Project B (OpenAI Gym agent) Second Report

----- Machine Learning Agent with binary classification

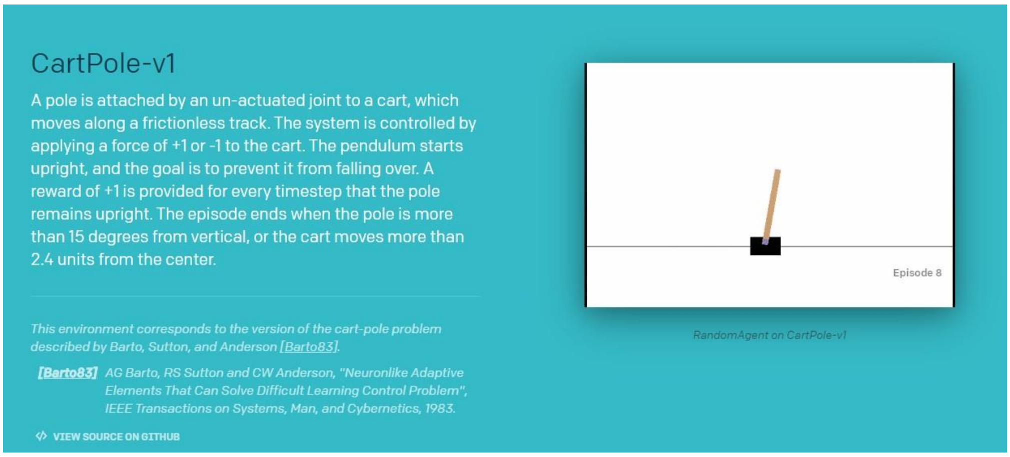
2017201980

李平山

11/15

Introductions:

Using 'CartPole-v1' environment in gym, game introduction:



[Retrieved from https://gym.openai.com/envs/CartPole-v1/]

Test Environment:

- windows 10, python 3,

Experimental Procedure:

1) Test data collection

From the first Report, we can get the variables in the CartPole-v1:

```
Observation:
        Type: Box(4)
        Num  Observation              Min         Max
        0    Cart Position           -4.8         4.8
        1    Cart Velocity           -Inf         Inf
        2    Pole Angle               -24 deg     24 deg
        3    Pole Velocity At Tip    -Inf         Inf

    Actions:
        Type: Discrete(2)
        Num  Action
        0    Push cart to the left
        1    Push cart to the right
```

For every decision making of action, observation variable is and is the only that matters. We use a well-trained agent in OpenAI platform(obtained from https://gym.openai.com/envs/CartPole-v0/) to be the well-performed example to generator data.

And use following program to generator 50,000 data

```python
In [ ]: #!/usr/bin/env python
# coding: utf-8


import gym
import time
import numpy as np
tmp = 0
env = gym.make('CartPole-v1')

f = open("sample.out","w")

#arr = [0.32455586, -0.09436489,  1.42703162,  1.14888277, -0.0177973]
#arr = [0.19566202, 0.11578184, 0.7173747,  1.48423667, 0.05098461]
#arr = [ 1.92704091e-01,  3.80987661e-01,  1.32745303e+00,  2.07162982e+00, -9.27898585e-04]
arr = [0.01159834, 0.26770383, 1.31941917, 1.93764616, 0.00291291]
def next_move(observation):
    if observation.dot(arr[0:4]) + arr[4] > 0:
        return 1
    else :
        return 0


for i_episode in range(20):
    observation = env.reset()
    action = 0
    #env.step(action)
    for t in range(501):
        #env.render()
        observation, reward, done, info = env.step(action)
        action = next_move(observation)
        f.write(str(observation[0]) + " " + str(observation[1]) + " " + str(observation[2]) + " " + str(observation[3]) + " " + str(action) +
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            tmp += t+1
            break
print("Average timesteps: {} ".format((tmp)/10))
env.close()
f.close()
```

Test data is saved in sample.out in formation

Observation, action

```
0.25994067578481495 0.14588215580153507 -0.0018933086888268738 -0.05474182539671418 0
0.26285831890084566 -0.049212597029636984 -0.0029881451967611575 0.23734314692314656 1
0.2618740669602529 0.14595191657922493 0.0017587177417017741 -0.056280829901917684 0
0.2647931052918374 -0.0491952075731692 0.0006331011436634204 0.2369564663708621 1
0.26380920114037404 0.14591769301017257 0.005372230471080662 -0.05552669412691552 0
0.2667275550005775 -0.04928087400347489 0.004261696588542352 0.23884635936599335 1
0.265741937520508 0.14577993695858554 0.009038623775862219 -0.052489265047447387 0
0.2686575362596797 -0.049470442860712394 0.007988838474913272 0.24303166048365643 1
```

2) Binary Classification and Cross-validation on test data

For binary classification, we are interested in classifying data into 0's and 1's in our data as action described in CartPole environment, using observation as features in classification.

In this matter, we use different methods on test data:

- SVM Classifier
- Logistic Regression Classifier
- RandomForest
- voting_classify with methods above

Before training, we set the environment using sklearn:

```
]: # -*- coding: utf-8 -*-
import gym
import pandas as pd
import matplotlib
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
import numpy as np

def load_csv_data(filename):     #读取文件 格式为每行 (observation[0],.[1],.[2],.[3],action)
    data = []
    labels = []
    datafile = open(filename)
    for line in datafile:
        fields = line.strip().split(' ')
        data.append([float(field) for field in fields[:-1]])
        labels.append(fields[-1])
    data = np.array(data)
    labels = np.array(labels)
    return data, labels


#load data
X, y = load_csv_data('sample.out')
```

We use train_test_split for Cross-validation on test data:
(Split arrays or matrices into random train and test subsets)

```
X_train,X_test,y_train,y_test = train_test_split(X,y)
```

3) SVM Classifier and its performance:

Support Vector Machines (SVMs) are a type of classification algorithm that are more flexible - they can do linear classification, but can use other non-linear *basis functions*. The following uses a linear classifier to fit the observation-action pattern that separates the data into 0's and 1's:

```
In [2]: ### SVM Classifier
print("==========================================")
from sklearn.svm import SVC
clf1 = SVC(gamma='auto',kernel='rbf', probability=True)
clf1.fit(X_train,y_train)
predictions = clf1.predict(X_test)
print("SVM")
print(classification_report(y_test,predictions))
print("AC",accuracy_score(y_test,predictions))
```

```
==========================================
SVM
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      3206
           1       0.91      0.92      0.91      3030

   micro avg       0.92      0.92      0.92      6236
   macro avg       0.92      0.92      0.92      6236
weighted avg       0.92      0.92      0.92      6236

AC 0.9161321359846055
```

Accuracy_score for SVM classifier:

0.9161321359846055

4) Logistic Regression Classifier and its performance:`

Logistic Regression is a type of Generalized Linear Model (GLM) that uses a logistic function to model a binary variable based on any kind of independent variables.

In [3]:
```
### Logistic Regression Classifier!
print("==========================================")
from sklearn.linear_model import LogisticRegression
clf2 = LogisticRegression(solver = "lbfgs",penalty='l2')
clf2.fit(X_train,y_train)
predictions = clf2.predict(X_test)
print("LR")
print(classification_report(y_test,predictions))
print("AC",accuracy_score(y_test,predictions))
```

```
==========================================
LR
              precision    recall  f1-score   support

           0       0.95      0.92      0.93      3206
           1       0.92      0.94      0.93      3030

   micro avg       0.93      0.93      0.93      6236
   macro avg       0.93      0.93      0.93      6236
weighted avg       0.93      0.93      0.93      6236

AC 0.9331302116741501
```

Accuracy_score for Logistic Regression classifier:

0.9331302116741501

5) RandomForest and its performance:

Random Forests are an ensemble learning method that fit multiple Decision Trees on subsets of the data and average the results. We can again fit them using sklearn, and use them to predict outcomes, as well as get mean prediction accuracy

```
In [4]: ### RandomForest!
        print("==========================================")
        RF = RandomForestClassifier(n_estimators=10,random_state=11)
        RF.fit(X_train,y_train)
        predictions = RF.predict(X_test)
        print("RF")
        print(classification_report(y_test,predictions))
        print("AC",accuracy_score(y_test,predictions))
```

```
==========================================
RF
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      3206
           1       0.98      0.98      0.98      3030

   micro avg       0.98      0.98      0.98      6236
   macro avg       0.98      0.98      0.98      6236
weighted avg       0.98      0.98      0.98      6236

AC 0.9830019243104554
```

Accuracy_score for RandomForest:

0.9830019243104554

6) Voting Classifier and its performance:`

In this matter, we combines the predictions from multiple machine learning algorithms (3 classifier discussed above). Mark that Voting classifier isn't an actual classifier but a wrapper for set of different ones that are trained and valuated in parallel in order to exploit the different peculiarities of each algorithm.

```
In [5]: ### voting_classify
print("==========================================")
from sklearn.ensemble import GradientBoostingClassifier, VotingClas
#import xgboost
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
#clf1 = GradientBoostingClassifier(n_estimators=200)
clf2 = RandomForestClassifier(random_state=0, n_estimators=500)
clf3 = LogisticRegression(solver = "lbfgs",random_state=1)
# clf4 = GaussianNB()
#clf5 = xgboost.XGBClassifier()
clf = VotingClassifier(estimators=[
    #('gbdt',clf1),
    ('rf',RF),
     ('lr',clf2),
    # ('nb',clf4),
    # ('xgboost',clf5),
    ('SVM',clf1)
    ],
    voting='soft')
clf.fit(X_train,y_train)
predictions = clf.predict(X_test)
print("voting_classify")
print(classification_report(y_test,predictions))
print("AC",accuracy_score(y_test,predictions))
```

```
==========================================
voting_classify
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3206
           1       0.98      0.98      0.98      3030

   micro avg       0.98      0.98      0.98      6236
   macro avg       0.98      0.98      0.98      6236
weighted avg       0.98      0.98      0.98      6236

AC 0.9810776138550352
```

Accuracy_score for Voting Classifier

0.9810776138550352

7) A simulation Demo using the best-performed classifier:

Using Random Forest to determine the action in CartPole game:

```python
def nex_action(observation):
    result = RF.predict([observation])
    return int(result[0])


#simulation Demo
env=gym.make('CartPole-v1')
for episode in range(10):
    observation = env.reset()
    tmp = 0
    for t in range(500):
        #env.render()
        action = nex_action(observation)
        observation, reward, done, info = env.step(action)
        tmp += 1
        if done:
            print("Episode finished after {} timesteps".format(tmp))
            break
    #print("reward: ", tmp)
env.close()
```

Results on RF:

```
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
Episode finished after 500 timesteps
```

We can see from the episodes that all reached best score!