2021_04_05

April 7, 2021

0.0.1 Information Trading

with the idea of Probability of Informed Trading (Easely and O'Hara, 1992), there is a chance α of an information happening and δ chance this is a good news. An informed trading associated with this event occurs with the probability μ (or arrival rate). An uninformed trading irrespective of informational event occurs with the probability of ϵ (or arrival rate)

This flow toxicity can be quantified by **Probability of Informed Trading**

$$\textbf{PIN} = \frac{\alpha\mu}{\alpha\mu + 2\epsilon}$$

[1]:
```
import itertools
import numpy as np
import datetime
import pandas as pd
from IPython.core.debugger import Tracer
import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from scipy.stats.stats import pearsonr
```

0.0.2 Split Data by Its Ticker/Sym_root

- should be written in back-end script
- further data clean process is required

[82]:
```
import csv
```

```
with open('data/vfin_bank/dailyquote.csv') as fin:
    csvin = csv.DictReader(fin)
    outputs = {}
    for row in csvin:
        cat = row['SYM_ROOT']
        if cat not in outputs:          # Open a new file and write the header
            fout = open('data/vfin_bank/quote_{}.csv'.format(cat), 'w')
            dw = csv.DictWriter(fout, fieldnames=csvin.fieldnames)
            dw.writeheader()
            outputs[cat] = fout, dw
        outputs[cat][1].writerow(row)  # Always write the row
    for fout, _ in outputs.values():
        fout.close()                   # Close all the files
```

[83]:
```
!du -h data/vfin_bank/*
```

```
24K     data/vfin_bank/AvgCorrAssets.csv
80K     data/vfin_bank/AvgCorrEx.csv
48K     data/vfin_bank/CDF.csv
26M     data/vfin_bank/dailyquote.csv
14M     data/vfin_bank/dailytrade.csv
4.6M    data/vfin_bank/quote_BAC.csv
8.8M    data/vfin_bank/quote_C.csv
6.4M    data/vfin_bank/quote_JPM.csv
4.3M    data/vfin_bank/quote_USB.csv
2.9M    data/vfin_bank/quote_WFC.csv
3.1M    data/vfin_bank/trade_BAC.csv
4.5M    data/vfin_bank/trade_C.csv
2.9M    data/vfin_bank/trade_JPM.csv
1.8M    data/vfin_bank/trade_USB.csv
1.9M    data/vfin_bank/trade_WFC.csv
```

[3]:
```
sym = ['C','BAC','USB','JPM','WFC']
df={}; sec_trades = {}
for s in sym:
    f = 'data/vfin_bank/trade_%s.csv' %s
    sec_trades[s] = pd.read_csv(f,parse_dates=[['DATE','TIME_M']],␣
 ↪index_col=[0],
                                usecols=['DATE','TIME_M','EX','SIZE','PRICE'])
```

0.0.3 VPIN and CDF

Easley, Prado and O'Hara (2012) extended the logic behind PIN to Volume Synchronized PIN (VPIN). It is assumed that people trade more when informational events are more relevant.

- to devide trades in equal information windows, volume bucketing (fixed @ 50 time units) is used. One can also try time windows with fixed trade volume that carry equal amount of information.

- PIN is hard to estimate since it depends on unobserved parameters that are hard to measure in practice. However, VPIN can be estimated using just the trade data.

The direction of volume change indicates the type of information - if good news are about to occur, we expect to see more buy volume than sell volume - if bad news are about to occur, we expect to see less buy volume than sell volume

However TAQ doesn't supply trade direction signals. I believe the most popular way to address this problem is to use change of price over a time window to infer relative trade direction. - increase in the price will indicate higher buy volume - decrease in the price will indicate higher sell volume - the gap between buy and sell volume will be higher if we observe a significant price change

Since a 50 time units long volume bucket is used, checking only start and end price will ignore the price path. - divide each window in small time invervals and divide total volume in each small intervals - aggregate trade signals on all small intervals. A z score (volume distribution) is guided to split buy/sell volumes

The difference between buy and sell volumes (i.e., the trade imbalance) estimate $\alpha\mu$, VPIN is updated for every new volume bucket (to start with n of course). While studying E-mini S&P 500 future around the 2010 flash crash, Easley, Prado and O'Hara (2012) observe the high Cumulative Distribution Density (CDF) of VPIN hours before the crash.

[111]:
```
from importlib import reload
import vwap
reload(vwap.vpin)
```

[111]:
```
<module 'vwap.vpin' from
'C:\\Users\\denni\\thesis_2021\\thesis_impact\\reports\\2021_04\\vwap\\vpin.py'>
```

[17]:
```
from vwap.vpin import calc_vpin

volume = {'C':300000,'USB':100000,'JPM':200000,'BAC':1250000,'WFC':300000}
for s in sym:
    print('Calculating VPIN on %s' %s)
    df[s] = calc_vpin(sec_trades[s],volume[s],50)
avg = pd.DataFrame()
metric = 'CDF'
avg[metric] = np.nan
for stock,frame in df.items():
    frame = frame[[metric]].reset_index().drop_duplicates(subset='Time',␣
 ↪keep='last').set_index('Time')
    avg = avg.
 ↪merge(frame[[metric]],left_index=True,right_index=True,how='outer',suffixes=('',stock))
avg = avg.dropna(axis=0,how='all').fillna(method='ffill')
avg.to_csv('data/vfin_bank/CDF.csv')
```

```
Calculating VPIN on C
Calculating VPIN on BAC
Calculating VPIN on USB
```

```
Calculating VPIN on JPM
Calculating VPIN on WFC
```

0.0.4 calculate pair-wise correlation

To measure vpin of similar stocks (bank securities), we compare average rolling correlation of CDF across these assets

[37]:
```
fields = ['Time','CDFC','CDFBAC','CDFUSB','CDFJPM','CDFWFC']
df = pd.read_csv('data/vfin_bank/CDF.
 ↪csv',parse_dates=['Time'],index_col=[0],usecols = fields)
rolling_pariwise_corr = df.rolling(50).corr(pairwise=True)
thres = pd.DataFrame()
thres['AvgCorrAssets'] = rolling_pariwise_corr.groupby(by=['Time']).sum().
 ↪sum(axis=1)/((len(fields)-1)**2)
thres.to_csv('data/vfin_bank/AvgCorrAssets.csv')
```

[77]:
```
df
```

[77]:

Time	CDFC	CDFBAC	CDFUSB	CDFJPM	CDFWFC
2005-01-03 13:37:00	NaN	NaN	NaN	0.393805	NaN
2005-01-03 13:45:00	NaN	NaN	NaN	0.442478	NaN
2005-01-03 14:01:00	NaN	NaN	NaN	0.517699	NaN
2005-01-03 14:08:00	NaN	NaN	NaN	0.597345	NaN
2005-01-03 14:09:00	NaN	NaN	NaN	0.668142	NaN
…	…	…	…	…	…
2005-01-07 15:58:00	0.617512	NaN	0.975248	0.106195	0.277778
2005-01-07 16:00:00	0.617512	NaN	0.836634	0.106195	0.277778
2005-01-07 16:01:00	0.617512	NaN	0.836634	0.053097	0.277778
2005-01-07 16:05:00	0.617512	NaN	0.836634	0.075221	0.277778
2005-01-07 16:08:00	0.585253	NaN	0.836634	0.075221	0.277778

[546 rows x 5 columns]

0.0.5 calculate information correlation between exchanges example: CITI stock

To measure VPIN relation across exchanges, we compare average rolling correlation of CDF across these exchanges while focusing on one stock

[85]:
```
fields = ['DATE','TIME_M','EX','SIZE','PRICE']
CITI = pd.read_csv('data/vfin_bank/trade_C.
 ↪csv',parse_dates=[['DATE','TIME_M']], index_col=[0], usecols=fields)
exchanges = CITI['EX'].unique(); exchanges = [x for x in exchanges if not x in␣
 ↪('C','X')]
```

[86]:
```
df_lst, df_vpin, vol_ex = [], [], []; n = len(exchanges)
for i in range(n):
```

```
    df_list.append(CITI[CITI['EX'] == exchanges[i]])
for i in range(n):
    vol_ex.append(df_list[i]['SIZE'].sum())
nbuckets = 6574
for i in range(n):
    bucket = int(vol_ex[i]/nbuckets)
    df_vpin.append(calc_vpin(df_list[i],bucket,50))
avg = pd.DataFrame(); metric = 'VPIN_'; avg[metric] = np.nan
for i in range(n):
    frame = df_vpin[i][[metric]].reset_index().drop_duplicates(subset='Time',␣
 ↪keep='last').set_index('Time')
    avg = avg.
 ↪merge(frame[[metric]],left_index=True,right_index=True,how='outer',suffixes=('',exchanges[i
avg = avg.dropna(axis=0,how='all').fillna(method='ffill')
del avg['VPIN']; avg = avg.dropna()
rolling_pariwise_corr = avg.rolling(50).corr(pairwise=True)
thres = pd.DataFrame()
thres['AvgCorrEx'] = rolling_pariwise_corr.groupby(by=['Time']).sum().
 ↪sum(axis=1)/(len(exchanges)**2)
thres.to_csv('data/vfin_bank/AvgCorrEx.csv')
```

[87]:
```
avg
```

[87]:

Time	VPINN	VPINM	VPINB	VPINT	VPINP
2005-01-03 09:36:00	0.684163	0.006207	0.692347	0.000000	0.636856
2005-01-03 09:37:00	0.704467	0.039655	0.692347	0.000000	0.563120
2005-01-03 09:38:00	0.716507	0.069807	0.706516	0.000000	0.586964
2005-01-03 09:39:00	0.730929	0.069807	0.740923	0.000000	0.586964
2005-01-03 09:40:00	0.728293	0.083953	0.760402	0.000000	0.591919
…	…	…	…	…	…
2005-01-07 16:09:00	0.000000	0.499014	0.711834	0.606546	0.556566
2005-01-07 16:11:00	0.000000	0.499014	0.718601	0.606546	0.556566
2005-01-07 16:12:00	0.000000	0.499014	0.720120	0.606546	0.556566
2005-01-07 16:17:00	0.000000	0.499014	0.775379	0.606546	0.556566
2005-01-07 16:21:00	0.000000	0.499014	0.704676	0.606546	0.556566

[1984 rows x 5 columns]

0.0.6 Calculate Quote Imbalance example: CITI stock

VPIN and its CDF are good at quantifying toxicity in the market. However they both are on absolute scale, which fails to give direction of impending market movements. Market makers tend to provide more liquidity on the side they expect the market to move. - if price is expected to rise, more liquidity is supplied on the bid side before price increases - if price is expected to drop, more liquidity is supplied on the ask side before price decreases

Quote Imbalance tries to quantify the difference in bid and ask quote volume. Most of the times,

price change takes place across all exchanges and stocks belonging to one industry tend to move together. Stock moves in a seperate direction only when news are specific to this stock (e.g., earning calls, change of ceo, etc.).

[112]:
```
from vwap.vpin import calc_imbalance

fields = ['DATE','TIME_M','EX','BID','BIDSIZ','ASK','ASKSIZ']
CITI = pd.read_csv('data/vfin_bank/quote_C.
 ↪csv',parse_dates=[['DATE','TIME_M']], index_col=[0], usecols=fields)
quote_imb, quote_price = calc_imbalance(CITI)
quote_imb.to_csv('data/vfin_bank/imb_C.csv'); quote_price.to_csv('data/
 ↪vfin_bank/price_C.csv')
```

[108]:
```
quote_imb[100:200]
```

[108]:
```
DATE_TIME_M
2005-01-03 10:40:00    11.852992
2005-01-03 10:41:00    74.140036
2005-01-03 10:42:00   -36.909339
2005-01-03 10:43:00   -14.880703
2005-01-03 10:44:00    10.990525
                          …
2005-01-03 12:15:00    -5.981021
2005-01-03 12:16:00   -16.902190
2005-01-03 12:17:00     1.111718
2005-01-03 12:18:00   -49.828659
2005-01-03 12:19:00    -4.938797
Freq: T, Length: 100, dtype: float64
```

0.0.7 algorithm trading on the basis of adversarial information

Assuming a good news about Citi bank is about to release, the informed traders submit a large number of buy orders which will push the VPIN higher. The market makers, anticipating this trend, will change the supply of liquidity. They do so by posting high bid quote volume to secure a net long position. Therefore, combining quote imbalance and VPIN can provide a good short-term buy/sell signal.

There are a couple of caveats in deriving this signal

- First, we must make sure high VPIN does not arise due to liquidity bottlenecks or arbitrage midconducts. To make sure the VPIN signal is indeed a signal of high information flow, we use the VPIN CDF average correlation metric between exchanges to validate a pan-exchange phenomenon
- Second, high VPIN can also arise due to stock-specific structural aspects that are not representative of an increase in flow toxicity. To isolate industry wide information signals, we screen out VPIN signals with low industry-wide VPIN average correlations

The directional information of the asset is given by the quote imbalance, which reports the average difference between bid volume and ask volume across exchanges.

[129]:
```
DATA_DIR = 'data/vfin_bank/'
df_corr_assets = pd.read_csv(DATA_DIR + 'AvgCorrAssets.
 ↪csv',parse_dates=['Time'],index_col='Time')
df_corr_ex = pd.read_csv(DATA_DIR + 'AvgCorrEx.
 ↪csv',parse_dates=['Time'],index_col='Time')
df_vpin = pd.read_csv(DATA_DIR + 'CDF.
 ↪csv',parse_dates=['Time'],usecols=['Time','CDFC'],index_col='Time')
df_quote_imb = pd.read_csv(DATA_DIR+'imb_C.
 ↪csv',parse_dates=['DATE_TIME_M'],index_col='DATE_TIME_M')
df_price = pd.read_csv(DATA_DIR+'price_C.
 ↪csv',parse_dates=['DATE_TIME_M'],index_col='DATE_TIME_M')
```

[136]:
```
total_df = pd.DataFrame()
frame = df_corr_assets[[df_corr_assets.columns[0]]].reset_index().
 ↪drop_duplicates(subset='Time', keep='last').set_index('Time')
total_df = total_df.merge(frame[[frame.
 ↪columns[0]]],left_index=True,right_index=True,how='outer')
print (total_df.shape)

frame = df_corr_ex[[df_corr_ex.columns[0]]].reset_index().
 ↪drop_duplicates(subset='Time', keep='last').set_index('Time')
total_df = total_df.merge(frame[[frame.
 ↪columns[0]]],left_index=True,right_index=True,how='outer')
print (total_df.shape)

frame = df_vpin[[df_vpin.columns[0]]].reset_index().
 ↪drop_duplicates(subset='Time', keep='last').set_index('Time')
total_df = total_df.merge(frame[[frame.
 ↪columns[0]]],left_index=True,right_index=True,how='outer')
print (total_df.shape)

frame = df_quote_imb[[df_quote_imb.columns[0]]].reset_index().
 ↪drop_duplicates(subset='DATE_TIME_M', keep='last').rename(columns =␣
 ↪{'DATE_TIME_M':'Time'}).set_index('Time')
total_df = total_df.merge(frame[[frame.
 ↪columns[0]]],left_index=True,right_index=True,how='outer')
print (total_df.shape)

frame = df_price[[df_price.columns[0]]].reset_index().
 ↪drop_duplicates(subset='DATE_TIME_M', keep='last').rename(columns =␣
 ↪{'DATE_TIME_M':'Time'}).set_index('Time')
total_df = total_df.merge(frame[[frame.
 ↪columns[0]]],left_index=True,right_index=True,how='outer')
print (total_df.shape)

total_df = total_df.dropna(axis=0,how='all').fillna(method='ffill')
```

```
total_df = total_df.dropna(how='any')
print (total_df.shape)
total_df.columns = ['assets','ex', 'vpin', 'imb', 'price']; total_df.
 ↪to_csv(DATA_DIR+'total_C.csv')
```

```
(546, 1)
(1990, 2)
(1990, 3)
(3093, 4)
(3093, 5)
(1623, 5)
```

0.0.8 Data Preprocessing

- read price data -> get return value of each time interval (1 min frequency)
- filter the data between 9:00am and 4:00 pm since VPIN metrics are updated very slowly before open and after close
- calculate z-scores using a window of previous 100 mins making sure that there is no look ahead and cap each score to be within the range of -3 to 3
- when CDF > 0.4 and high correlation among assets and exchanges are observed, it is quite likely to see a flow toxicity signal.
- if a higher enough quote imbalance is also present, we maintain a long position within 10 periods (i.e., 10 mins, tracked by `bcount` and `scount`) if there are no further buy signals. We treat the short positions in a similar manner.

[152]:
```
df= pd.read_csv(DATA_DIR+'total_C.csv', parse_dates=['Time'],index_col='Time' )
df['returns']=df['price'].pct_change(); df['returns']=df['returns'].shift(-1)
df=df.replace([np.inf, -np.inf], np.nan)
df=df.ffill()
df=df.between_time('9:00','16:00')
df=df[df.index.dayofweek < 5]
df=df[df.index<pd.to_datetime('2007-12-30')]
savg=df[['assets','ex','imb']].rolling(window=100).mean()
sstd=df[['assets','ex','imb']].rolling(window=100).std()
df['ex']=((df['ex']-savg['ex'])/sstd['ex']).clip(-3,3)
df['assets']=((df['assets']-savg['assets'])/sstd['assets']).clip(-3,3)
df['imb']=((df['imb']-savg['imb'])/sstd['imb']).clip(-3,3)
df = df.iloc[100:]; df.to_csv(DATA_DIR+'final_C.csv')
```

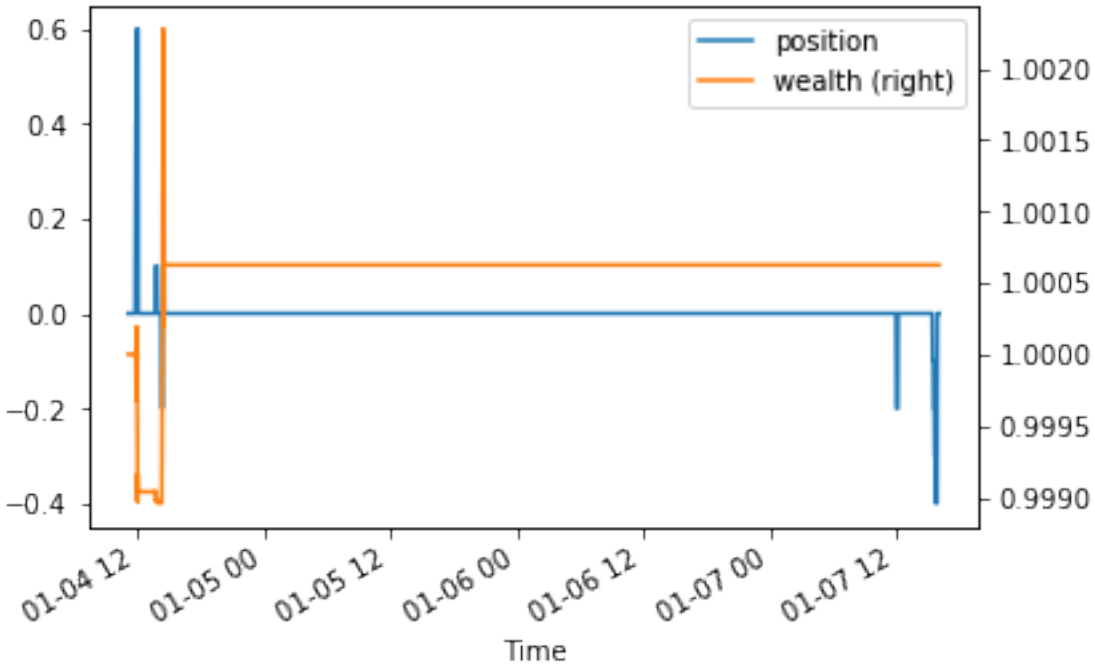
[174]:
```
pos=pd.DataFrame(columns=['position','returns','Time'])
position=0

for index,row in df.iterrows():
    if(row['vpin']>0.4 and (row['ex']+row['assets'])>3):
        if(row['imb']>1.5):
            if(position<0):  position=0
            position+=0.1; bcount=10
        if(row['imb']<-1.5):
```

```
            if(position>0):  position=0
            position-=0.1; scount=10
    if(position>0):
        bcount=bcount-1
        if(bcount==0): position=0
    elif(position<0):
        scount=scount-1
        if(scount==0): position=0
    pos=pos.append({'position':position,'returns':row['returns'], 'price':
 ↪row['price'],'Time':index},ignore_index=True)
pos=pos.set_index('Time')
pos['wealth']=(1+pos['returns']*pos['position']).cumprod()
```

[175]:
```
pos[['position','wealth']].plot(secondary_y=['wealth'])
```

[175]: <matplotlib.axes._subplots.AxesSubplot at 0x1f4db7c1370>



0.0.9 Machine Learning Model

[189]:
```
df= pd.read_csv(DATA_DIR+ 'final_C.csv', parse_dates=['Time'],index_col='Time' )
df=df[(df.assets.notnull()) & (df.imb.notnull())]
X = df[['vpin','assets','ex','imb']]; y=(df['returns']>0).astype(int)
size = int(len(df) * 0.85)
```

```
X_train, y_train = X[0:size], y[0:size]
X_test, y_test = X[size:], y[size:]
print(f"""
train data size: {len(y_train)}
test data size : {len(y_test)}
""")
```

```
train data size: 544
test data size : 97
```

[209]:
```
import gc


score=[]; pos=None
classifiers = [
    KNeighborsClassifier(4),
    SVC(kernel="linear", C=0.025,probability=True),
    SVC(gamma=0.25, C=1,probability=True),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=4),
    MLPClassifier(alpha=1),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]

names = ["Nearest Neighbors", "Linear SVM", "RBF SVM",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

fig, axes = plt.subplots(nrows=3, ncols=3,␣
 ↪figsize=(12,12),sharex=True,sharey=True)
fig.subplots_adjust(hspace=0.5)
fig.suptitle('Change of Position/Wealth and Price Dynamics')

for i, (clf,name) in enumerate(zip(classifiers,names)):
    clf.fit(X_train, y_train); axis=axes[i//3][i%3]
    score.append(accuracy_score(y_test,clf.predict(X_test)))
    position=((clf.predict_proba(X_train))*[-0.05,0.05]).sum(axis=1)
    # print(classification_report(y_test, clf.predict(X_test)))
    wealth=(1+df['returns'][:size]*position).cumprod()
    if(pos is None):
        pos=pd.DataFrame({'returns':df['returns'][:size], name:wealth,'price':
 ↪df['price'][:size],'Time':df.index[:size]}).set_index('Time')
    else:
        pos[name]=position
```

```
    pos[[name,'price']].plot(secondary_y=[name],ax=axis)
```

```
c:\users\denni\envs\thesis\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```