@peisuke 2019年12月26日に更新

ABEJA Advent Calendar 2019 10日目

株AIを結構頑張ったら、儲かりそうな雰囲気が出ている

DeepLearning Chainer chainerRL

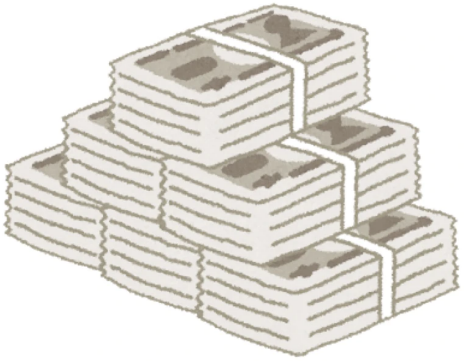
ABEJA Advent Calendarの10日目です。

はじめにのはじめに

以下は、あくまでテストデータで上手く行ってるよという話で、本当にこれをやったら儲かるかというと、まだまだわかりませんのであしからず！あとネタがネタだけに、今回のはあくまで個人のやってみた記録であり、組織の取り組みとは関係ありません。

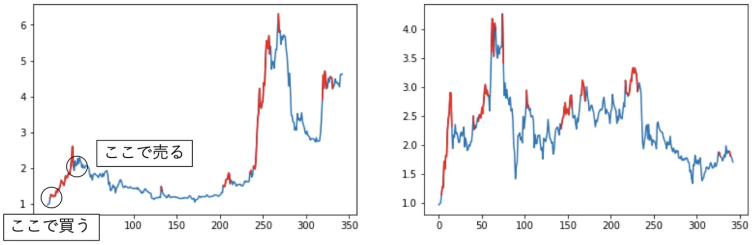
はじめに

お金が欲しい！無限に寿司が食いたい！株で儲けたい！



研究やエンジニアリングをしながら生きてく上で、将来のキャリアや技術スタックについて日々考えてるんですが、よくよく原点に立ち返るとそもそも技術スタックとかどうでもよくて、好きなものを作って漫画読んで生きていきたいんです。つまり結局、世の中は金なんですよね金。なので、何とかして寝てても圧倒的に儲かる仕組みを作りたい！そんな気持ちで私利私欲のために機械学習を使ったという記録です。

以下は、今回紹介する方法で実験したテストデータの一部です。赤が出力された買いポジションで、上方向に赤線が伸びている部分が儲けている領域です。定量的な結果は記事の最後に。



全体方針

そもそも機械学習で株で勝つためにはどうするか。株予想の研究は色々とありますが、翌日株が上がるか下がるかをある程度の確率で予測できても、中々勝てません。人間は心理的な障壁があるせいで、どうしても買い時、売り時を逃すものだからです。行動経済学ではプロスペクト理論というのですが、株価がプラスになってもすぐ売り、マイナスになると上がることを期待して保持してしまい、下がる株を永遠に持ってしまうというものです。

機械学習で株に勝つためには、売買戦略までを任せる必要があると個人的には考えています。そこで、強化学習を用いて売買のタイミングも含めて学習することにしました。強化学習は、過去の経験から良い戦略を学習するもので、過去の株価データから、何度も売買シミュレーションを繰り替えし、「こういうタイミングで買ったら利益を得た」というデータを蓄積・学習していきます。

この方針に乗っ取って、２年ほど前から細々と実験してきました。折角なので、その時の記録も少し載せていきます。

２年前の取り組み

まず、シンプルに強化学習を試してみようということで、米国のとある大企業の株をダウンロードして、全体でプラスマイナスの無い期間を選び、時間で学習/テストで分割して強化学習してみました。強化学習のロジックとしてDeep Q-Networkをそのまま使い、数層の1次元の畳み込みニューラルネットを利用してみました。結果としては、**年間±10%くらいの範囲で推移し、長期間の平均で年間利益3%くらい？という結果**でした。人生を500回くらい転生すれば、もしかしたら儲かるかもしれない。銘柄増やしてガバっと売買したとしても、景気の変動の方が大きいので、早々は勝てなさそう。まぁそんなに甘くはないよね・・・。

出典：https://golden-tamatama.com/blog-entry-1662.html

１年前の取り組み

やはり機械学習はデータとモデルを頑張らないといけないという事で、日本の株を収集しまくることにしました。日本株を収集する上で、Yahooからスクレイピングするのは禁止されているので、Yahoo VIP倶楽部に入会したり、Kabu+などで毎日の値動きを購入することにしました。さらに機械学習の方も事前学習をしてみたりベイジアンを使った予測モデルを併用して強化学習するなど頑張ってみました。株の値動きは本質的には観測不可能なランダムな値動きをすると仮定して、ベイジアンNNにAleatoric Uncertaintyを利用しました。

すると、なんと**売買するなとAIに言われるという結果**になり資産変化ゼロという結果に・・・orz

今回の取り組み

背景

結局の所、やはり機械学習にもドメイン知識は大切なんです。**何でもいいからデータぶっ込めば良い結果が出るなんて幻想**なんです。ということで、上記で失敗してから、僕自身が株売買に取り組むことにしました。あくまで儲けというよりはドメイン知識を増やすことを目的とし、色んな事を試しました。株関連書籍を読み漁り、株だけではなくFXや先物などにも手を出してみたり、Youtuberの株解説を聞いてみたり、~~40万円位するセミナーに入ってみたりしました~~。

さて、数々の勉強をしているうちに、ようやく勝てそうなポイントが見えてきました。それは、**仕手株**という奴です。仕手株とは、噂によると大口（本尊と言ったりもする）が予め安い価格で株を買い集めておき、その後株価を釣り上げて、個人投資家が参入した段階で売りをぶつけるという奴らしいです。本当にそんな事したら逮捕されるんじゃね？という話もあり、あくまでそういう噂があるよって話ですが、本も沢山出ているしマンガもあるので読んでみてはいかがでしょう。

漫画はこれです。

なぜ仕手株で勝てそうか、というと。裏にストーリーが存在するため、その流れに基づけば一定のルールが出来やすくなるからです。具体的なルールとしては大きくは３つあると言われています。

- 玉集め
- 玉転がし
- 売り抜け

玉集めとは、株価が高騰しないように注意しながら買い集める事です。チャートを見ると一定の値段のレンジで比較的少ない取引量で安定するという傾向があります。集める時期は、数ヶ月から数年と広いレンジがあるようです。玉転がしとは、集めた株を売りに出しつつ自分で高い値段で買うというテクニックらしいです。他の人に買わせず、自分自身で高い値段で買い取ってしまえば値段を吊り上げることができます。このタイミングで値段が高騰するので一般の投資家の注目を集め始め、投資家たちは自身も利益を出そうと、株を買い始めます。これに対して、買い集めてた株を売りまくります。結果として後から入ってきた個人投資家は高値を掴まされ、最初に株を持っていた人が儲ける、という仕組みです。この話がもし本当であるならば、明確なルールがあるため、玉集めされている銘柄を認識し、玉転がしのタイミングで買いを入れれば、儲けることができそうです。

実際に、とある銘柄の値動きをプロットすると下記のようになります。明らかにランダムウォークではなく何らかのルールがある事が見えてくるかと思います。

というわけで、やっていきましょう！

仕手株の認識に向けた準備

まずは、仕手株を認識していきましょう。仕手株といわれる銘柄は、時価総額などからある程度スクリーニングはできるものの、それを正しく認識した所でいつ高騰するかは分かりません。そこで、チャートを入力したら、その後に仕手株らしいルールのある動きになるかどうかを認識する識別器を作っていきます。仕手株らしいルールのある銘柄は、前節に載せたような「きれいな」チャートになります。具体的に「きれいな」とは、例えば支持線・抵抗線、三尊/逆三尊、移動平均線での反発などのルールを持つ事を指します。しかし、非常に曖昧であるため、これをプログラムでルール化することは非常に難しいです。

そこで、今回は上記ルール有無の認識を機械学習でやっていきます。機械学習は人手でルールを与えるのが面倒な場合に、とにかくデータをぶっ込めば良い結果を得られる凄いやつです。ここで、以下が凄い重要なのですが、学習データを与えるために**僕自身が大量に「この銘柄だったら勝負したい」「この銘柄は勝負したくない」を手動で与えました**。三日三晩ひたすらチャートに良し悪しのラベルを付けまくりました。以下のように、チャートに対するアノテーションを**9000件**実施しました。これは非常に重要なことなのですが、**機械学習に一番必要なのは数学ではなく根気**です。

以下が240日間のチャートに対して僕が良いと思った値動きです。単に上がるだけのものではなく、左下のように途中で急落してますが、途中までは下値を切り上げている動きに着目し、良しとします。

以下は僕が悪いと思った値動きです。反発ルールが見えなかったり、ダラダラ値下がりをしているというチャートになっていると思います。

仕手株を認識する機械学習モデル

Co-teachingによるnoisy labelに対する学習

さて、上記のようなアノテーションを僕自身が完璧にできるかというと非常に難しいです。判断がブレていったりチャートを見るスキルが次第に上がっていくこともあり、アノテーションの一貫性を保てません。そこで、誤ったラベルが含まれるデータセットに対する学習を

実施します。様々な手法が存在しますが、今回はCo-teachingを利用しました。これは２つの学習モデルを用意しておき、片方のモデルが「このデータは推論しやすい」と判断したデータを、もう片方のモデルが学習に利用する、これを相互に実施するというものです。これにより、判断に迷うデータを学習データから外すことで、確実に使えるデータにフォーカスして学習する事が可能になります。なおCo-teachingを日本語で解説している記事も、こちらにありますので、読んでみて下さい。

コードは下記のようになります。２つのネットワークに推論させて、lossの低い順に並べ替えた上で、学習するという流れです。Chainerを用いて実装しました。Updaterは下記のようになります。

```
y1 = network1(x)
y2 = network2(x)

loss1 = F.softmax_cross_entropy(y1, t, reduce='no')
loss2 = F.softmax_cross_entropy(y2, t, reduce='no')
ind1 = xp.argsort(loss1.data)
ind2 = xp.argsort(loss2.data)

count = int(np.round(x.shape[0] * (1 - self.rate_schedule[self.epoch - 1])))
logit1 = network1(x[ind2[:count]])
logit2 = network2(x[ind1[:count]])

xp = (1 + 2 * (xp.random.rand(*x.shape).astype(np.float32) - 0.5) * 0.02)  * x
yp1 = network1(xp)
yp2 = network2(xp)

kl_left1 = F.softmax(y1) * F.log_softmax(y1)
kl_right1 = F.softmax(y1) * F.log_softmax(yp1)
reg_loss1 = F.sum((kl_left1 - kl_right1)) / y1.shape[0]

kl_left2 = F.softmax(y2) * F.log_softmax(y2)
kl_right2 = F.softmax(y2) * F.log_softmax(yp2)
reg_loss2 = F.sum((kl_left1 - kl_right2)) / y2.shape[0]

optimizer1.update(self.get_loss, network1, logit1, t[ind2[:count]], reg_loss1)
optimizer2.update(self.get_loss, network2, logit2, t[ind1[:count]], reg_loss2)
```

Optimizerに流している `get_loss` は下記です。SoftmaxCrossEntropyをlossにしているだけですね。

```
def get_loss(self, net, x, t, reg=None):
    loss = F.softmax_cross_entropy(x, t)
    if reg is not None:
        loss += self.alpha * reg
    acc = F.accuracy(x, t)
    reporter.report({'loss': loss}, net)
    reporter.report({'accuracy': acc}, net)

    return loss
```

Temporal Convolutional Networkを用いた時系列データに対する学習

ネットワークではTemporal Convolutional Network (TCN)を用いました。TCNとは、WaveNetにも使われるDilated causal convolution layerを持つ、時系列の処理用に開発されたネットワークの一種です。TCNを用いることで、RNNと同じように過去の情報を利用できる手法です。RNNと比較して、状態を持たずとも過去の情報を利用できるものです。

```
import chainer
from chainer import link_hook
from chainer import functions as F
from chainer import links as L
from chainer import variable

class TemporalBlock(chainer.Chain):
    def __init__(self, n_inputs, n_outputs, kernel_size, stride, dilation, padding, dropout_r
        init = {
            'initialW': initializers.Normal(0.001),
            'initial_bias': initializers.Zero(),
        }

        super(TemporalBlock, self).__init__()
        self.padding = padding
        self.dropout_ratio = dropout_ratio
        self.is_downsample = False
        if n_inputs != n_outputs:
            self.is_downsample = True

        hook1 = WeightNormalization()
        hook2 = WeightNormalization()
        with self.init_scope():
            self.conv1 = L.Convolution1D(n_inputs, n_outputs, kernel_size,
                                         stride=stride, pad=padding, dilate=dilation, **init)
            self.bn1 = L.BatchNormalization(n_outputs)
            self.conv2 = L.Convolution1D(n_outputs, n_outputs, kernel_size,
                                         stride=stride, pad=padding, dilate=dilation, **init)
            self.bn2 = L.BatchNormalization(n_outputs)
            if self.is_downsample:
                self.downsample = L.Convolution1D(n_inputs, n_outputs, 1, **init)
        self.conv1.add_hook(hook1)
        self.conv2.add_hook(hook2)

    def forward(self, x):
        h = self.conv1(x)
        h, _ = F.split_axis(h, [-self.padding,], 2)
        h = F.dropout(F.relu(h), ratio=self.dropout_ratio)

        h = self.conv2(h)
        h, _ = F.split_axis(h, [-self.padding,], 2)
        h = F.dropout(F.relu(h), ratio=self.dropout_ratio)

        if self.is_downsample:
            res = self.downsample(x)
        else:
            res = x

        return F.relu(h + res)

class TemporalConvNet(chainer.ChainList):
    def __init__(self, num_inputs, num_channels, kernel_size=2, dropout_ratio=0.2):
        super(TemporalConvNet, self).__init__()
        layers = []
        num_levels = len(num_channels)
        for i in range(num_levels):
            dilation_size = 2 ** i
            in_channels = num_inputs if i == 0 else num_channels[i-1]
            out_channels = num_channels[i]
            tmp_lnk = TemporalBlock(in_channels, out_channels, kernel_size,
                                    stride=1, dilation=dilation_size,
                                    padding=(kernel_size-1) * dilation_size,
                                    dropout_ratio=dropout_ratio)
            self.add_link(tmp_lnk)

    def forward(self, x):
        for link in self.children():
            x = link(x)
        return x

class TCN(chainer.Chain):
    def __init__(self, input_size, output_size, num_channels, kernel_size, dropout_ratio):
        super(TCN, self).__init__()
        num_channel = num_channels[-1]
        with self.init_scope():
```

```
            self.tcn = TemporalConvNet(input_size, num_channels,
                                       kernel_size=kernel_size, dropout_ratio=dropout_ratio)
            self.fc = L.Linear(num_channel, output_size)

    def forward(self, x):
        feat = self.tcn(x)
        y = self.fc(feat[:, :, -1])
        return y
```

なお、内部でweight normalizationを使っておりますが、式とコードは下記になります。

```
def _get_expander(ndim, axis):
    expander = [None] * ndim
    expander[axis] = Ellipsis
    return expander


def _norm_axis(weight, reduce_axes, eps=1e-12, keepdims=True):
    # reduce_axes = tuple([i for i in range(weight.ndim) if i != axis])
    reduce_axes = tuple(reduce_axes)
    squared_norm = F.sum(weight * weight, axis=reduce_axes, keepdims=keepdims)
    norm = F.sqrt(squared_norm + eps)
    return norm


def _expand_and_broadcast(var, expand_axes, shape):
    for i in expand_axes:
        var = F.expand_dims(var, i)
    return F.broadcast_to(var, shape)

class WeightNormalization(link_hook.LinkHook):
    def __init__(self, axis=0, eps=1e-12, weight_name='W', name=None):
        self.axis = axis
        self.eps = eps
        self._initialied = False

        self.weight_name = weight_name

        if name is not None:
            self.name = name

    def forward_preprocess(self, cb_args):
        link = cb_args.link
        if not self._initialied:
            input_variable = cb_args.args[0]
            self._prepare_parameters(link, input_variable)
            print('Initialized weights.')

        weight = getattr(link, self.weight_name)
        self.original_weight = weight

        normalized_weight = self._reparameterize_weight(link)
        setattr(link, self.weight_name, normalized_weight)

    def forward_postprocess(self, cb_args):
        link = cb_args.link
        setattr(link, self.weight_name, self.original_weight)

    def _prepare_parameters(self, link, input_variable=None):
        if getattr(link, self.weight_name).array is None:
            if input_variable is None:
                return
            else:
                link._initialize_params(input_variable.shape[1])

        initialW = getattr(link, self.weight_name)

        expander = _get_expander(initialW.ndim, self.axis)
        self.expand_axes = tuple([i for i, d in enumerate(expander) if d is None])
```

```
        with chainer.no_backprop_mode():
            g_variable = _norm_axis(initialW, self.expand_axes, self.eps)

        with link.init_scope():
            link.g = variable.Parameter(g_variable.array)

        self.shape = initialW.shape
        self._initialied = True

    def _reparameterize_weight(self, link):
        weight = getattr(link, self.weight_name)
        norm = _norm_axis(weight, self.expand_axes, self.eps)
        norm = F.broadcast_to(norm, self.shape)
        g = F.broadcast_to(link.g, self.shape)
        W = g * weight / norm
        return W
```

学習方法について

さて、推論モデルが最終的に求めたいのは、与えられたチャートが仕手株かつ値動きのある良いタイミングかどうかです。アノテーションは240日間のデータに対して実施していますが、このチャートを全部突っ込んで学習・推論しても、その後どうなるかはわかりません。そこで、240日の内のはじめの60日の値動きを入力データ、付与したラベルを正解として認識することとしました。入力データの具体的な値としては、始値・終値・高値・安値、5日・25日の移動平均線やMACD、RCIの情報、出来高を入力としました。

やってしまった

なんと、ここまで来て気づいたんですが、**アノテーションデータを作る際に期間で切り分けるのを忘れていた**ため、仕方なく銘柄で学習/テストをスプリットしました。まぁ仕手株だったら、日経平均の影響を受けないし、銘柄感の動きはそこまで連動しないので、一旦は良しとしよう・・・！アノテーションデータ数千件やり直すわけにも行かないしね。実際の所、去年のMTジェネとテリロジーや最近だとKlabとenishなど多少連動します。が、株と機械学習をやっている感覚としては、日付も含めて丸覚えでもしないかぎりは、多分大丈夫！明後日の方向でないのであれば、まずは走り切るのが大事。

追記：

そんなわけで、実験の条件としては、以下となります。まず、上場銘柄約3600のうち、ボラが高めで、そこそこの出来高のある600種の銘柄を予め抽出しておきます。600銘柄をtrain/testで3:1に分けます。その上で、各銘柄から240日分の期間のチャートを、ランダムに15個ピックして、450*15個のチャートをtrain、150*15個チャートをtestとしました（240日もピック出来ない銘柄もあり、厳密なデータ数は多少違う値ですが、Qiitaということで雰囲気でお願いします）。その上で、モデルには最初60日のみを与えて、240日を見通せる神の目から見たチャートの良し悪しを学習・推論します。本当は銘柄を分けた上で、更に期間も分けるのが正しいのですが、そこは残念、今後の課題。

認識結果

まずは、定性的な評価として、「これは良い」と判定したテストデータの例を以下に示します。左のグラフが最初の60日の値動き、右が240日全体の値動きになります。グラフを見る限り良い動きをしていそうです。

一方、定量評価なのですが、アノテーションは僕自身が作ったため論文にあるような「正しい」ラベルデータを持っておらず、適切な精度評価は出来ません。一旦、僕自身が作成したアノテーションデータに対する精度を測ることとしまして、以下のようになりました。ここでは、co-teaching無しと比較するのと、co-teachingした際に、「このデータは推論しやすい」とする閾値を0.5、0.8にした場合で比較しました。閾値を高くするほど利用できるデータ数は減りますが、勝率のほうが重要ですので問題無しですね。

手法	条件	精度
w/o Co-teaching	---	60.4%
w/ Co-teaching	0.8	**93.9%**
w/ Co-teaching	0.5	80.5%

売買タイミングの強化学習

こちらはシンプルに先程選んだ銘柄について強化学習で売買していきます。強化学習をする際に、長期リターンを目指した報酬設計が必要になります。そのためシンプルに直近の収益を報酬にすると単なる翌日の株価の推論モデルを作ろうとしますが、株においては短期の値動きはランダムに近いため学習が非常に難しい、つまり報酬がノイズになってしまいます。そこで、前回評価から一定期間あけて、数日後に益がでたかどうかで0/1の報酬を与えることとしました。また手数料を加味して一回の取引で所定の負の報酬を与えることとしました。実装では幾つかの有名な工夫を入れましたが、大まかには通常の強化学習と一緒です。

Dueling Network

ネットワークの構成として、Dueling Networkというものを用います。このネットワークでは、各動作を取ったときの将来報酬の予測と、現在の状態の価値を同時に算出する方法です。経験上、多くの場合でこの方法を使うことで精度があがったりします。機構的にはどの領域に注目しているかが内部で計算されているようです。

```
def forward(self, x):
    batchsize = len(x)

    h = F.relu(self.c0(x))
    h = F.relu(self.c1(h))
    feat = self.tcn(h)

    h = F.reshape(feat[:, :, -1], (batchsize, -1))
    ha = self.advantage_value(F.relu(self.fc1_a(h))) # Linear
    mean = F.reshape(F.sum(ha, axis=1) / self.n_actions, (batchsize, 1))
    ha, mean = F.broadcast(ha, mean)
    ha -= mean

    hs = self.state_value(F.relu(self.fc1_s(h))) # Linear
    ha, hs = F.broadcast(ha, hs)

    q = ha + hs

    return q
```

Prioritized Experince Replay

DQNなどのリプレイバッファではランダムに過去の動作をピックして学習に利用します。これに対して、Prioritized Experience Replayでは、その中でも学習に寄与する割合が多そうな動作をtd誤差を計算することで優先的にピックします。本機能はChainer RLに実装済みなので、サクッと試すことが出来ます。

結果！！！

さて、先に述べたように期間でスプリットせず銘柄でスプリットして実験しました・・・。
さて、まずはランダムに幾つかピックアップしてみた結果を載せます。赤が買いポジションで青がノーポジです。なお、空売りや信用は増担保規制とか金利とか色々と規制ルールがあるのでやりませんでした。

良い感じじゃね！！？

定量評価した結果、300日運用した場合のパフォーマンス平均は239%増!!となり、大幅に利を取れそうな感じになりました。まぁスプリットミスってたりするくらいだし、バグもある気もしますけどね。

おわりに

当然ながら、239%おかしくね？という意見もありますが、仕手株で初動を捉えると雑に売買しても30%くらいは軽く取れるので、比較的体感とは合う数値かなと思います。あと仕手株は取引量が少ないため、どう頑張っても提灯（本尊に乗っかって利鞘を取る）だと大きくは儲けられません。あと、そもそもアノテーションを1万件近くやっていると目が肥えてきて、そもそもAIが無くても売買出来るようになってしまうという課題も見つかりました、AIいらんわ・・・。

さて、これ自体は今年の秋にやった結果で、株を自動売買するAPIが公開されていないことから、現在はBitcoin売買AIを中心に実験を進めています。しかし今の所、連戦連敗、上手くいったと思ったらAIが買えの一点張り。やはり先物やFXなど、規模の大きい所に攻めていくのは難しいなと感じました。こちらもいつか公開したいなと思っていますので乞うご期待。

追記

はてぶ等で頂いているツッコミ＆コメントへのレスポンス、あくまで僕の経験と考えです、参考になれば幸いです。

ファンドが超短期の自動売買でやっているのに勝てないのでは

FXや先物と違って、仕手株は流動性が足りないので、スキャでは取れても10000円とかのレベルになるので、ファンドもやらないんじゃないかなと思います。仕手株は、大衆心理を利用して煽るという仕組みなので、スイング～デイくらいの目線になると思っています。

9勝1敗で負ける可能性がある

その通りと言えばその通りですが、冒頭に書いたように、それは人の心理に基づく売買をした場合だと思ってます。小さい利鞘で売買して、最後に塩漬けみたいな事は、機械はあまりやらないですね。実際にグラフを見ても容赦なく切っているのが見えると思います。

経済が上向きの時は儲かる

仕手株は日経平均と殆ど連動しないので、その辺はそこまでは関係ないかとは思っています。もちろん、仕手でも地合を読みはするので、関連性ゼロってほどではないですけど。日経が上がるから期待値がプラスかというと、そうでもないかなと思います。一方、BitcoinAIの方は、その問題にハマって、とにかく買えとAIに言われ続けております・・・。

取引不成立のストップ安みたいなブラックスワンへの対応

そ れ な

実はストップの対応までは学習させるの面倒だったのでサボっていて、そこは今後の課題です。現状のロジックだと、少し前のサンバイオみたいなのを食らってしまう。

IB証券ってAPI公開

これは知らなかったです！後で申し込もう。

ファンダの方が儲かるのでは

少なくとも仕手株はファンダと相性悪すぎですね。もしかしたら、決算タイミングで何かしらの材料予想みたいなのはできるかもしれないですが。

このノウハウをいい感じのパッケージで販売するのが一番儲かりそう

これがですね、事前にアンケート取ってみたんですが、売買パッケージまで作っちゃうと面白いことにプロスペクト理論が働いて意外と売れないんですよね。人間は不透明なことに投資を出来ない、というのが大多数の心理だったりするんだなと思いました。逆に、一発逆転を狙った不透明情報みたいにすると売れるのは心理学的に有り得そうだなと思ったけど、そういうのって馬券の情報屋みたいな感じで確かに既にいるなと。世の中上手くできています。

既にやられているのでは？

僕も界隈の研究事情はあまり知らないけど、仕手筋の個別銘柄に対するアプローチって、機関単位だと大きな儲にならないから、積極的にはやらないんじゃないかなという気はしてます、分からんけど。個人的な立場としは、提灯側の立場での論文とかがあったら読んでみたい。超短期売買でスキャルピングする研究はそこそこ進んでいるようですね。ただ、問題の複雑さより、情報の速さが支配的なので、あまり面白みを感じないかも・・・。

ちなみに、仕手銘柄の動きで見かけるのは、(1)セクター単位で大きく急落させる操作と、(2)それに伴うロスカット的な動きでしょうか。(1)に関しては市場を操作する側の動きなので、今回のような提灯側の動きとは別の方針ですね。(2)は機械学習というよりはロスカット入れているかどうかというレベルかなと思います。