从零开始，用Python创建AI炒股机器人-技术篇（建议收藏）

为AI呐喊 2020-01-20 17:38:23

定量交易

随着新技术的出现，算法交易越来越受欢迎，使得更多的定量投资者可以使用它。我过去曾写过关于 Java 算法交易系统的开发的文章。然而， Python 具有难以置信的强大的分析库，易于理解文档和实现。本文将向您介绍用Python 开发算法交易系统以及部署经过训练的AI模型来执行实时交易的核心组件。本文中介绍的这个项目的代码可以在 GitHub 上找到。

本文将分为以下几个步骤：

- I:连接到经纪公司
- II:贸易制度发展

- III:AI 贸易模式发展
- IV:AI 交易模式部署
- (奖励):云部署

连接到经纪公司

第一步是连接到经纪公司，这将使我们能够接收有关我们感兴趣的证券交易的实时数据。在本文中，我将使用 Alpaca ，这是最简单的自由方式开始算法交易，并为我们的目的， AI 交易。创建帐户并转到仪表板以生成 API 密钥。

一旦你生成了你的 API 密钥，你就可以直接使用 python 。我创建了一个 helper 类来管理 API 连接。

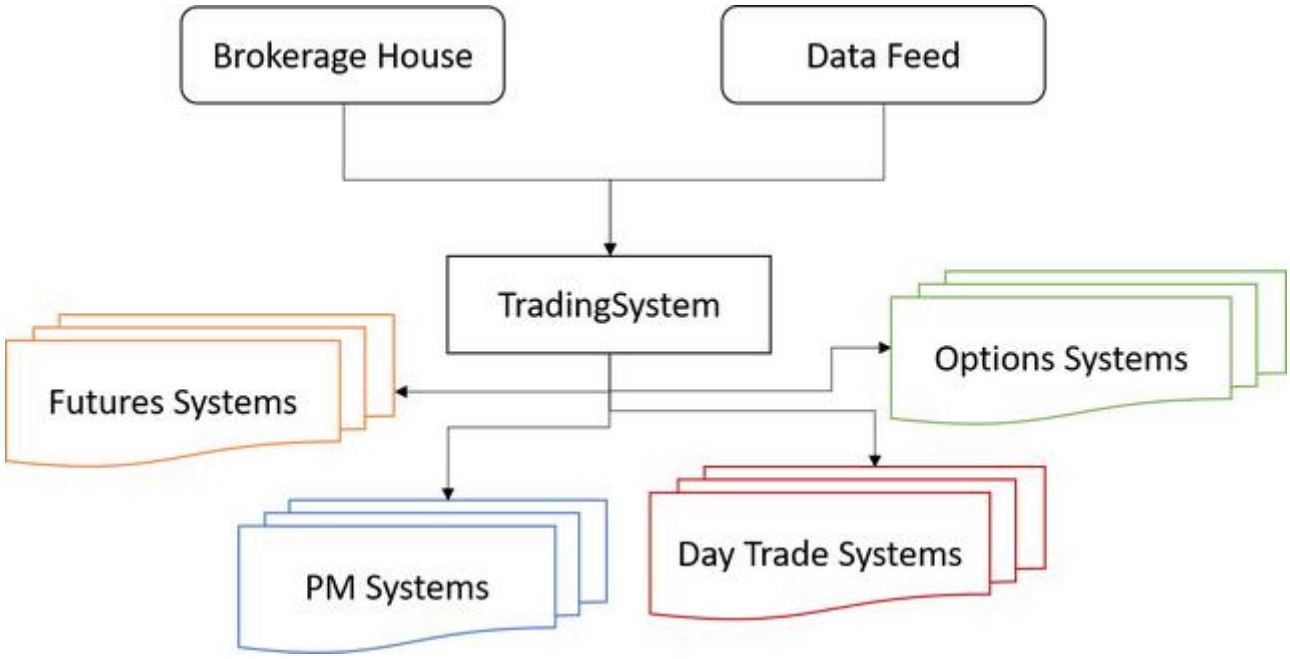
```
class AlpacaPaperSocket(REST):    def __init__(self):
super().__init__(            key_id='YOUR_KEY_HERE',
secret_key='YOUR_SECRET_KEY_HERE',           base_url='https://paper-
api.alpaca.markets'        )
```

这就是经纪连接，我们可以使用 AlpapaperSocket 类的实例作为对 API 的引用。下面我们将请求库存数据给我们的 AI 模型，但是如果您对如何请求库存信息感兴趣，现在可以在这里找到文档订

单。对我来说，这是一个很重要的问题，你可以使用任何你想要的经纪公司。（显然，它提供了一个 API 来满足数据和订单请求。）

交易系统开发

现在我们已经与经纪公司建立了联系，我们可以建立我们的交易系统。我已经创建了一个新的设计模式，能够容纳系统的任何安全与不同的时间框架和不同的 AI 模型。听起来很复杂吗？别担心，它实际上是一个非常简单的设计。



主要的思想是构造一个抽象的 TradingSystem 类，这样我们就可以为我们希望交易的每种类型的系统实现自定义规则集。代码是相当直接的，允许我们初始化系统和线程一个无限循环。

```
class TradingSystem(abc.ABC):    def __init__(self, api, symbol, time_frame,
system_id, system_label):        # Connect to api        # Connect to
BrokenPipeError        # Save fields to class        self.api = api
self.symbol = symbol        self.time_frame = time_frame
self.system_id = system_id        self.system_label = system_label
thread = threading.Thread(target=self.system_loop)        thread.start()
@abc.abstractmethod    def place_buy_order(self):        pass
@abc.abstractmethod    def place_sell_order(self):        pass
@abc.abstractmethod    def system_loop(self):        pass
```

TradingSystem 是一个抽象类，有几个抽象函数。允许函数是抽象的，让它们在保持类似类结构的同时，在实现和实现之间有所不同。例如，假设你有一个投资组合管理系统和一个日交易系统。投资组合管理系统的 system _ loop 将包含与日交易系统的 system _ loop 不同的 AI 模型。抽象类

TradingSystem 的实现是非常直接的。为了本文的目的，我将构建一个投资组合管理系统，稍后您将看到我训练一个 AI 模型来执行交易。现在，考虑以下实现...

```
class PortfolioManagementSystem(TradingSystem):    def __init__(self):
super().__init__(AlpacaPaperSocket(), 'IBM', 604800, 1, 'AI_PM')    def
place_buy_order(self):        pass    def place_sell_order(self):        pass
def system_loop(self):        pass
```

这个投资组合管理系统将容纳执行交易的 AI 。在我们开发了 AI 交易模型之后，我们将返回到这个实现。

AI 交易模型开发

对于这个系统，我将构建和训练 AI 模型作为我的系统的投资组合经理。其想法是训练神经网络在一定的负变化阈值下买进，并在一定的正变化阈值下卖出股票价格。我们本质上是在教我们的人工智能去买底泥和卖掉底泥。[请注意，我不建议您在实时系统中实现此功能，我们将进一步讨论此主题]为了培训此神经网络，我将基于 IBM 的每周历史市场数据构建和注释数据集，并创建一个称为信号的功能，该功能将根据更改阈值在集{0,1,-1}中产生值。

	A	B
1	Delta Close	Signal
2	-2.559997	1
3	3.410003	-1
4	3.679993	-1
5	3.25	-1
6	-1.709992	1
7	-3.440002	1
8	-4.220001	1
9	8.600006	-1
10	-2.480011	1
11	0.050003	0

现在让我们来考虑这个神经网络的架构。我们将在反向传播之后保存权重，以便在成功测试模型之后，我们可以部署它。

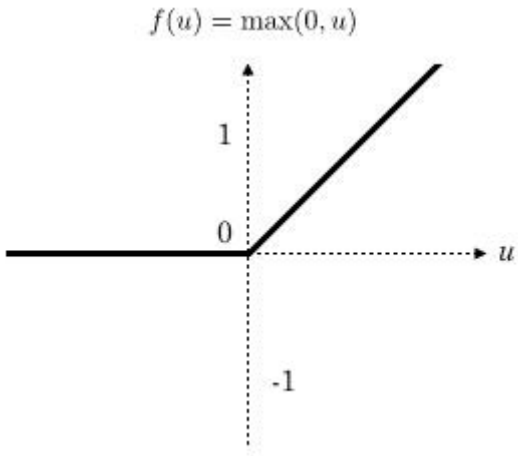
```
# Class to develop your AI portfolio managerclass AIPMDevelopment:    def
__init__(self):        # Read your data in and split the dependent and
independent        data = pd.read_csv('IBM.csv')        X = data['Delta
Close']        y = data.drop(['Delta Close'], axis=1)        # Train test
```

```
spit        X_train, X_test, y_train, y_test = train_test_split(X, y)
# Create the sequential        network = Sequential()        # Create the
structure of the neural network        network.add(Dense(1, input_shape=(1,),
activation='relu'))        network.add(Dense(3, activation='relu'))
network.add(Dense(3, activation='relu'))        network.add(Dense(3,
activation='relu'))        network.add(Dense(1, activation='relu'))        #
Compile the model        network.compile(
optimizer='rmsprop',                    loss='binary_crossentropy',
metrics=['accuracy']                    )        # Train the model
network.fit(X_train.values, y_train.values, epochs=100)        # Evaluate the
predictions of the model        y_pred = network.predict(X_test.values)
y_pred = np.around(y_pred, 0)        print(classification_report(y_test,
y_pred))        # Save structure to json        model = network.to_json()
with open("model.json", "w") as json_file:            json_file.write(model)
# Save weights to HDF5
network.save_weights("weights.h5")AIPMDevelopment()
```

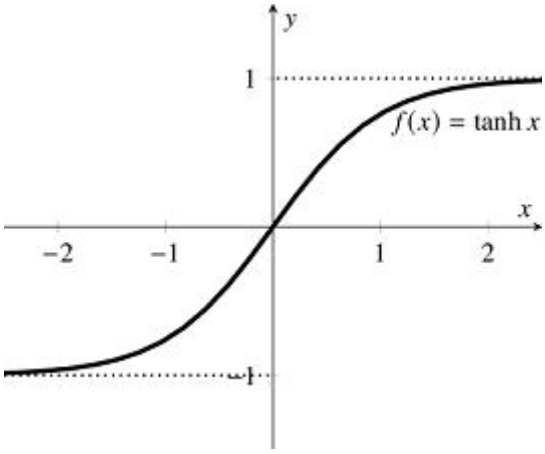
我们将使用序列模型，然而，重要的是要理解幕后的数学，否则我们不能期望人工智能做出我们想要的决定。在这种情况下，选择激活函数是至关重要的，如果我们关闭我们的眼睛，并选择 ReLU 与二值交叉熵损失函数，我们将得到一个混淆矩阵看起来类似于以下...

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	21
0	0.33	1.00	0.50	22
1	0.00	0.00	0.00	23
accuracy			0.33	66
macro avg	0.11	0.33	0.17	66
weighted avg	0.11	0.33	0.17	66

为什么这样？让我们可视化 ReLU 函数...



如果我们将负输入设置为零，或者使用损失函数期望二值输出，我们如何期望我们的模型识别短信号？双曲正切函数和铰链损失在这里帮助。



更新我们的神经网络，认识到我们出错的地方，我们有以下模型。

```
# Class to develop your AI portfolio managerclass AIPMDevelopment:    def
__init__(self):        # Read your data in and split the dependent and
independent        data = pd.read_csv('IBM.csv')        X = data['Delta
Close']        y = data.drop(['Delta Close'], axis=1)        # Train test
spit        X_train, X_test, y_train, y_test = train_test_split(X, y)
# Create the sequential        network = Sequential()        # Create the
structure of the neural network        network.add(Dense(1, input_shape=(1,),
activation='tanh'))        network.add(Dense(3, activation='tanh'))
network.add(Dense(3, activation='tanh'))        network.add(Dense(3,
activation='tanh'))        network.add(Dense(1, activation='tanh'))        #
Compile the model        network.compile(
optimizer='rmsprop',                        loss='hinge',
metrics=['accuracy']                        )        # Train the model
network.fit(X_train.values, y_train.values, epochs=100)        # Evaluate the
predictions of the model        y_pred = network.predict(X_test.values)
y_pred = np.around(y_pred, 0)        print(classification_report(y_test,
```

```
y_pred))        # Save structure to json        model = network.to_json()
with open("model.json", "w") as json_file:            json_file.write(model)
# Save weights to HDF5
network.save_weights("weights.h5")AIPMDevelopment()
```

在训练之后，我们发现分类报告有了显著的改进。

```
              precision    recall  f1-score   support

          -1       0.83      1.00      0.91        25
           0       1.00      0.06      0.12        16
           1       0.71      1.00      0.83        25

    accuracy                           0.77        66
   macro avg       0.85      0.69      0.62        66
weighted avg       0.83      0.77      0.69        66
```

现在我们已经成功地开发了我们的模型，它是保存模型并将其加载到专门托管它的类中了。

```
# AI Portfolio Managerclass PortfolioManagementModel:    def __init__(self):
# Data in to test that the saving of weights worked        data =
pd.read_csv('IBM.csv')        X = data['Delta Close']        y =
data.drop(['Delta Close'], axis=1)        # Read structure from json
json_file = open('model.json', 'r')        json = json_file.read()
json_file.close()        self.network = model_from_json(json)        # Read
weights from HDF5        self.network.load_weights("weights.h5")        #
Verify weights and structure are loaded        y_pred =
self.network.predict(X.values)        y_pred = np.around(y_pred, 0)
print(classification_report(y, y_pred))PortfolioManagementModel()
```

我们通过查看整个数据集的分类报告来验证神经网络的结构和正确加载的权重。

```
              precision    recall  f1-score   support

          -1       0.73      1.00      0.84        95
           0       1.00      0.07      0.14        69
           1       0.77      1.00      0.87        97

    accuracy                           0.75       261
   macro avg       0.83      0.69      0.62       261
weighted avg       0.82      0.75      0.67       261
```

太棒了！现在，我们已经建立了一个 AI 投资组合经理来根据股价的变化做出购买、出售和持有的决定。这通常是最受追捧的交易系统之一。回到大学时代，当我在 futures 年的市场运行算法交易系统时，年回报率超过20%，第一个问题总是“但是，它怎么知道什么时候交易？”".这是一个最难回答的问题，但当你能回答它的时候，你有一个有利可图的交易系统。开发一个有利可图的 AI 交易模型超出了这个项目的范围。然而，如果这篇文章得到足够的支持，我会很乐意写出我用来开发有利可图的 AI 交易模型的量化技术。

将 AI 模型部署到交易系统

在重新考虑抽象 TradingSystem 类的实现时，我们有了 PortfolioManagementSystem 。我更新了抽象函数以实现它们各自的目的。

```
class PortfolioManagementSystem(TradingSystem):    def __init__(self):
super().__init__(AlpacaPaperSocket(), 'IBM', 86400, 1, 'AI_PM')
self.AI = PortfolioManagementModel()    def place_buy_order(self):
self.api.submit_order(                          symbol='IBM',
qty=1,                          side='buy',
type='market',                          time_in_force='day',
)    def place_sell_order(self):        self.api.submit_order(
symbol='IBM',                          qty=1,
side='sell',                          type='market',
time_in_force='day',                          )    def system_loop(self):        #
Variables for weekly close        this_weeks_close = 0
last_weeks_close = 0        delta = 0        day_count = 0
while(True):            # Wait a day to request more data
time.sleep(1440)            # Request EoD data for IBM            data_req =
self.api.get_barset('IBM', timeframe='1D', limit=1).df            # Construct
dataframe to predict            x = pd.DataFrame(                data=[[
data_req['IBM']['close'][0]]], columns='Close'.split()            )
if(day_count == 7):                day_count = 0
last_weeks_close = this_weeks_close                this_weeks_close =
x['Close']                delta = this_weeks_close - last_weeks_close
# AI choosing to buy, sell, or hold                if
np.around(self.AI.network.predict([delta])) <= -.5:
self.place_sell_order()                elif
np.around(self.AI.network.predict([delta]) >= .5):
self.place_buy_order()PortfolioManagementSystem()
```

让我们来谈谈 system _ loop 。system _ loop 初始化本周关闭、上周关闭、当前增量和日数的变量。无限循环（并发系统的线程）负责每天收集一次数据，并确定我们是否已经达到每周拆分。在

达到每周拆分后，变量会被更新，我们会咨询我们的 AI 是否买卖。

24/7全天候运行的云部署

自然会出现的一个问题是“你是否希望我在我的电脑上运行这个 Python脚本？如果电源关闭了，我就会失去互联网等等…”。答案是，不，我不希望你整个星期都运行这个 Python脚本，我希望你利用云部署来托管它。这将允许您的软件有24/7的时间，同时减少在您自己的机器上运行它的复杂性。如果你有兴趣将你的模型部署到云上，你可以从 Google Cloud 上的算法交易系统部署（步骤将是 AI 交易机器人的同义词）的奇妙教程中完成这一点。

结论

这篇文章是为了让你开始开发人工智能股票交易机器人。我们讨论了如何连接到经纪公司，特别是 Alpaca 的这个例子。然后，我们创建了 TradingSystem 类本身及其继承字段，并在专门用于投资组合管理的系统中实现了该类。之后，我们建立了一个人工智能模型来进行交易决策，并讨论了在幕后缺乏对数学理解的问题。最后，我们将模型部署到实现的系统中，使我们的 AI 具有购买、销售和持有的能力。你可以对这个机器人进行各种升级，优化速度， AI 架构， P / L 报告等…然而，这是你可以在 Python 建立一个免费的人工智能股票交易机器人。

本文由**未艾信息（**www.weainfo.net**）**编译，