cross_validation

December 11, 2019

https://github.com/QuantCS109/TrumpTweets/blob/master/notebooks_modelling/cross_validation.ipynb

[1]:
```
import sys
sys.path.append('..')
from modules import opts
```

[2]:
```
from __future__ import absolute_import

import pickle
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit

from sklearn.metrics import r2_score, accuracy_score, f1_score, log_loss

from tqdm import tqdm

import sys
sys.path.append('..')
from modules.project_helper import VolFeatures, FuturesCloseData, TradeModel

import warnings
warnings.filterwarnings('ignore')
```

0.1 Cross Validation for Finance

Since we have one trading model per asset, we require choosing hyper-parameters for 13 different assets.

Cross validation in finance is tricky. You can't just shuffle your data randomly to obtain a fold, since you are dealing with time series. The independent and identically distributed assumption does not hold well to time series data. There will be serial correlation between data points, and trying to predict a particular point with information in the training set coming before and after in time will lead to leakage and inflation of performance results.

We choose to follow a time series approach. We do 5-fold time series cross validation, as we would test in a real trading situation.

For Logistic Regression, we search for the C parameter in in the range:

C =[0.001, 0.01, 0.1,1,10,100,1000,10000,100000] For both Random Forest and Gradient Boosting, we perform a grid search in the ranges:

depth_list = [5, 6, 7, … ,15] max_features_list = [3, 4, 5, … 25] In his book Advances in Financial Machine Learning, Marcos Lopez de Prado suggests we use negative log-loss, or cross entropy, as the measure to focus on for hyper parameter tuning. From (2) pg. 130:

"…accuracy accounts equally for an erroneous buy prediction with high probability and for an erroneous buy prediction with low probability… Investment strategies profit from predicting the right label with high confidence. Gains from good predictions with low confidence will not suffice to offset the losses from bad predictions with high confidence."

Cross entropy is the log-likelihood of the classifier given the true label, which takes prediction's probabilities into account.

Took 5 hours in my computer to complete the cross-validation!

[3]:
```
import pickle
file = open("../data/features/full_features.pkl",'rb')
full_features = pickle.load(file)
```

[8]:
```
x_dict['C'].index[-1]
```

[8]:
```
Timestamp('2019-11-07 00:00:00')
```

[4]:
```
instrument_list = ['ES', 'NQ', 'CD', 'EC', 'JY', 'MP', 'TY', 'US', 'C', 'S',␣
 ↪'W', 'CL', 'GC']
x_dict={}
y_dict={}
for inst in instrument_list:
    y_dict[inst] = (full_features[inst][inst]>=0).astype(int)
    x_dict[inst] = full_features[inst].drop([inst], axis=1)
```

[141]:
```
instrument_list = ['ES', 'NQ', 'CD', 'EC', 'JY', 'MP', 'TY', 'US', 'C', 'S',␣
 ↪'W', 'CL', 'GC']

C_list = [0.001, 0.01, 0.1,1,10,100,1000,10000,100000]
accuracy_logreg = pd.DataFrame(columns=C_list, index=instrument_list)
f1score_logreg = pd.DataFrame(columns=C_list, index=instrument_list)
logloss_logreg = pd.DataFrame(columns=C_list, index=instrument_list)
plong_logreg = pd.DataFrame(columns=C_list, index=instrument_list)
```

```
n_splits = 5
for inst in instrument_list:
    for c in C_list:

        X, X_test, y, y_test = train_test_split(x_dict[inst], y_dict[inst],␣
 ↪test_size=0.20, shuffle=False)
        X_test = X_test[2:]
        y_test = y_test[2:]

        tm = TradeModel(model=LogisticRegression, C=c)
        tscv = TimeSeriesSplit(n_splits=n_splits)
        time_split = tscv.split(X)
        ac = 0
        f1 = 0
        p = 0
        ll = 0
        for train_index, valid_index in time_split:
            X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
            y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]
            X_valid = X_valid[2:]
            y_valid = y_valid[2:]
            tm.fit(X_train, y_train)
            ac = ac + tm.model.score(X_valid, y_valid)
            f1 = f1 + f1_score(tm.model.predict(X_valid),y_valid)
            p = p + tm.model.predict(X_valid).mean()
            ll = ll + log_loss(y_valid, tm.model.predict(X_valid))

        accuracy_logreg.loc[inst][c] = ( round(ac / n_splits, 3) )
        f1score_logreg.loc[inst][c] = ( round(f1 / n_splits, 3) )
        plong_logreg.loc[inst][c] = ( round(p / n_splits, 3) )
        logloss_logreg.loc[inst][c] = ( round(ll / n_splits, 3) )
    cv_logreg = logloss_logreg.astype('float').idxmin(axis=1)
```

[150]:
```
logloss_logreg
```

[150]:

	0.001	0.010	0.100	1.000	10.000	100.000	\
ES	15.789	15.03	14.727	16.169	16.548	16.093	
NQ	16.093	14.954	15.106	15.562	15.638	15.258	
CD	16.473	16.245	16.928	18.446	19.129	18.826	
EC	17.535	17.232	17.232	16.473	17.383	16.776	
JY	15.638	16.473	16.473	16.928	16.852	16.852	
MP	17.08	16.928	17.004	17.156	16.7	16.093	
TY	13.816	14.651	15.638	16.548	16.548	17.308	
US	15.03	16.017	16.473	16.928	17.232	17.156	

```
C       16.093     15.562     15.941     16.321     16.017     16.321
S       17.308      17.08     16.852     16.776      17.08      17.08
W       16.852       16.7     17.156     16.852     16.472      17.08
CL      16.093     16.169     15.865     15.941     17.232     17.156
GC      17.687     17.156     16.852       16.7     16.093     15.941

   1000.000   10000.000  100000.000
ES      15.638     16.017     16.245
NQ      15.486     16.245     17.004
CD      18.522      18.75     19.053
EC      17.232     17.687     17.156
JY      16.624     16.624     16.321
MP      16.548     15.941     15.941
TY      17.156     16.928     17.004
US      17.763     17.535     17.991
C       16.245     16.397     16.321
S       17.232     17.308     17.535
W       16.624     16.624      17.08
CL      17.232     17.763     17.687
GC      16.093     15.865     16.548
```

[164]:
```
cv_logreg
```

[164]:
```
ES        0.100
NQ        0.010
CD        0.010
EC        1.000
JY        0.001
MP    10000.000
TY        0.001
US        0.001
C         0.010
S         1.000
W        10.000
CL        0.100
GC    10000.000
dtype: float64
```

[163]:
```
logloss_logreg
```

[163]:
```
   0.001       0.010       0.100       1.000       10.000      100.000     \
ES      15.789      15.03     14.727     16.169     16.548     16.093
NQ      16.093     14.954     15.106     15.562     15.638     15.258
CD      16.473     16.245     16.928     18.446     19.129     18.826
EC      17.535     17.232     17.232     16.473     17.383     16.776
JY      15.638     16.473     16.473     16.928     16.852     16.852
MP       17.08     16.928     17.004     17.156       16.7     16.093
```

```
TY      13.816      14.651      15.638      16.548      16.548      17.308
US       15.03      16.017      16.473      16.928      17.232      17.156
C       16.093      15.562      15.941      16.321      16.017      16.321
S       17.308       17.08      16.852      16.776       17.08       17.08
W       16.852        16.7      17.156      16.852      16.472       17.08
CL      16.093      16.169      15.865      15.941      17.232      17.156
GC      17.687      17.156      16.852        16.7      16.093      15.941

   1000.000   10000.000  100000.000
ES      15.638      16.017      16.245
NQ      15.486      16.245      17.004
CD      18.522       18.75      19.053
EC      17.232      17.687      17.156
JY      16.624      16.624      16.321
MP      16.548      15.941      15.941
TY      17.156      16.928      17.004
US      17.763      17.535      17.991
C       16.245      16.397      16.321
S       17.232      17.308      17.535
W       16.624      16.624       17.08
CL      17.232      17.763      17.687
GC      16.093      15.865      16.548
```

[171]:
```python
with open('cv_logreg.pickle', 'wb') as handle:
    pickle.dump(accuracy_logreg, handle)
    pickle.dump(f1score_logreg, handle)
    pickle.dump(plong_logreg, handle)
    pickle.dump(logloss_logreg, handle)
    pickle.dump(cv_logreg, handle)
```

[146]:
```python
depth_list = range(4,15)
max_features_list = list(range(3,7,1)) + list(range(7,26,3))
accuracies_rf = {inst:pd.DataFrame(columns=max_features_list, index=depth_list) for inst in instrument_list}
f1scores_rf = {inst:pd.DataFrame(columns=max_features_list, index=depth_list) for inst in instrument_list}
logloss_rf = {inst:pd.DataFrame(columns=max_features_list, index=depth_list) for inst in instrument_list}
plong_rf = {inst:pd.DataFrame(columns=max_features_list, index=depth_list) for inst in instrument_list}
cv_rf = pd.DataFrame(index = instrument_list, columns = ['max_depth','max_features'])
n_splits = 5
for inst in tqdm(instrument_list):
    for dl in depth_list:
        for mf in max_features_list:
```

```
            X, X_test, y, y_test = train_test_split(x_dict[inst], y_dict[inst],␣
↪test_size=0.20, shuffle=False)
            tm = TradeModel(n_estimators=1000, max_features=mf, max_depth=dl,␣
↪criterion ='entropy')

            tscv = TimeSeriesSplit(n_splits=n_splits)
            time_split = tscv.split(X)
            ac = 0
            f1 = 0
            p = 0
            ll = 0
            for train_index, valid_index in time_split:
                X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
                y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]
                X_valid = X_valid[2:]
                y_valid = y_valid[2:]
                tm.fit(X_train, y_train)
                ac = ac + tm.model.score(X_valid, y_valid)
                f1 = f1 + f1_score(tm.model.predict(X_valid),y_valid)
                p = p + tm.model.predict(X_valid).mean()
                ll = ll + log_loss(y_valid, tm.model.predict(X_valid))

            accuracies_rf[inst].loc[dl,mf] = round(ac / n_splits,3)
            f1scores_rf[inst].loc[dl,mf] = round(f1 / n_splits,3)
            plong_rf[inst].loc[dl,mf] = ( round(p / n_splits, 3) )
            logloss_rf[inst].loc[dl,mf] = ( round(ll / n_splits, 3) )

    x1 = logloss_rf[inst].astype('float').min(axis=1).idxmin()
    x2 = logloss_rf[inst].astype('float').loc[x1].idxmin()
    cv_rf.loc[inst] = np.array([x1,x2])
```

```
0%|          | 0/13 [00:00<?, ?it/s]
```

```
 8%|          | 1/13 [16:42<3:20:33, 1002.76s/it]
```

```
15%|          | 2/13 [33:27<3:03:56, 1003.32s/it]
```

```
23%|          | 3/13 [50:20<2:47:43, 1006.35s/it]
```

```
31%|          | 4/13 [1:07:07<2:30:58, 1006.53s/it]
```

```
38%|          | 5/13 [1:24:59<2:16:47, 1025.96s/it]
```

```
46%|          | 6/13 [1:42:46<2:01:09, 1038.44s/it]
```

```
54%|          | 7/13 [2:04:40<1:52:06, 1121.08s/it]
```

```
62%|         | 8/13 [2:25:35<1:36:46, 1161.27s/it]
```

```
69%|         | 9/13 [2:42:38<1:14:39, 1119.90s/it]
```

```
77%|        | 10/13 [2:59:49<54:38, 1092.98s/it]
```

```
85%|       | 11/13 [3:17:01<35:49, 1074.78s/it]
```

```
 92%|      | 12/13 [3:33:53<17:35, 1055.99s/it]
```

```
100%|      | 13/13 [3:50:40<00:00, 1064.62s/it]
```

[181]:
```
cv_rf
```

[181]:

	max_depth	max_features
ES	4	13
NQ	4	3
CD	13	6
EC	12	6
JY	4	22
MP	8	19
TY	7	13
US	5	3
C	8	19
S	13	10
W	4	3
CL	4	4
GC	4	3

[151]:
```
logloss_rf['ES']
```

[151]:

	3	4	5	6	7	10	13	16	19 \
4	14.803	15.182	14.954	14.727	14.651	14.803	14.423	14.803	14.879
5	14.879	14.879	14.727	15.03	14.954	14.803	14.651	14.954	15.03
6	15.182	14.954	15.106	15.41	14.727	15.258	14.954	15.334	15.258
7	14.879	15.106	14.954	14.954	14.651	15.334	15.41	15.182	14.954
8	15.182	15.182	15.182	15.03	15.106	15.334	15.562	15.106	15.41
9	14.879	14.879	14.879	15.182	14.803	15.334	15.182	15.334	15.03
10	15.334	15.258	15.258	15.334	14.879	14.954	15.106	15.486	14.954
11	15.03	14.954	15.486	14.954	14.878	15.41	15.258	15.334	15.41
12	15.182	15.182	14.954	15.182	15.41	15.03	15.03	15.182	15.41
13	15.562	15.258	15.562	15.03	15.258	15.638	15.334	15.258	14.954

```
14  15.182  15.182  14.954  14.879  15.334  15.258   15.41  15.258  15.486

        22      25
4   14.803  14.651
5   15.258  14.954
6   15.182  15.334
7   15.258  15.106
8   15.713  15.258
9   14.803  15.258
10  15.258  15.182
11  15.334  14.878
12  15.258  15.182
13   15.03  15.258
14  15.334  15.258
```

[182]:

```
with open('cv_rf.pickle', 'wb') as handle:
    pickle.dump(accuracies_rf, handle)
    pickle.dump(f1scores_rf, handle)
    pickle.dump(plong_rf, handle)
    pickle.dump(logloss_rf, handle)
    pickle.dump(cv_rf, handle)
```

[145]:

```
depth_list = range(4,15)
max_features_list = list(range(3,7,1)) + list(range(7,26,3))
accuracies_boost = {inst:pd.DataFrame(columns=max_features_list,␣
 ↪index=depth_list) for inst in instrument_list}
f1scores_boost = {inst:pd.DataFrame(columns=max_features_list,␣
 ↪index=depth_list) for inst in instrument_list}
logloss_boost = {inst:pd.DataFrame(columns=max_features_list, index=depth_list)␣
 ↪for inst in instrument_list}
plong_boost = {inst:pd.DataFrame(columns=max_features_list, index=depth_list)␣
 ↪for inst in instrument_list}
cv_boost = pd.DataFrame(index = instrument_list, columns =␣
 ↪['max_depth','max_features'])
n_splits = 5
for inst in tqdm(instrument_list):
    for dl in depth_list:
        for mf in max_features_list:

            X, X_test, y, y_test = train_test_split(x_dict[inst], y_dict[inst],␣
 ↪test_size=0.20, shuffle=False)
            tm = TradeModel(model=GradientBoostingClassifier,
                            n_estimators=1000,
                            max_features=mf,
                            max_depth=dl,
                            )
```

```
        tscv = TimeSeriesSplit(n_splits=n_splits)
        time_split = tscv.split(X)
        ac = 0
        f1 = 0
        p = 0
        ll = 0
        for train_index, valid_index in time_split:
            X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
            y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]
            X_valid = X_valid[2:]
            y_valid = y_valid[2:]
            tm.fit(X_train, y_train)
            ac = ac + tm.model.score(X_valid, y_valid)
            f1 = f1 + f1_score(tm.model.predict(X_valid),y_valid)
            p = p + tm.model.predict(X_valid).mean()
            ll = ll + log_loss(y_valid, tm.model.predict(X_valid))

        accuracies_boost[inst].loc[dl,mf] = round(ac / n_splits,3)
        f1scores_boost[inst].loc[dl,mf] = round(f1 / n_splits,3)
        plong_boost[inst].loc[dl,mf] = ( round(p / n_splits, 3) )
        logloss_boost[inst].loc[dl,mf] = ( round(ll / n_splits, 3) )

x1 = logloss_boost[inst].astype('float').min(axis=1).idxmin()
x2 = logloss_boost[inst].astype('float').loc[x1].idxmin()
cv_boost.loc[inst] = np.array([x1,x2])
```

```
0%|          | 0/13 [00:00<?, ?it/s]
```

```
8%|          | 1/13 [04:11<50:16, 251.36s/it]
```

```
15%|          | 2/13 [08:24<46:12, 252.01s/it]
```

```
23%|          | 3/13 [12:40<42:10, 253.03s/it]
```

```
31%|          | 4/13 [22:20<52:40, 351.20s/it]
```

```
38%|          | 5/13 [26:40<43:09, 323.72s/it]
```

```
46%|          | 6/13 [30:55<35:22, 303.22s/it]
```

```
54%|          | 7/13 [38:17<34:28, 344.75s/it]
```

```
62%|          | 8/13 [42:48<26:53, 322.74s/it]
```

```
69%|          | 9/13 [47:10<20:17, 304.47s/it]
```

```
77%|          | 10/13 [51:28<14:31, 290.51s/it]
```

```
85%|          | 11/13 [55:49<09:23, 281.84s/it]
```

```
 92%|      | 12/13 [1:00:19<04:38, 278.06s/it]
```

```
100%|      | 13/13 [1:04:35<00:00, 298.14s/it]
```

[159]:
```
cv_boost
```

[159]:

	max_depth	max_features
ES	6	16
NQ	14	5
CD	8	6
EC	11	3
JY	13	16
MP	11	22
TY	8	16
US	11	16
C	11	5
S	6	10
W	12	6
CL	5	22
GC	11	25

[161]:
```
logloss_boost['C']
```

[161]:

	3	4	5	6	7	10	13	16	19 \
4	16.017	16.624	15.486	15.713	16.017	16.093	16.7	16.624	16.245
5	16.624	16.7	16.245	16.624	16.852	16.093	15.941	17.004	16.473
6	16.321	16.017	16.017	15.865	16.321	16.093	15.941	16.7	16.397
7	16.397	16.017	16.928	15.789	16.928	15.638	16.245	16.473	16.7
8	16.245	16.093	16.7	16.093	17.156	16.7	16.624	16.7	15.865
9	16.548	16.624	16.093	16.624	15.258	16.397	16.321	16.397	16.624
10	17.156	15.41	15.865	16.017	16.245	16.169	16.852	16.321	15.789
11	16.548	17.535	15.182	16.624	16.852	16.852	16.928	15.941	16.928
12	16.7	16.548	17.156	16.093	16.776	16.473	16.017	15.789	16.093
13	16.548	15.258	16.548	16.321	16.397	17.232	16.548	16.017	17.08
14	16.852	16.548	16.169	16.093	16.928	16.776	16.624	16.7	15.865

	22	25
4	16.7	16.397

```
5   16.928  16.776
6   16.548  16.093
7   16.852  16.776
8   16.473  15.713
9   16.852  16.245
10  16.017  16.473
11  16.624  16.473
12  16.093  16.397
13  16.852  15.334
14  16.245  16.245
```

[173]:
```
with open('cv_boost.pickle', 'wb') as handle:
    pickle.dump(accuracies_boost, handle)
    pickle.dump(f1scores_boost, handle)
    pickle.dump(plong_boost, handle)
    pickle.dump(logloss_boost, handle)
    pickle.dump(cv_boost, handle)
```