Рычко Артем, БЭК-155, 13.06.2018

Сравнение торговых алгоритмов на разных финансовых активах.

Введение

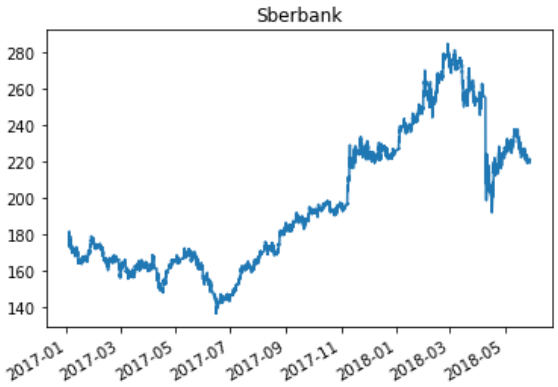
Данная работа ставит перед собой основную цель - сравнить алгоритм технического анализа, классификатора (обученного через алгоритмы машинного обучения) и принципа buy & hold для таких финансовых инструментов, как Форекс, фондовый рынок. Обучающей выборкой будет являться 2017-2018.3.1 год, а тестовой для оценки алгоритмов –с 1 марта 2018 года по 28 мая.

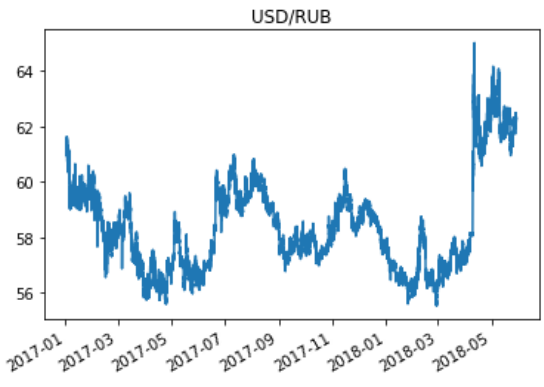
С каждым годом появляется все большее количество финансовых активов и алгоритмов для предсказания поведения данных активов. Индустрия трейдинга является высоко финансированной и бурно развивающейся, так по расчетам Всемирного Банка 4-кратный объем Мирового ВВП в 2017 году составляется рынок Форекс, где ежедневный оборот составляет 5.3 триллиона долларов США. Годовой объем на фондовом рынке составляет около 78 триллиона, и 0.5 триллиона на рынке криптовалют.[1]

Для расчетов и получения данных используется язык программирования Python 3.6 и необходимые модули. Официальный репозиторий исследования, где есть: скрипт для получения данных, скрипт для построения алгоритмов, скрипт оценки эффективности торговых стратегий.[2]

Описательная статистика данных

Данные для расчет представлены в виде цен акций Сбербанка и котировки курса USD/RUB по минутам:



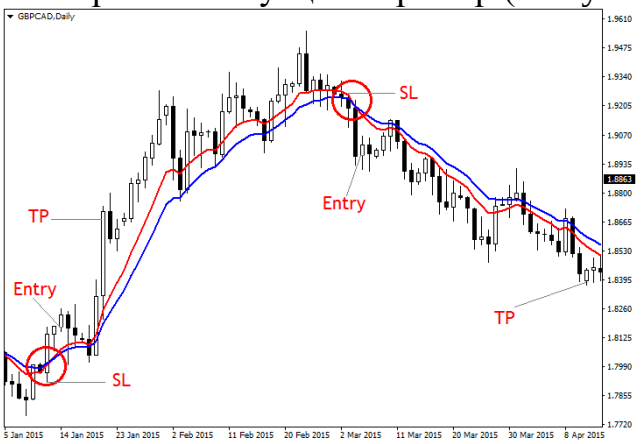


[1] https://data.worldbank.org/indicator/CM.MKT.TRAD.CD?end=2017&start=1975&view=chart
[2] github.com/artemrychko/trading_algorithms

Рычко Артем, БЭК-155, 13.06.2018

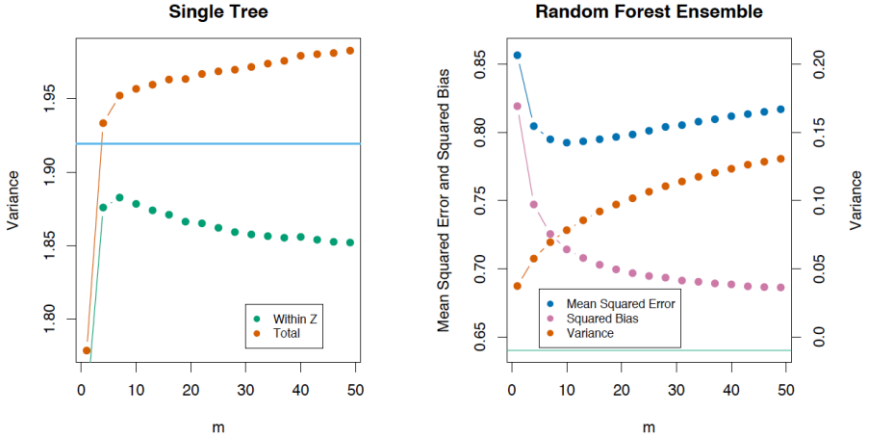
Алгоритм 1

Торговая стратегия с пересечением двух скользящих средних — простая система, которая основана на пересечении двух стандартных индикаторов — быстрой MA (скользящей средней или moving average) и медленной MA. FMA – быстрая MA, SMA – медленная MA. Условием покупки является пересечение FMA и SMA снизу, условием продажи является обратная ситуация. Пример (Entry - покупка):



Алгоритм 2

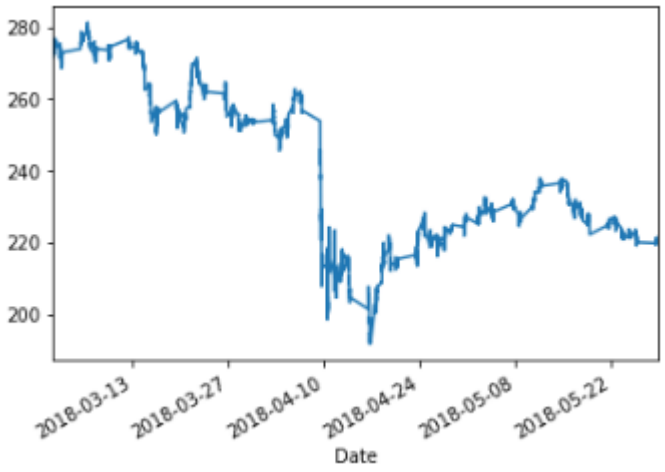
Классификационный алгоритм методом случайного леса – является одним из эффективнейших алгоритмов машинного обучения. Random forest (с англ. — «случайный лес») — алгоритм машинного обучения, предложенный Лео Брейманом и Адель Катлер, заключающийся в использовании комитета (ансамбля) решающих деревьев. Алгоритм сочетает в себе две основные идеи: метод бэггинга Бреймана, и метод случайных подпространств, предложенный Тин Кам Хо. Алгоритм применяется для задач классификации, регрессии и кластеризации. Основная идея заключается в использовании большого ансамбля решающих деревьев, каждое из которых само по себе даёт очень невысокое качество классификации, но за счёт их большого количества результат получается хорошим. В данном применении будет использоваться классификатор в методе случайного леса. В качестве переменных подадим лаги определенных параметров: цены, объемов, дисперсии, индексы.

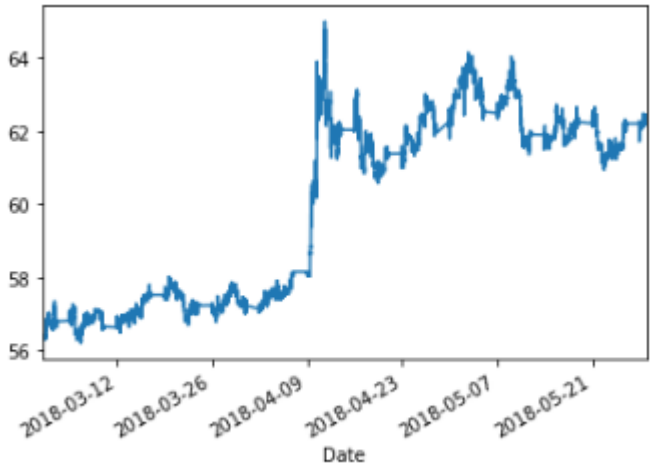


Рычко Артем, БЭК-155, 13.06.2018

Алгоритм 3

Поскольку нашей задачей была возможность заработать с 1 марта по 28 мая 2018 года, то последним алгоритмом для сравнения предыдущих будет являться инвестиционный алгоритм “Buy and Hold”, где главным принципом является купить на стартовой дате и продать на финальной. Прибылью является разница между продажей и покупкой скорректированная на комиссию. Сбербанк и доллар/рубль:





Общий результат = -50.65 рублей на акцию или -18.6% за почти 3 месяца для Сбербанка и 5.94 рубля на доллар или 10.5% прибыльности.

Техническая Реализация

Данные алгоритмы реализованы посредством использования программного обеспечения Python 3.6 и фрэймворка Jupyter Notebook. Сама работа, ее реализация и данные доступны на официальном репозитории данного исследования [github.com/artemrychko/price_prediction].

Для более быстрого бэктестинга был написан собственный модель, позволяющий проводить тесты прибыльности и имплементировать алгоритмы, а также подбирать параметры для используемых методов. Каждый алгоритм обучался на тренировочных данных (с 1 января по 1 марта) и тестировался на тестовых данных (с 1 марта по 28 мая).

Для первого алгоритма производился итерационный бэктестинг параметров FMA и SMA, где максимум для SMA являлся 10 часть выборки для обучения, шаг равный кубическому корню из выборки (для данных по акциям Сбербанка равен был 2973 минутам). Для FMA шагом оказался квадратный корень от шага SMA (или 55 для Сбербанка). Итого скорость подсчета, где t = время бэктеста одной итерации:

$T = t * log_{step_{SMA=n^{0.33}}}(n) * log_{step\_SMA^{0.5}}(n)$, вместо $T = t * n^2$, что значительно увеличило скорость подбора параметров, хоть и не реализовала расчет всевозможных параметров, но является приближенным к теоритически-существующим более прибыльным параметрам. Пример как выглядит зависимость MA:

Для второго алгоритма использовался модель sklearn со встроенным алгоритмом «случайного леса».

Для алгоритма “Buy and Hold” бралась разница между ценой в конце дня 28 мая и 1 марта 2018 года.

Результат Алгоритма 1

В данном алгоритме были подобраны малая и большая скользящая средняя и посчитана прибыльность для данных по Сбербанку:

	profit	Mov.Avs
291	95.81	(8919, 7095)
295	96.42	(8919, 7315)
3	96.90	(2973, 165)
294	97.12	(8919, 7260)
300	97.18	(8919, 7590)
297	97.86	(8919, 7425)
293	98.11	(8919, 7205)
292	98.33	(8919, 7150)
298	99.35	(8919, 7480)
299	100.68	(8919, 7535)

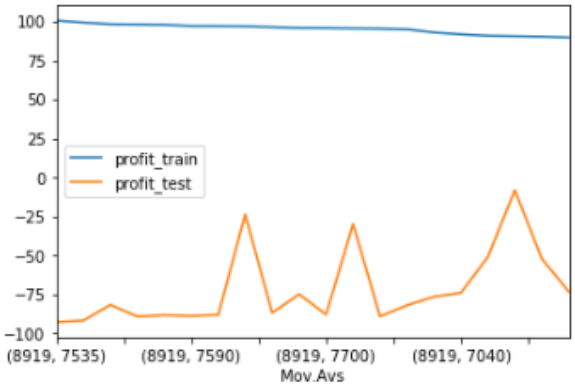
Если имплементировать данный алгоритм для тестовой выборке, то результаты получается не лучшими:

Рычко Артем, БЭК-155, 13.06.2018

	profit_train	profit_test	Mov.Avs
0	100.68	-92.59	(8919, 7535)
1	99.35	-91.66	(8919, 7480)
2	98.33	-81.57	(8919, 7150)
3	98.11	-89.07	(8919, 7205)
4	97.86	-88.23	(8919, 7425)
5	97.18	-88.67	(8919, 7590)
6	97.12	-88.07	(8919, 7260)
7	96.90	-23.69	(2973, 165)
8	96.42	-86.71	(8919, 7315)
9	95.81	-74.82	(8919, 7095)

Даже следующие 10 прибыльных параметров также оказываются малоприбыльными:

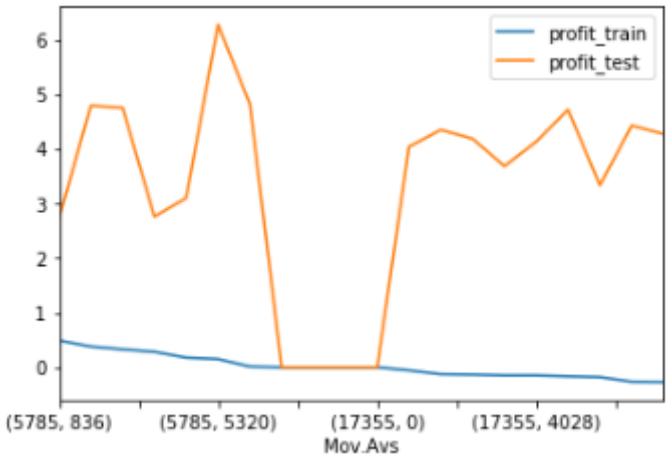
	profit_train	profit_test	Mov.Avs
0	100.68	-92.59	(8919, 7535)
1	99.35	-91.66	(8919, 7480)
2	98.33	-81.57	(8919, 7150)
3	98.11	-89.07	(8919, 7205)
4	97.86	-88.23	(8919, 7425)
5	97.18	-88.67	(8919, 7590)
6	97.12	-88.07	(8919, 7260)
7	96.90	-23.69	(2973, 165)
8	96.42	-86.71	(8919, 7315)
9	95.81	-74.82	(8919, 7095)
10	95.76	-87.80	(8919, 7700)
11	95.57	-29.76	(2973, 330)
12	95.45	-89.02	(8919, 7645)
13	95.00	-81.89	(8919, 7370)
14	92.99	-76.39	(8919, 6985)
15	91.79	-73.90	(8919, 7040)
16	90.90	-50.92	(2973, 715)
17	90.63	-8.07	(5946, 165)
18	90.34	-52.03	(2973, 550)
19	89.78	-73.09	(8919, 6930)



Итого, можно говорить о убыточности проделанных результатов для Сбербанка, но мы могли заметить, что для Сбербанка тестовый период был медвежьим. Для пары доллар/рубль он был бычьим, что оставляет вероятность успешной реализации. Давайте проверим:

	profit_train	profit_test	Mov.Avs
0	0.4920	2.7250	(5785, 836)
1	0.3802	4.7965	(5785, 5700)
2	0.3283	4.7564	(11570, 5624)
3	0.2830	2.7633	(5785, 760)
4	0.1764	3.1034	(5785, 912)
5	0.1508	6.2813	(5785, 5320)
6	0.0137	4.8255	(11570, 5700)
7	0.0000	0.0000	(5785, 0)
8	0.0000	0.0000	(23140, 0)
9	0.0000	0.0000	(11570, 0)
10	0.0000	0.0000	(17355, 0)
11	-0.0574	4.0420	(11570, 6004)
12	-0.1272	4.3526	(11570, 5092)
13	-0.1348	4.1888	(11570, 5928)
14	-0.1482	3.6857	(17355, 4180)
15	-0.1485	4.1360	(17355, 4028)
16	-0.1658	4.7229	(11570, 5548)
17	-0.1800	3.3344	(11570, 7980)
18	-0.2638	4.4308	(11570, 7524)
19	-0.2703	4.2762	(17355, 14440)

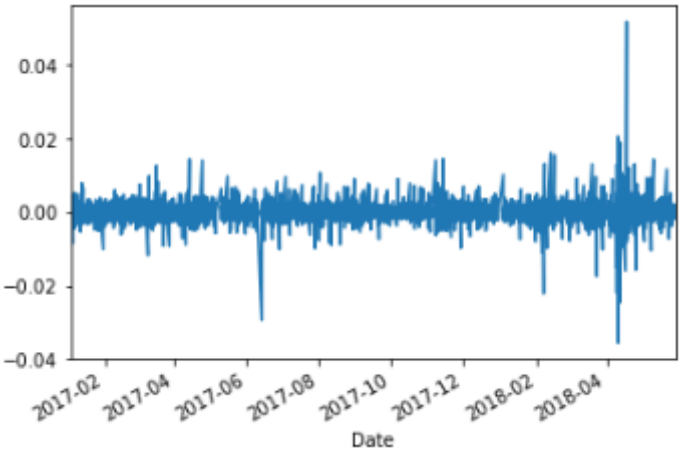
Можем заметить, что 7 алгоритмов с прибылью более, чем 0 на единицу в profit_train имеют высокую прибыльность на тестовом периоде от 2.7 до 6.3 рублей на один доллар.

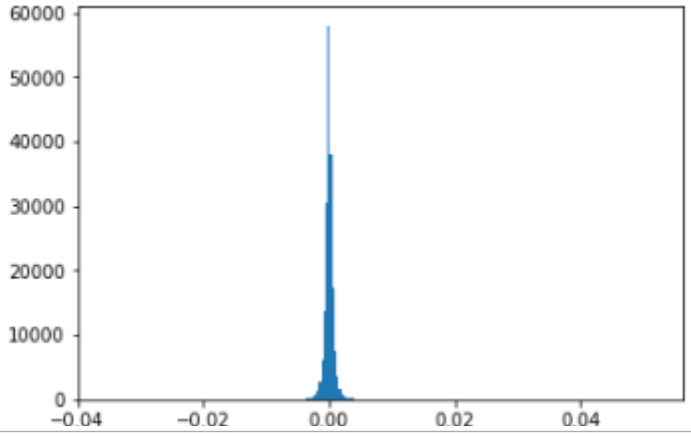


Рычко Артем, БЭК-155, 13.06.2018

Результат Алгоритма 2

Для данного алгоритма были использованы следующие параметры, как выявленные оптимальными: n_estimators = 100, максимальное количество деревьев = 30. Предсказывается переменная больше ли 0% изменение в цене. Были созданы 50 лагов для предсказываемой переменной, лагов дисперсий, изменения цены. Для Сбербанка можем видеть следующие результаты:






```
In [341]:
1 from sklearn.ensemble import RandomForestClassifier
2 np.random.seed(20180301)
3 rf = RandomForestClassifier(n_estimators=100, max_depth=30, n_jobs=8, verbose=4)
4 # rf.set_params()
5 rf.fit(x_tr, y_tr)
6 print('train = ', rf.score(x_tr, y_tr))
7 print('test = ', rf.score(x_te, y_te))

building tree 94 of 100
building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:   42.2s finished
[Parallel(n_jobs=8)]: Done   9 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done  82 tasks      | elapsed:    1.6s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    1.9s finished
train =  0.999866768811
[Parallel(n_jobs=8)]: Done   9 tasks      | elapsed:    0.0s
test =  0.526011195092
[Parallel(n_jobs=8)]: Done  82 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.3s finished
```

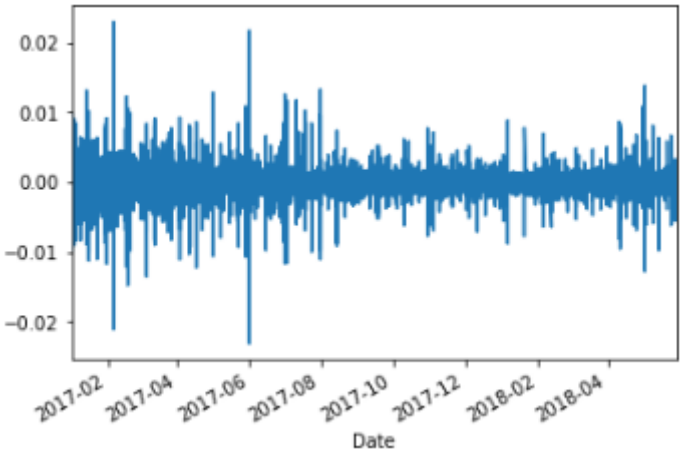
Модель на тестовой выборке показывает 0.52, что является лучше, чем подбрасывание монеты и теоретически при увеличении количества будущих сделок и устойчивости скоринга должно окупать проценты за комиссию и приносить прибыль. Но в итоге модель показывает отрицательный результат в прибыле: -38.45 рублей на акцию, что может объясняться видом оцениваемого скоринга модели, а также распределением классов в

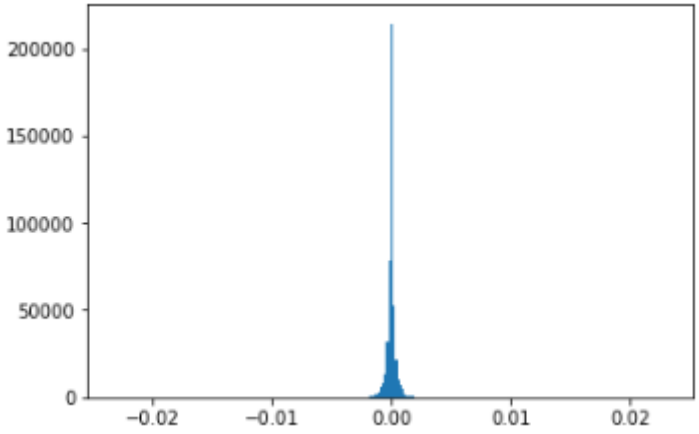
предсказываемой величине. Скоринг считает правильно угаданные все классы на общее число наблюдений, а мы имеем смещение в пользу отрицательных изменений, разница между скорингом и средним = -0.64, что объясняет отрицательную прибыль. Остается шанс при создании хорошей модели со скорингом больше распределения получить прибыльную модель.

```
ohlc['change'].describe()
```

```
count    181736.000000
mean          1.532360
std           0.498953
min           1.000000
25%           1.000000
50%           2.000000
75%           2.000000
max           2.000000
Name: change, dtype: float64
```

Давайте посмотрим, что получается на Форексе:

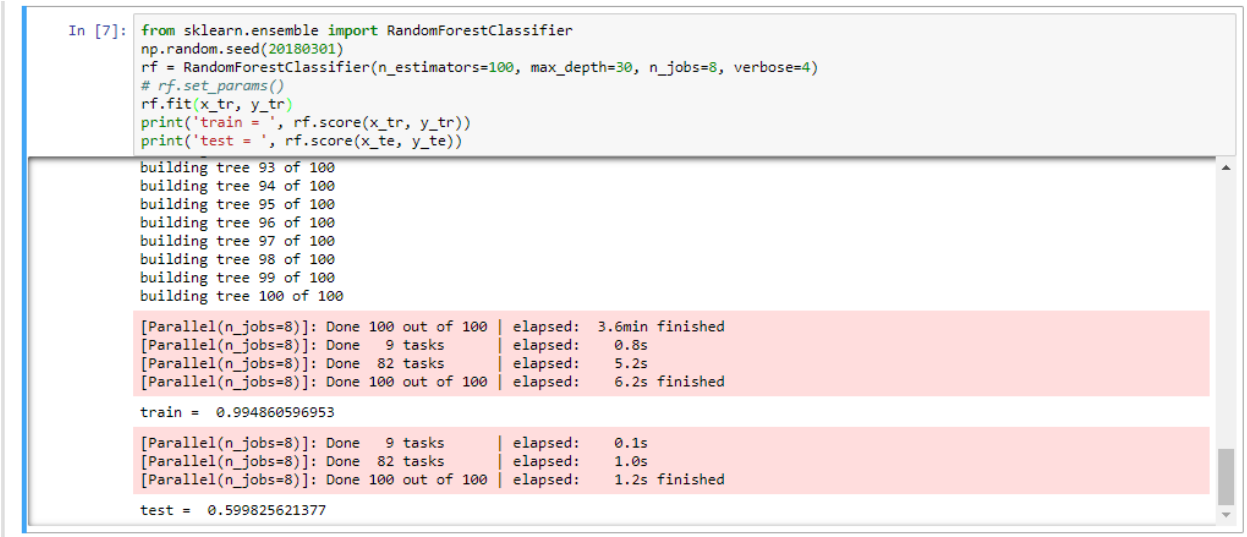





```
In [4]: ohlc['change'].describe()

Out[4]: count    464794.000000
        mean          1.557626
        std           0.496669
        min           1.000000
        25%           1.000000
        50%           2.000000
        75%           2.000000
        max           2.000000
        Name: change, dtype: float64
```

В данном случаи на тестовой выборке нужно получить больше 0.557:



```
In [7]: from sklearn.ensemble import RandomForestClassifier
        np.random.seed(20180301)
        rf = RandomForestClassifier(n_estimators=100, max_depth=30, n_jobs=8, verbose=4)
        # rf.set_params()
        rf.fit(x_tr, y_tr)
        print('train = ', rf.score(x_tr, y_tr))
        print('test = ', rf.score(x_te, y_te))

building tree 93 of 100
building tree 94 of 100
building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:  3.6min finished
[Parallel(n_jobs=8)]: Done   9 tasks      | elapsed:    0.8s
[Parallel(n_jobs=8)]: Done  82 tasks      | elapsed:    5.2s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    6.2s finished
train =  0.994860596953
[Parallel(n_jobs=8)]: Done   9 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Done  82 tasks      | elapsed:    1.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    1.2s finished
test =  0.599825621377
```

И мы его получаем, что говорит о вероятности прибыльности на бэктестинге: -70.206 рублей на доллар, и это без учета комиссий.

Итоги

Алго/тип	Score_train (sber)	Score_test (sber)	Train (us/ru)	Test (us/ru)
Тех.анализ	89< <101 (руб/акция)	-93 < < -8 (руб/акция)	0 < < 0.49 (руб/долл)	2.7 < < 6.3 (руб/долл)
R.Forest	0.99	-38.45 (руб/акция)	0.599	-70.206 (руб/долл)
B&H	0	-50.65 (руб/акция)	0	5.94 (руб/долл)

Перспективы

В будущем планируется улучшить алгоритм подбора параметров (бэктэстить больше параметров для оптимизации алгоритмов), а также сравнить с рынком криптовалют. Следующим пунктом будет добавления для сравнения алгоритмы, описанные раннее в введении: торговля на новостях, другие алгоритмы из машинного обучения и более продвинутых алгоритмов технического анализа.