | numeric |
| fbs | The person's fasting blood sugar | > 120 mg/dl, 1 = true; 0 = false | categorical |
| restecg | Resting electrocardiographic measurement | 0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria | categorical |
| thalach | The person's maximum heart rate achieved | | numeric |
| exang | Exercise induced angina | 1 = yes; 0 = no | categorical |
| oldpeak | ST depression induced by exercise relative to rest | ST' relates to positions on the ECG plot | Float |
| slope | the slope of the peak exercise ST segment | Value 0: upsloping, Value 1: flat, Value 2: down sloping | categorical |
| ca | number of major vessels colored by fluoroscopy | | numeric |
| thal | A blood disorder called thalassemia | 0 = null; 1 = fixed defect; 2 = normal; 3 = reversable defect | categorical |
| **target** | **Heart disease** | **0 = no, 1 = yes** | **categorical** |

## Curious Feature Selection pseudo code (Moran & Gordon, 2019):

**Algorithm**: Curious Feature Selection (CFS)

**input:** set of possible features: $F$, $|F| = N_{features}$, dataset: $\vec{x_i}$ with $N_{data}$ samples

**output:** set of selected features $F_{sel}$

**initialize:**

$Q \leftarrow \mathbb{1}_{n \times n}$ ,$n = |F| + 1$

$N_{epi} = 100$

$itr = 10 \times N_{data}/N_{epi}$

$\gamma = 0.01$

$learner = DT$

**for** i = 1 to $itr$ **do**

$x_{epi} \leftarrow x_{j,k}$, $j = sample(\vec{x_i}, n = N_{epi})$, $k \in F$

$x^{epi\_train}, x^{epi\_validation} = split(x_{epi}, validation\_size = 0.2)$

$e_0 \leftarrow 0.5$

$s_t \leftarrow s_0$

$F_{sel} \leftarrow \{\}$

$F_{available} \leftarrow F$

$$\epsilon = \begin{cases} 0.09 & i < itr/4 \\ 0.05 & itr/4 \leq i < itr/2 \\ 0.01 & itr/2 \leq i < itr * 3/4 \\ 0.005 & i \geq itr * 3/4 \end{cases}$$

$$\alpha = \begin{cases} 0.9 & i < itr/4 \\ 0.5 & itr/4 \leq i < itr/2 \\ 0.3 & itr/2 \leq i < itr * 3/4 \\ 0.1 & i \geq itr * 3/4 \end{cases}$$

**while** $|F_{available}| > 0$ **do**

**if** $p \sim U(0,1) < epislon$ **then**

$a_t = random(a \in F_{available})$

**else**

**if** $max(Q(s_t, a_t)) < threshold$ **then** end episode

**else** $a_t = max(Q(s_t, a_t))$

$F_{sel} \leftarrow F_{sel} \cup a_t$

$x^{epi\_train_{s_t}} = x^{epi\_train}_{k \in F_{sel}}$

$x^{epi\_validation_{s_t}} = x^{epi\_validation}_{k \in F_{sel}}$

$O_{learner} = learner(x^{epi\_train_{s_t}})$

$e_t = \sum_i \left( O_{learner}\left(x_i^{epi\_validation_{s_t}}\right) - O_{real}\left(x_i^{epi\_validation_{s_t}}\right)\right)$

$r_t \leftarrow (e_0 - e_t)$

$Q(s_t, a) = Q(s_t, a) + \alpha \left[r + \gamma * \max_a Q(s_{t+1}, a) - Q(s_t, a)\right]$

$s_{t+1} \leftarrow a_t$

$e_0 \leftarrow e_t$

$F_{available} \leftarrow F_{available} \setminus a_t$

$t \leftarrow t + 1$

| Parameter | Description | Value |
|---|---|---|
| Episode size | Number of records in each episode | 100 |
| Iterations | Total numbers of episodes | 10 · data size ÷ episode size |
| Learner | The supervised machine learning model which is used to define the internal reward | Decision Tree, Naive Bayes |
| Initial error | The error which associate with the initial state (i.e. prior error) | 0.5 |
| Discount factor $\gamma$ | The discount factor determines the importance of future rewards in the Q-Learning algorithm. A factor of 0 makes the agent "myopic" by only considering current rewards | 0-0.01 |
| Epsilon $\epsilon$ | The probability to select a random action (from available actions) in exploration mode | 0.9 to 0.1 with step decay policy |

## Detailed results for the algorithm:

```
started iteration #0 of 48 iterations
best features: []
```

```
started iteration #1 of 48 iterations
best features: []
started iteration #2 of 48 iterations
best features: ['fbs', 'sex', 'cp']
test accuracy best features: 0.8360655737704918
started iteration #3 of 48 iterations
best features: ['fbs', 'sex', 'cp']
test accuracy best features: 0.8360655737704918
started iteration #4 of 48 iterations
best features: ['fbs', 'cp', 'thal', 'thalach']
test accuracy best features: 0.5245901639344263
started iteration #5 of 48 iterations
best features: ['exang', 'oldpeak']
test accuracy best features: 0.5245901639344263
started iteration #6 of 48 iterations
best features: ['exang', 'oldpeak']
test accuracy best features: 0.5245901639344263
started iteration #7 of 48 iterations
best features: ['cp', 'thal', 'thalach', 'age', 'oldpeak']
test accuracy best features: 0.5245901639344263
started iteration #8 of 48 iterations
best features: ['cp', 'thal', 'thalach', 'age', 'oldpeak', 'trestbps',
'fbs', 'sex', 'slope', 'chol']
test accuracy best features: 0.5245901639344263
started iteration #9 of 48 iterations
best features: ['exang', 'oldpeak', 'trestbps', 'fbs', 'cp', 'thal',
'thalach', 'age']
test accuracy best features: 0.5245901639344263
started iteration #10 of 48 iterations
best features: ['exang', 'trestbps', 'fbs', 'cp', 'thal', 'ca']
test accuracy best features: 0.5245901639344263
started iteration #11 of 48 iterations
best features: ['cp', 'sex', 'restecg', 'ca']
test accuracy best features: 0.8032786885245902
started iteration #12 of 48 iterations
best features: ['cp', 'sex', 'restecg', 'ca']
test accuracy best features: 0.8032786885245902
started iteration #13 of 48 iterations
best features: ['exang', 'trestbps', 'thalach', 'slope', 'chol',
'thal', 'ca', 'sex', 'restecg']
test accuracy best features: 0.5245901639344263
started iteration #14 of 48 iterations
best features: ['exang', 'trestbps', 'thalach', 'slope', 'chol',
'thal', 'ca', 'sex', 'restecg']
test accuracy best features: 0.5245901639344263
started iteration #15 of 48 iterations
best features: ['exang', 'trestbps', 'thalach', 'slope', 'ca', 'sex',
'restecg']
test accuracy best features: 0.5245901639344263
started iteration #16 of 48 iterations
best features: ['exang', 'trestbps', 'thalach', 'age', 'oldpeak',
'restecg', 'ca', 'sex', 'thal']
test accuracy best features: 0.5245901639344263
started iteration #17 of 48 iterations
best features: ['oldpeak', 'restecg', 'ca', 'sex', 'thal', 'exang',
'trestbps', 'fbs', 'cp']
test accuracy best features: 0.5245901639344263
started iteration #18 of 48 iterations
```

```
best features: ['oldpeak', 'restecg', 'ca', 'sex', 'thal', 'exang',
'trestbps', 'fbs', 'cp']
test accuracy best features: 0.5245901639344263
started iteration #19 of 48 iterations
best features: ['oldpeak', 'exang', 'slope', 'ca', 'sex', 'restecg']
test accuracy best features: 0.7704918032786885
started iteration #20 of 48 iterations
best features: ['thalach', 'slope', 'ca', 'sex', 'restecg', 'oldpeak',
'exang', 'fbs', 'age']
test accuracy best features: 0.639344262295082
started iteration #21 of 48 iterations
best features: ['thalach', 'slope', 'ca', 'sex', 'restecg', 'oldpeak',
'exang', 'fbs', 'age']
test accuracy best features: 0.639344262295082
started iteration #22 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #23 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #24 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #25 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #26 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #27 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #28 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #29 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #30 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #31 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #32 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
```

```
started iteration #33 of 48 iterations
best features: ['thalach', 'slope', 'cp', 'sex', 'restecg', 'ca',
'age', 'oldpeak', 'exang', 'thal']
test accuracy best features: 0.5245901639344263
started iteration #34 of 48 iterations
best features: ['thalach', 'sex', 'restecg', 'ca', 'age', 'oldpeak',
'cp', 'thal', 'exang', 'fbs']
test accuracy best features: 0.5245901639344263
started iteration #35 of 48 iterations
best features: ['thalach', 'sex', 'exang', 'oldpeak', 'cp', 'thal']
test accuracy best features: 0.8688524590163934
started iteration #36 of 48 iterations
best features: ['thalach', 'sex', 'exang', 'oldpeak', 'cp', 'thal']
test accuracy best features: 0.8688524590163934
started iteration #37 of 48 iterations
best features: ['thalach', 'sex', 'exang', 'oldpeak', 'cp', 'thal']
test accuracy best features: 0.8688524590163934
started iteration #38 of 48 iterations
best features: ['thalach', 'sex', 'exang', 'oldpeak', 'cp', 'thal']
test accuracy best features: 0.8688524590163934
started iteration #39 of 48 iterations
best features: ['thalach', 'sex', 'exang', 'oldpeak', 'cp', 'thal']
test accuracy best features: 0.8688524590163934
started iteration #40 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
started iteration #41 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
started iteration #42 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
started iteration #43 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
started iteration #44 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
started iteration #45 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
started iteration #46 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
started iteration #47 of 48 iterations
best features: ['oldpeak', 'cp', 'sex', 'exang', 'thal']
test accuracy best features: 0.8524590163934426
```