Labeling and MetaLabeling for Supervised Classification

November 2, 2018

Table of Contents

2.5.3 (c) What is accuracy of predictions from primary model if the secondary model does not filter bets? What is classification report?

1 Labeling and MetaLabeling

1.1 Overview

In this chapter of the book AFML, De Prado introduces several novel techniques for labeling returns for the purposes of supervised machine learning.

First he identifies the typical issues of fixed-time horizon labeling methods - primarily that it is easy to mislabel a return due to dynamic nature of volatility throughout a trading period.

More importantly he addresses a major overlooked aspect of the financial literature. He emphasizes that every investment strategy makes use of stop-loss limits of some kind, whether those are enforced by a margin call, risk department or self-imposed. He highlights how unrealistic it is to test/implement/propagate a strategy that profits from positions that would have been stopped out.

> That virtually no publication accounts for that when labeling observations tells you something about the current state of financial literature.
>
> -De Prado, "Advances in Financial Machine Learning", pg.44

He also introduces a technique called metalabeling, which is used to augment a strategy by improving recall while also reducing the likelihood of overfitting.

```
In [1]: %load_ext watermark
        %watermark

        %load_ext autoreload
        %autoreload 2

        # import standard libs
        from IPython.display import display
        from IPython.core.debugger import set_trace as bp
        from pathlib import PurePath, Path
        import sys
        import time
        from collections import OrderedDict as od
        import re
        import os
        import json

        # import python scientific stack
        import pandas as pd
        import pandas_datareader.data as web
        pd.set_option('display.max_rows', 100)
        from dask import dataframe as dd
        from dask.diagnostics import ProgressBar
        from multiprocessing import cpu_count
```

```
pbar = ProgressBar()
pbar.register()
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
from numba import jit
import math
import ffn

# import visual tools
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
import seaborn as sns

plt.style.use('seaborn-talk')
plt.style.use('bmh')
#plt.rcParams['font.family'] = 'DejaVu Sans Mono'
plt.rcParams['font.size'] = 9.5
plt.rcParams['font.weight'] = 'medium'
plt.rcParams['figure.figsize'] = 10,7
blue, green, red, purple, gold, teal = sns.color_palette('colorblind', 6)

# import util libs
from tqdm import tqdm, tqdm_notebook
import warnings
warnings.filterwarnings("ignore")
import missingno as msno
from src.utils.utils import *
import src.features.bars as brs
import src.features.snippets as snp

RANDOM_STATE = 777

print()
%watermark -p pandas,pandas_datareader,dask,numpy,sklearn,statsmodels,scipy,ffn,matplotl
```

```
2018-10-09T19:38:59-06:00

CPython 3.6.6
IPython 6.5.0

compiler   : GCC 7.2.0
system     : Linux
release    : 4.15.0-36-generic
machine    : x86_64
processor  : x86_64
```

```
CPU cores  : 12
interpreter: 64bit

pandas 0.23.4
pandas_datareader 0.6.0+21.gda18fbd
dask 0.19.2
numpy 1.14.6
sklearn 0.20.0
statsmodels 0.9.0
scipy 1.1.0
ffn (0, 3, 3)
matplotlib 3.0.0
seaborn 0.9.0
```

1.2 Code Snippets

Below I reproduce all the relevant code snippets found in the book that are necessary to work through the excercises found at the end of chapter 3.

1.2.1 Symmetric CUSUM Filter [2.5.2.1]

```
In [2]: def getTEvents(gRaw, h):
            tEvents, sPos, sNeg = [], 0, 0
            diff = np.log(gRaw).diff().dropna()
            for i in tqdm(diff.index[1:]):
                try:
                    pos, neg = float(sPos+diff.loc[i]), float(sNeg+diff.loc[i])
                except Exception as e:
                    print(e)
                    print(sPos+diff.loc[i], type(sPos+diff.loc[i]))
                    print(sNeg+diff.loc[i], type(sNeg+diff.loc[i]))
                    break
                sPos, sNeg=max(0., pos), min(0., neg)
                if sNeg<-h:
                    sNeg=0;tEvents.append(i)
                elif sPos>h:
                    sPos=0;tEvents.append(i)
            return pd.DatetimeIndex(tEvents)
```

1.2.2 Daily Volatility Estimator [3.1]

```
In [3]: def getDailyVol(close,span0=100):
            # daily vol reindexed to close
            df0=close.index.searchsorted(close.index-pd.Timedelta(days=1))
            df0=df0[df0>0]
            df0=(pd.Series(close.index[df0-1],
                           index=close.index[close.shape[0]-df0.shape[0]:]))
```

```
    try:
        df0=close.loc[df0.index]/close.loc[df0.values].values-1 # daily rets
    except Exception as e:
        print(f'error: {e}\nplease confirm no duplicate indices')
    df0=df0.ewm(span=span0).std().rename('dailyVol')
    return df0
```

1.2.3 Triple-Barrier Labeling Method [3.2]

```
In [4]: def applyPtSlOnT1(close,events,ptSl,molecule):
            # apply stop loss/profit taking, if it takes place before t1 (end of event)
            events_=events.loc[molecule]
            out=events_[['t1']].copy(deep=True)
            if ptSl[0]>0: pt=ptSl[0]*events_['trgt']
            else: pt=pd.Series(index=events.index) # NaNs
            if ptSl[1]>0: sl=-ptSl[1]*events_['trgt']
            else: sl=pd.Series(index=events.index) # NaNs
            for loc,t1 in events_['t1'].fillna(close.index[-1]).iteritems():
                df0=close[loc:t1] # path prices
                df0=(df0/close[loc]-1)*events_.at[loc,'side'] # path returns
                out.loc[loc,'sl']=df0[df0<sl[loc]].index.min() # earliest stop loss
                out.loc[loc,'pt']=df0[df0>pt[loc]].index.min() # earliest profit taking
            return out
```

1.2.4 Gettting Time of First Touch (getEvents) [3.3], [3.6]

```
In [5]: def getEvents(close, tEvents, ptSl, trgt, minRet, numThreads, t1=False, side=None):
            #1) get target
            trgt=trgt.loc[tEvents]
            trgt=trgt[trgt>minRet] # minRet
            #2) get t1 (max holding period)
            if t1 is False:t1=pd.Series(pd.NaT, index=tEvents)
            #3) form events object, apply stop loss on t1
            if side is None:side_,ptSl_=pd.Series(1.,index=trgt.index), [ptSl[0],ptSl[0]]
            else: side_,ptSl_=side.loc[trgt.index],ptSl[:2]
            events=(pd.concat({'t1':t1,'trgt':trgt,'side':side_}, axis=1)
                    .dropna(subset=['trgt']))
            df0=mpPandasObj(func=applyPtSlOnT1,pdObj=('molecule',events.index),
                            numThreads=numThreads,close=close,events=events,
                            ptSl=ptSl_)
            events['t1']=df0.dropna(how='all').min(axis=1) # pd.min ignores nan
            if side is None:events=events.drop('side',axis=1)
            return events
```

1.2.5 Adding Vertical Barrier [3.4]

```
In [6]: def addVerticalBarrier(tEvents, close, numDays=1):
            t1=close.index.searchsorted(tEvents+pd.Timedelta(days=numDays))
```

```
        t1=t1[t1<close.shape[0]]
        t1=(pd.Series(close.index[t1],index=tEvents[:t1.shape[0]]))
        return t1
```

1.2.6 Labeling for side and size [3.5]

```
In [7]: def getBinsOld(events,close):
            #1) prices aligned with events
            events_=events.dropna(subset=['t1'])
            px=events_.index.union(events_['t1'].values).drop_duplicates()
            px=close.reindex(px,method='bfill')
            #2) create out object
            out=pd.DataFrame(index=events_.index)
            out['ret']=px.loc[events_['t1'].values].values/px.loc[events_.index]-1
            out['bin']=np.sign(out['ret'])
            # where out index and t1 (vertical barrier) intersect label 0
            try:
                locs = out.query('index in @t1').index
                out.loc[locs, 'bin'] = 0
            except:
                pass
            return out
```

1.2.7 Expanding getBins to Incorporate Meta-Labeling [3.7]

```
In [8]: def getBins(events, close):
            '''
            Compute event's outcome (including side information, if provided).
            events is a DataFrame where:
            -events.index is event's starttime
            -events['t1'] is event's endtime
            -events['trgt'] is event's target
            -events['side'] (optional) implies the algo's position side
            Case 1: ('side' not in events): bin in (-1,1) <-label by price action
            Case 2: ('side' in events): bin in (0,1) <-label by pnl (meta-labeling)
            '''
            #1) prices aligned with events
            events_=events.dropna(subset=['t1'])
            px=events_.index.union(events_['t1'].values).drop_duplicates()
            px=close.reindex(px,method='bfill')
            #2) create out object
            out=pd.DataFrame(index=events_.index)
            out['ret']=px.loc[events_['t1'].values].values/px.loc[events_.index]-1
            if 'side' in events_:out['ret']*=events_['side'] # meta-labeling
            out['bin']=np.sign(out['ret'])
            if 'side' in events_:out.loc[out['ret']<=0,'bin']=0 # meta-labeling
            return out
```

1.2.8 Dropping Unnecessary Labels [3.8]

```
In [9]: def dropLabels(events, minPct=.05):
            # apply weights, drop labels with insufficient examples
            while True:
                df0=events['bin'].value_counts(normalize=True)
                if df0.min()>minPct or df0.shape[0]<3:break
                print('dropped label: ', df0.argmin(),df0.min())
                events=events[events['bin']!=df0.argmin()]
            return events
```

1.2.9 Linear Partitions [20.4.1]

```
In [10]: def linParts(numAtoms,numThreads):
             # partition of atoms with a single loop
             parts=np.linspace(0,numAtoms,min(numThreads,numAtoms)+1)
             parts=np.ceil(parts).astype(int)
             return parts
```

```
In [11]: def nestedParts(numAtoms,numThreads,upperTriang=False):
             # partition of atoms with an inner loop
             parts,numThreads_=[0],min(numThreads,numAtoms)
             for num in range(numThreads_):
                 part=1+4*(parts[-1]**2+parts[-1]+numAtoms*(numAtoms+1.)/numThreads_)
                 part=(-1+part**.5)/2.
                 parts.append(part)
             parts=np.round(parts).astype(int)
             if upperTriang: # the first rows are heaviest
                 parts=np.cumsum(np.diff(parts)[::-1])
                 parts=np.append(np.array([0]),parts)
             return parts
```

1.2.10 multiprocessing snippet [20.7]

```
In [12]: def mpPandasObj(func,pdObj,numThreads=24,mpBatches=1,linMols=True,**kargs):
             '''
             Parallelize jobs, return a dataframe or series
             + func: function to be parallelized. Returns a DataFrame
             + pdObj[0]: Name of argument used to pass the molecule
             + pdObj[1]: List of atoms that will be grouped into molecules
             + kwds: any other argument needed by func

             Example: df1=mpPandasObj(func,('molecule',df0.index),24,**kwds)
             '''
             import pandas as pd
             #if linMols:parts=linParts(len(argList[1]),numThreads*mpBatches)
             #else:parts=nestedParts(len(argList[1]),numThreads*mpBatches)
             if linMols:parts=linParts(len(pdObj[1]),numThreads*mpBatches)
             else:parts=nestedParts(len(pdObj[1]),numThreads*mpBatches)
```

```
        jobs=[]
        for i in range(1,len(parts)):
            job={pdObj[0]:pdObj[1][parts[i-1]:parts[i]],'func':func}
            job.update(kargs)
            jobs.append(job)
        if numThreads==1:out=processJobs_(jobs)
        else: out=processJobs(jobs,numThreads=numThreads)
        if isinstance(out[0],pd.DataFrame):df0=pd.DataFrame()
        elif isinstance(out[0],pd.Series):df0=pd.Series()
        else:return out
        for i in out:df0=df0.append(i)
        df0=df0.sort_index()
        return df0
```

1.2.11 single-thread execution for debugging [20.8]

```
In [13]: def processJobs_(jobs):
             # Run jobs sequentially, for debugging
             out=[]
             for job in jobs:
                 out_=expandCall(job)
                 out.append(out_)
             return out
```

1.2.12 Example of async call to multiprocessing lib [20.9]

```
In [14]: import multiprocessing as mp
         import datetime as dt

         #________________________________
         def reportProgress(jobNum,numJobs,time0,task):
             # Report progress as asynch jobs are completed
             msg=[float(jobNum)/numJobs, (time.time()-time0)/60.]
             msg.append(msg[1]*(1/msg[0]-1))
             timeStamp=str(dt.datetime.fromtimestamp(time.time()))
             msg=timeStamp+' '+str(round(msg[0]*100,2))+'% '+task+' done after '+ \
                 str(round(msg[1],2))+' minutes. Remaining '+str(round(msg[2],2))+' minutes.'
             if jobNum<numJobs:sys.stderr.write(msg+'\r')
             else:sys.stderr.write(msg+'\n')
             return
         #________________________________
         def processJobs(jobs,task=None,numThreads=24):
             # Run in parallel.
             # jobs must contain a 'func' callback, for expandCall
             if task is None:task=jobs[0]['func'].__name__
             pool=mp.Pool(processes=numThreads)
             outputs,out,time0=pool.imap_unordered(expandCall,jobs),[],time.time()
```

```
        # Process asyn output, report progress
        for i,out_ in enumerate(outputs,1):
            out.append(out_)
            reportProgress(i,len(jobs),time0,task)
        pool.close();pool.join() # this is needed to prevent memory leaks
        return out
```

1.2.13 Unwrapping the Callback [20.10]

```
In [15]: def expandCall(kargs):
             # Expand the arguments of a callback function, kargs['func']
             func=kargs['func']
             del kargs['func']
             out=func(**kargs)
             return out
```

1.2.14 Pickle Unpickling Objects [20.11]

```
In [16]: def _pickle_method(method):
             func_name=method.im_func.__name__
             obj=method.im_self
             cls=method.im_class
             return _unpickle_method, (func_name,obj,cls)
         #_______________________________
         def _unpickle_method(func_name,obj,cls):
             for cls in cls.mro():
                 try:func=cls.__dict__[func_name]
                 except KeyError:pass
                 else:break
             return func.__get__(obj,cls)
         #_______________________________
         import copyreg,types, multiprocessing as mp
         copyreg.pickle(types.MethodType,_pickle_method,_unpickle_method)
```

2 Exercises

2.1 Import Dataset

Note this dataset below has been resampled to `1s` and then `NaNs` removed. This was done to remove any duplicate indices not accounted for in a simple call to `pd.DataFrame.drop_duplicates()`.

```
In [17]: infp = PurePath(data_dir/'processed'/'IVE_dollarValue_resampled_1s.parquet')
         df = pd.read_parquet(infp)
         cprint(df)
```

```
-------------------------------------------------------------------------------
dataframe information
```

```
--------------------------------------------------------------------------------
                     price     bid     ask      size         v            dv
2018-02-26 15:59:59  115.35  115.34  115.36     412.5     412.5  4.758188e+04
2018-02-26 16:00:00  115.35  115.34  115.35    5362.0    5362.0  6.185067e+05
2018-02-26 16:10:00  115.35  115.22  115.58       0.0       0.0  0.000000e+00
2018-02-26 16:16:14  115.30  114.72  115.62  778677.0  778677.0  8.978146e+07
2018-02-26 18:30:00  115.35  114.72  117.38       0.0       0.0  0.000000e+00
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 941297 entries, 2009-09-28 09:30:00 to 2018-02-26 18:30:00
Data columns (total 6 columns):
price    941297 non-null float64
bid      941297 non-null float64
ask      941297 non-null float64
size     941297 non-null float64
v        941297 non-null float64
dv       941297 non-null float64
dtypes: float64(6)
memory usage: 50.3 MB
None
--------------------------------------------------------------------------------
```

2.2 [3.1] Form Dollar Bars

```
In [18]: dbars = dollar_bar_df(df, 'dv', 1_000_000).drop_duplicates().dropna()
         cprint(dbars)
```

```
100%|| 941297/941297 [00:00<00:00, 2919179.94it/s]

--------------------------------------------------------------------------------
dataframe information
--------------------------------------------------------------------------------
                     price         bid         ask           size  \
2018-02-26 15:31:06  115.29  115.280000  115.290000    2022.000000
2018-02-26 15:40:15  115.41  115.400000  115.410000     723.000000
2018-02-26 15:49:42  115.20  115.176667  115.186667    4487.166667
2018-02-26 15:59:04  115.27  115.260000  115.270000     300.000000
2018-02-26 16:16:14  115.30  114.720000  115.620000  778677.000000

                                 v            dv
2018-02-26 15:31:06    2022.000000  2.331164e+05
2018-02-26 15:40:15     723.000000  8.344143e+04
2018-02-26 15:49:42    4487.166667  5.171190e+05
2018-02-26 15:59:04     300.000000  3.458100e+04
2018-02-26 16:16:14  778677.000000  8.978146e+07
--------------------------------------------------
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 30860 entries, 2009-09-28 09:53:49 to 2018-02-26 16:16:14
Data columns (total 6 columns):
price    30860 non-null float64
bid      30860 non-null float64
ask      30860 non-null float64
size     30860 non-null float64
v        30860 non-null float64
dv       30860 non-null float64
dtypes: float64(6)
memory usage: 1.6 MB
None
-------------------------------------------------------------------------------
```

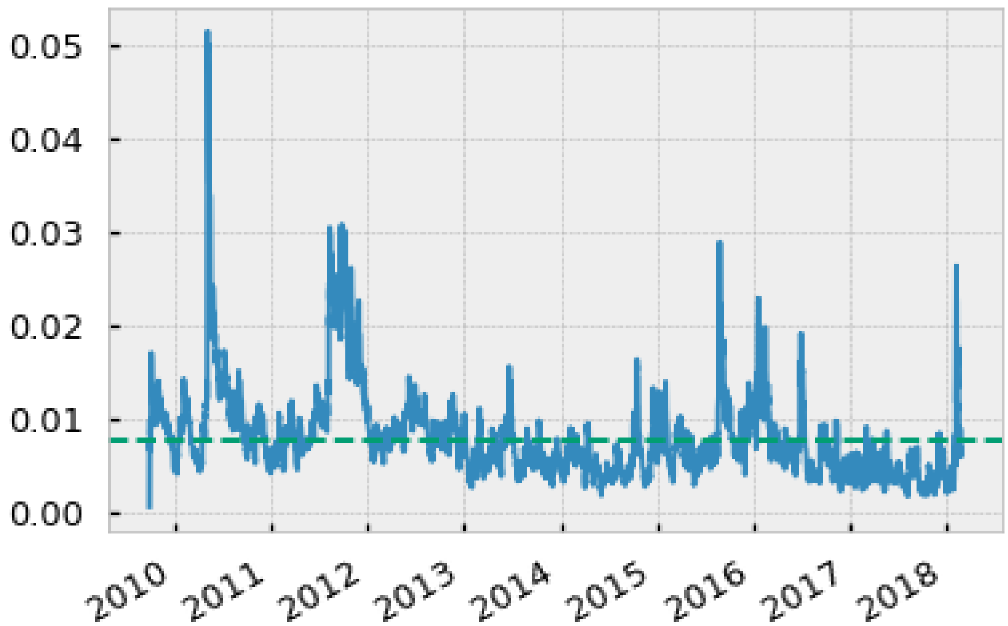
2.2.1 (a) Run cusum filter with threshold equal to std dev of daily returns

```
In [19]: close = dbars.price.copy()
         dailyVol = getDailyVol(close)
         cprint(dailyVol.to_frame())
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                     dailyVol
2018-02-26 15:31:06  0.006852
2018-02-26 15:40:15  0.006893
2018-02-26 15:49:42  0.006889
2018-02-26 15:59:04  0.006894
2018-02-26 16:16:14  0.006902
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 30843 entries, 2009-09-29 10:03:18 to 2018-02-26 16:16:14
Data columns (total 1 columns):
dailyVol    30842 non-null float64
dtypes: float64(1)
memory usage: 481.9 KB
None
-------------------------------------------------------------------------------
```

```
In [20]: f,ax=plt.subplots()
         dailyVol.plot(ax=ax)
         ax.axhline(dailyVol.mean(),ls='--',color=red)
```

Out[20]: <matplotlib.lines.Line2D at 0x7f65295ccd30>



In [21]: tEvents = getTEvents(close,h=dailyVol.mean())
         tEvents

```
100%|| 30858/30858 [00:01<00:00, 15448.31it/s]
```

Out[21]:
```
DatetimeIndex(['2009-09-29 09:33:01', '2009-09-30 09:45:21',
               '2009-09-30 13:31:12', '2009-10-01 09:43:58',
               '2009-10-01 11:12:07', '2009-10-02 09:44:14',
               '2009-10-02 10:35:05', '2009-10-05 09:51:42',
               '2009-10-05 14:55:48', '2009-10-06 09:29:52',
               ...
               '2018-02-16 14:23:51', '2018-02-20 09:30:00',
               '2018-02-20 15:21:07', '2018-02-21 14:04:12',
               '2018-02-21 15:12:30', '2018-02-22 12:18:21',
               '2018-02-22 14:56:14', '2018-02-23 11:37:32',
               '2018-02-23 15:58:39', '2018-02-26 13:06:34'],
              dtype='datetime64[ns]', length=2278, freq=None)
```

2.2.2 (b) Add vertical barrier

In [22]:
```
t1 = addVerticalBarrier(tEvents, close)
t1
```

```
Out[22]: 2009-09-29 09:33:01   2009-09-30 09:45:21
         2009-09-30 09:45:21   2009-10-01 10:00:48
         2009-09-30 13:31:12   2009-10-01 13:33:25
         2009-10-01 09:43:58   2009-10-02 09:44:14
         2009-10-01 11:12:07   2009-10-02 11:50:21
         2009-10-02 09:44:14   2009-10-05 09:51:42
         2009-10-02 10:35:05   2009-10-05 09:51:42
         2009-10-05 09:51:42   2009-10-06 10:16:02
         2009-10-05 14:55:48   2009-10-06 15:35:49
         2009-10-06 09:29:52   2009-10-07 09:47:16
         2009-10-06 11:32:02   2009-10-07 11:48:22
         2009-10-06 14:07:37   2009-10-07 14:22:36
         2009-10-08 09:29:51   2009-10-09 09:31:12
         2009-10-12 09:31:02   2009-10-13 09:47:54
         2009-10-13 10:52:10   2009-10-14 11:12:03
         2009-10-14 09:29:52   2009-10-15 09:37:24
         2009-10-14 15:30:48   2009-10-15 15:57:25
         2009-10-16 09:55:03   2009-10-19 09:37:41
         2009-10-16 15:40:15   2009-10-19 09:37:41
         2009-10-19 11:39:38   2009-10-20 11:50:28
         2009-10-20 11:50:28   2009-10-21 12:44:38
         2009-10-21 10:11:57   2009-10-22 10:47:06
         2009-10-21 15:32:09   2009-10-22 15:49:30
         2009-10-22 09:55:51   2009-10-23 10:03:53
         2009-10-22 14:33:52   2009-10-23 14:49:39
         2009-10-23 10:57:52   2009-10-26 09:52:17
         2009-10-26 09:52:17   2009-10-27 09:57:46
         2009-10-26 11:32:02   2009-10-27 12:04:42
         2009-10-26 11:59:14   2009-10-27 12:04:42
         2009-10-27 13:37:35   2009-10-28 14:04:15
         2009-10-28 10:00:16   2009-10-29 10:00:59
         2009-10-28 14:41:52   2009-10-29 15:00:53
         2009-10-29 09:32:01   2009-10-30 09:43:02
         2009-10-29 13:40:22   2009-10-30 13:54:51
         2009-10-30 09:58:07   2009-11-02 09:51:15
         2009-10-30 11:51:20   2009-11-02 09:51:15
         2009-10-30 12:57:50   2009-11-02 09:51:15
         2009-10-30 15:06:13   2009-11-02 09:51:15
         2009-10-30 15:44:12   2009-11-02 09:51:15
         2009-11-02 10:17:36   2009-11-03 10:42:33
         2009-11-02 12:23:50   2009-11-03 12:24:26
         2009-11-02 12:58:06   2009-11-03 13:10:26
         2009-11-02 14:07:16   2009-11-03 14:22:31
         2009-11-02 14:55:04   2009-11-03 15:18:16
         2009-11-03 14:22:31   2009-11-04 14:41:42
         2009-11-04 09:34:15   2009-11-05 09:59:36
         2009-11-04 15:46:56   2009-11-05 16:09:46
         2009-11-05 09:59:36   2009-11-06 10:06:33
```

```
2009-11-05 16:09:46   2009-11-09 09:54:17
2009-11-09 09:54:17   2009-11-10 10:09:52
                                ...
2018-02-06 09:36:34   2018-02-07 09:43:03
2018-02-06 09:58:38   2018-02-07 10:04:28
2018-02-06 10:18:08   2018-02-07 10:22:20
2018-02-06 10:38:41   2018-02-07 10:39:35
2018-02-06 11:35:33   2018-02-07 11:46:44
2018-02-06 11:53:57   2018-02-07 11:57:50
2018-02-06 12:32:24   2018-02-07 12:42:28
2018-02-06 13:04:03   2018-02-07 13:08:44
2018-02-06 14:19:57   2018-02-07 14:20:37
2018-02-06 14:49:56   2018-02-07 14:53:22
2018-02-06 15:05:41   2018-02-07 15:11:44
2018-02-06 15:42:53   2018-02-07 15:47:02
2018-02-07 09:43:03   2018-02-08 09:57:38
2018-02-07 11:15:27   2018-02-08 11:18:31
2018-02-07 13:16:25   2018-02-08 13:17:26
2018-02-07 15:28:09   2018-02-08 15:33:11
2018-02-07 15:58:58   2018-02-08 15:59:48
2018-02-08 10:33:27   2018-02-09 10:41:40
2018-02-08 12:29:28   2018-02-09 12:40:46
2018-02-08 13:45:14   2018-02-09 13:52:34
2018-02-08 15:07:57   2018-02-09 15:09:17
2018-02-08 15:45:50   2018-02-09 15:47:50
2018-02-09 09:30:00   2018-02-12 09:30:00
2018-02-09 10:41:40   2018-02-12 09:30:00
2018-02-09 12:05:08   2018-02-12 09:30:00
2018-02-09 13:27:21   2018-02-12 09:30:00
2018-02-09 13:52:34   2018-02-12 09:30:00
2018-02-09 14:11:06   2018-02-12 09:30:00
2018-02-09 15:05:41   2018-02-12 09:30:00
2018-02-09 15:29:15   2018-02-12 09:30:00
2018-02-09 15:47:50   2018-02-12 09:30:00
2018-02-12 09:30:00   2018-02-13 09:30:00
2018-02-12 10:25:02   2018-02-13 10:36:48
2018-02-12 12:12:51   2018-02-13 12:34:24
2018-02-13 09:30:00   2018-02-14 09:30:00
2018-02-13 13:43:37   2018-02-14 13:53:59
2018-02-14 10:30:48   2018-02-15 10:42:27
2018-02-14 13:36:02   2018-02-15 13:42:09
2018-02-15 09:31:56   2018-02-16 09:42:36
2018-02-15 14:05:41   2018-02-16 14:15:08
2018-02-16 11:11:50   2018-02-20 09:30:00
2018-02-16 14:23:51   2018-02-20 09:30:00
2018-02-20 09:30:00   2018-02-21 09:34:28
2018-02-20 15:21:07   2018-02-21 15:22:14
2018-02-21 14:04:12   2018-02-22 14:20:25
```

```
2018-02-21 15:12:30   2018-02-22 15:16:50
2018-02-22 12:18:21   2018-02-23 12:30:16
2018-02-22 14:56:14   2018-02-23 15:02:21
2018-02-23 11:37:32   2018-02-26 09:30:00
2018-02-23 15:58:39   2018-02-26 09:30:00
Length: 2277, dtype: datetime64[ns]
```

2.2.3 (c) Apply triple-barrier method where `ptSl = [1,1]` and `t1` is the series created in `1.b`

```
In [23]: # create target series
         ptsl = [1,1]
         target=dailyVol
         # select minRet
         minRet = 0.01

         # Run in single-threaded mode on Windows
         import platform
         if platform.system() == "Windows":
             cpus = 1
         else:
             cpus = cpu_count() - 1

         events = getEvents(close,tEvents,ptsl,target,minRet,cpus,t1=t1)
```

```
2018-10-09 19:40:14.051086 100.0% applyPtSlOnT1 done after 0.0 minutes. Remaining 0.0 minutes.
```

```
In [24]: cprint(events)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                                     t1      trgt
2018-02-13 13:43:37 2018-02-14 13:53:59  0.014365
2018-02-14 10:30:48 2018-02-15 09:31:56  0.012136
2018-02-14 13:36:02 2018-02-15 13:42:09  0.011688
2018-02-15 09:31:56 2018-02-16 09:42:36  0.011244
2018-02-15 14:05:41 2018-02-16 12:05:18  0.010183
---------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 929 entries, 2009-10-05 14:55:48 to 2018-02-15 14:05:41
Data columns (total 2 columns):
t1      929 non-null datetime64[ns]
trgt    929 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 21.8 KB
None
-------------------------------------------------------------------------------
```

2.2.4 (d) Apply `getBins` to generate labels

```
In [25]: labels = getBins(events, close)
         cprint(labels)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                          ret  bin
2018-02-13 13:43:37  0.010108  1.0
2018-02-14 10:30:48  0.015045  1.0
2018-02-14 13:36:02  0.005056  1.0
2018-02-15 09:31:56  0.003964  1.0
2018-02-15 14:05:41  0.010431  1.0
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 929 entries, 2009-10-05 14:55:48 to 2018-02-15 14:05:41
Data columns (total 2 columns):
ret    929 non-null float64
bin    929 non-null float64
dtypes: float64(2)
memory usage: 61.8 KB
None
-------------------------------------------------------------------------------
```

```
In [26]: labels.bin.value_counts()
```

```
Out[26]:  1.0    523
         -1.0    406
         Name: bin, dtype: int64
```

2.3 [3.2] Use snippet 3.8 to drop under-populated labels

```
In [27]: clean_labels = dropLabels(labels)
         cprint(clean_labels)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                          ret  bin
2018-02-13 13:43:37  0.010108  1.0
2018-02-14 10:30:48  0.015045  1.0
2018-02-14 13:36:02  0.005056  1.0
2018-02-15 09:31:56  0.003964  1.0
```

```
2018-02-15 14:05:41  0.010431  1.0
------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 929 entries, 2009-10-05 14:55:48 to 2018-02-15 14:05:41
Data columns (total 2 columns):
ret    929 non-null float64
bin    929 non-null float64
dtypes: float64(2)
memory usage: 61.8 KB
None
-------------------------------------------------------------------------------
```

In [28]:
```
clean_labels.bin.value_counts()
```

Out[28]:
```
 1.0    523
-1.0    406
Name: bin, dtype: int64
```

2.4 [3.4] Develop moving average crossover strategy. For each obs. the model suggests a side but not size of the bet

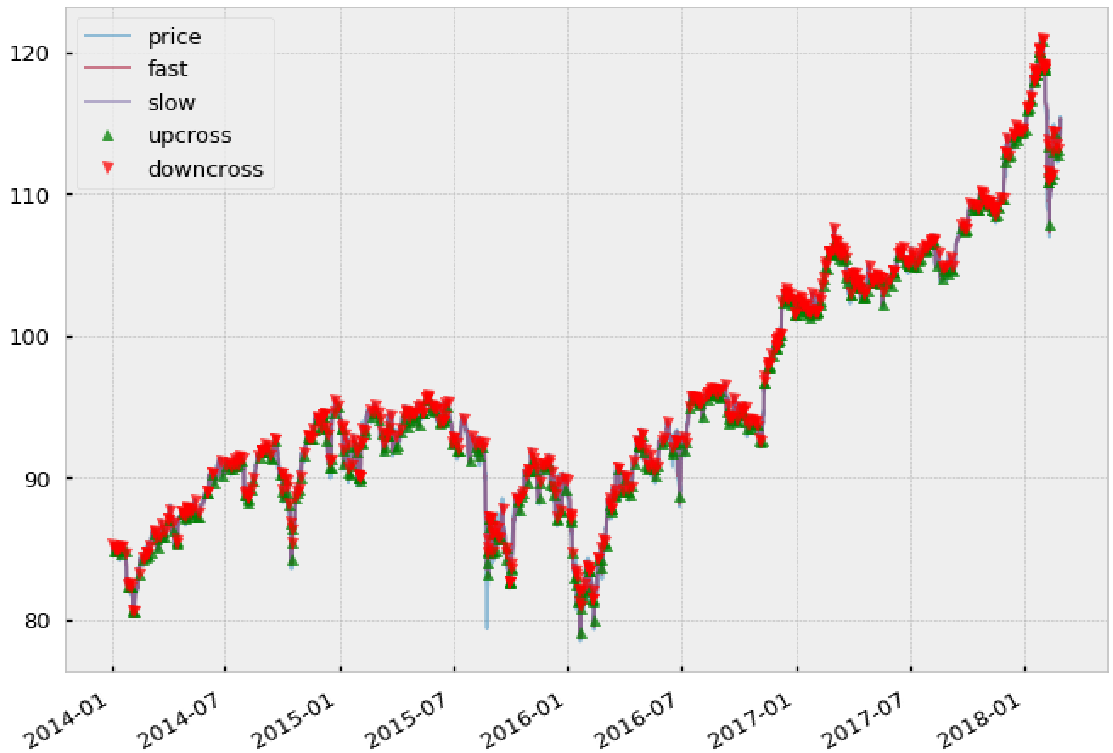
In [29]:
```
fast_window = 3
slow_window = 7

close_df = (pd.DataFrame()
            .assign(price=close)
            .assign(fast=close.ewm(fast_window).mean())
            .assign(slow=close.ewm(slow_window).mean()))
cprint(close_df)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                      price        fast        slow
2018-02-26 15:31:06  115.29  115.227691  115.057569
2018-02-26 15:40:15  115.41  115.273268  115.101623
2018-02-26 15:49:42  115.20  115.254951  115.113920
2018-02-26 15:59:04  115.27  115.258713  115.133430
2018-02-26 16:16:14  115.30  115.269035  115.154251
------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 30860 entries, 2009-09-28 09:53:49 to 2018-02-26 16:16:14
Data columns (total 3 columns):
price    30860 non-null float64
fast     30860 non-null float64
slow     30860 non-null float64
dtypes: float64(3)
```

```
memory usage: 964.4 KB
None
--------------------------------------------------------------------------------
```

In [30]:
```
def get_up_cross(df):
    crit1 = df.fast.shift(1) < df.slow.shift(1)
    crit2 = df.fast > df.slow
    return df.fast[(crit1) & (crit2)]

def get_down_cross(df):
    crit1 = df.fast.shift(1) > df.slow.shift(1)
    crit2 = df.fast < df.slow
    return df.fast[(crit1) & (crit2)]

up = get_up_cross(close_df)
down = get_down_cross(close_df)

f, ax = plt.subplots(figsize=(11,8))

close_df.loc['2014':].plot(ax=ax, alpha=.5)
up.loc['2014':].plot(ax=ax,ls='',marker='^', markersize=7,
                     alpha=0.75, label='upcross', color='g')
down.loc['2014':].plot(ax=ax,ls='',marker='v', markersize=7,
                       alpha=0.75, label='downcross', color='r')

ax.legend()
```

Out[30]:
```
<matplotlib.legend.Legend at 0x7f652c2c8978>
```



2.4.1 (a) Derive meta-labels for `ptSl = [1,2]` and `t1` where `numdays=1`. Use as `trgt` dailyVol computed by snippet 3.1 (get events with sides)

```
In [31]: side_up = pd.Series(1, index=up.index)
         side_down = pd.Series(-1, index=down.index)
         side = pd.concat([side_up,side_down]).sort_index()
         cprint(side)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                     0
2018-02-21 11:10:00  1
2018-02-21 15:12:30 -1
2018-02-22 11:48:39  1
2018-02-22 13:34:29 -1
2018-02-23 10:01:41  1
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1712 entries, 2009-09-30 09:45:21 to 2018-02-23 10:01:41
Data columns (total 1 columns):
0    1712 non-null int64
dtypes: int64(1)
```

```
memory usage: 26.8 KB
None
--------------------------------------------------------------------------------
```

```
In [32]: minRet = .01
         ptsl=[1,2]
         ma_events = getEvents(close,tEvents,ptsl,target,minRet,cpus,t1=t1,side=side)
         cprint(ma_events)
```

```
--------------------------------------------------------------------------------
dataframe information
--------------------------------------------------------------------------------
                     side                  t1      trgt
2018-02-13 13:43:37   NaN 2018-02-14 13:53:59  0.014365
2018-02-14 10:30:48   NaN 2018-02-15 10:42:27  0.012136
2018-02-14 13:36:02   NaN 2018-02-15 13:42:09  0.011688
2018-02-15 09:31:56   NaN 2018-02-16 09:42:36  0.011244
2018-02-15 14:05:41   NaN 2018-02-16 14:15:08  0.010183
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 929 entries, 2009-10-05 14:55:48 to 2018-02-15 14:05:41
Data columns (total 3 columns):
side    102 non-null float64
t1      929 non-null datetime64[ns]
trgt    929 non-null float64
dtypes: datetime64[ns](1), float64(2)
memory usage: 29.0 KB
None
--------------------------------------------------------------------------------
```

```
2018-10-09 19:40:17.323883 100.0% applyPtSlOnT1 done after 0.0 minutes. Remaining 0.0 minutes.
```

```
In [33]: ma_events.side.value_counts()

Out[33]:  1.0    53
         -1.0    49
         Name: side, dtype: int64

In [34]: ma_side = ma_events.dropna().side

In [35]: ma_bins = getBins(ma_events,close).dropna()
         cprint(ma_bins)
```

```
--------------------------------------------------------------------------------
dataframe information
```

```
--------------------------------------------------------------------------------
                          ret  bin
2016-07-07 14:28:00 -0.018703  0.0
2016-07-08 09:30:57  0.010571  1.0
2018-02-06 10:18:08 -0.026702  0.0
2018-02-07 15:28:09 -0.030792  0.0
2018-02-13 09:30:00 -0.001803  0.0
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 102 entries, 2009-10-29 13:40:22 to 2018-02-13 09:30:00
Data columns (total 2 columns):
ret    102 non-null float64
bin    102 non-null float64
dtypes: float64(2)
memory usage: 2.4 KB
None
--------------------------------------------------------------------------------
```

```
In [36]: Xx = pd.merge_asof(ma_bins, side.to_frame().rename(columns={0:'side'}),
                            left_index=True, right_index=True, direction='forward')
         cprint(Xx)
```

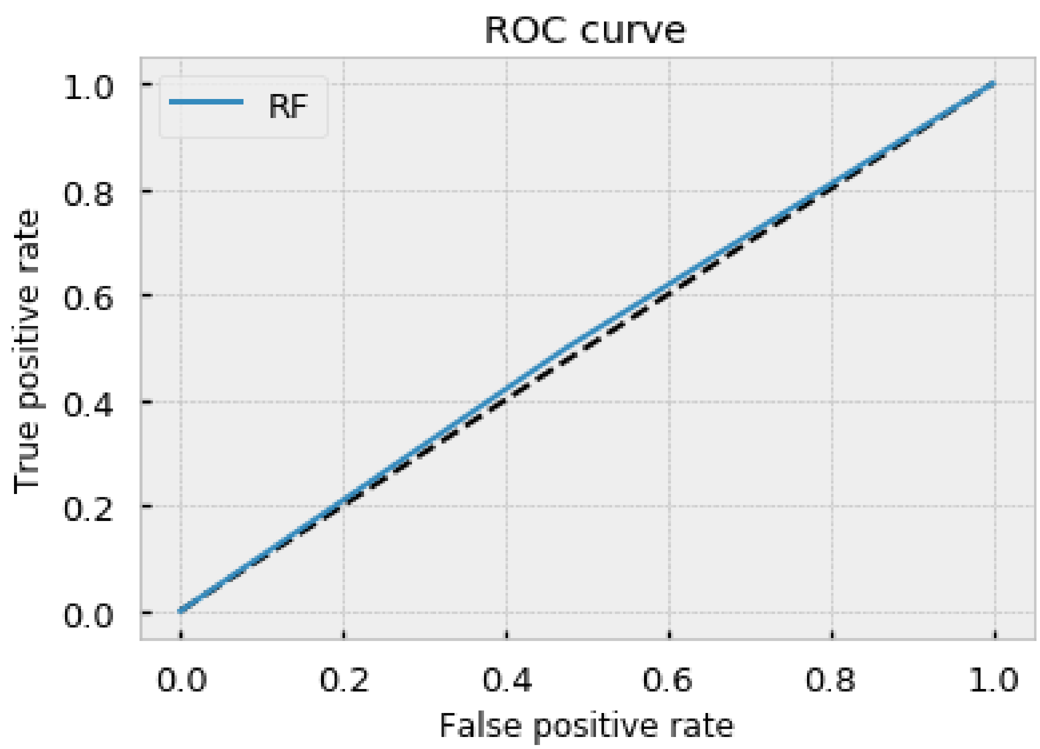
```
--------------------------------------------------------------------------------
dataframe information
--------------------------------------------------------------------------------
                          ret  bin  side
2016-07-07 14:28:00 -0.018703  0.0    -1
2016-07-08 09:30:57  0.010571  1.0     1
2018-02-06 10:18:08 -0.026702  0.0    -1
2018-02-07 15:28:09 -0.030792  0.0     1
2018-02-13 09:30:00 -0.001803  0.0    -1
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 102 entries, 2009-10-29 13:40:22 to 2018-02-13 09:30:00
Data columns (total 3 columns):
ret     102 non-null float64
bin     102 non-null float64
side    102 non-null int64
dtypes: float64(2), int64(1)
memory usage: 3.2 KB
None
--------------------------------------------------------------------------------
```

2.4.2 (b) Train Random Forest to decide whether to trade or not {0,1} since underlying model (crossing m.a.) has decided the side, {-1,1}

In [37]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, classification_report
```

In [38]:
```
X = ma_side.values.reshape(-1,1)
#X = Xx.side.values.reshape(-1,1)
y = ma_bins.bin.values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

n_estimator = 10000
rf = RandomForestClassifier(max_depth=2, n_estimators=n_estimator,
                            criterion='entropy', random_state=RANDOM_STATE)
rf.fit(X_train, y_train)

# The random forest model by itself
y_pred_rf = rf.predict_proba(X_test)[:, 1]
y_pred = rf.predict(X_test)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)
print(classification_report(y_test, y_pred))

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00        21
         1.0       0.59      1.00      0.74        30

   micro avg       0.59      0.59      0.59        51
   macro avg       0.29      0.50      0.37        51
weighted avg       0.35      0.59      0.44        51
```



2.5 [3.5] Develop mean-reverting Bollinger Band Strategy. For each obs. model suggests a side but not size of the bet.

```
In [39]: def bbands(price, window=None, width=None, numsd=None):
             """ returns average, upper band, and lower band"""
             ave = price.rolling(window).mean()
             sd = price.rolling(window).std(ddof=0)
             if width:
                 upband = ave * (1+width)
                 dnband = ave * (1-width)
                 return price, np.round(ave,3), np.round(upband,3), np.round(dnband,3)
             if numsd:
                 upband = ave + (sd*numsd)
                 dnband = ave - (sd*numsd)
                 return price, np.round(ave,3), np.round(upband,3), np.round(dnband,3)
```

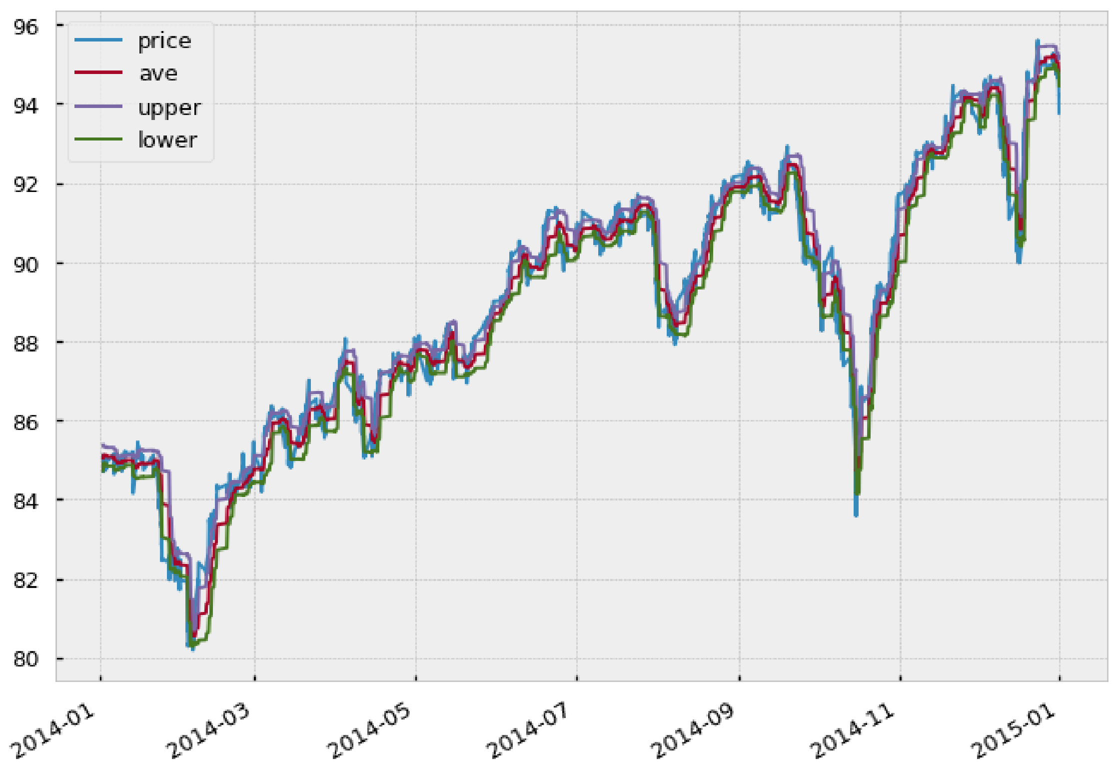
```
In [40]: window=50
         bb_df = pd.DataFrame()
         bb_df['price'],bb_df['ave'],bb_df['upper'],bb_df['lower']=bbands(close, window=window,
         bb_df.dropna(inplace=True)
         cprint(bb_df)
```

```
--------------------------------------------------------------------------------
dataframe information
```

```
--------------------------------------------------------------------------------
                      price      ave    upper    lower
2018-02-26 15:31:06  115.29  114.005  114.959  113.051
2018-02-26 15:40:15  115.41  114.069  115.008  113.129
2018-02-26 15:49:42  115.20  114.124  115.047  113.202
2018-02-26 15:59:04  115.27  114.183  115.083  113.282
2018-02-26 16:16:14  115.30  114.231  115.125  113.338
---------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 30811 entries, 2009-10-01 15:51:02 to 2018-02-26 16:16:14
Data columns (total 4 columns):
price    30811 non-null float64
ave      30811 non-null float64
upper    30811 non-null float64
lower    30811 non-null float64
dtypes: float64(4)
memory usage: 1.2 MB
None
--------------------------------------------------------------------------------
```

In [41]:
```python
f,ax=plt.subplots(figsize=(11,8))
bb_df.loc['2014'].plot(ax=ax)
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f652955fbe0>



```
In [42]: def get_up_cross(df, col):
             # col is price column
             crit1 = df[col].shift(1) < df.upper.shift(1)
             crit2 = df[col] > df.upper
             return df[col][(crit1) & (crit2)]

         def get_down_cross(df, col):
             # col is price column
             crit1 = df[col].shift(1) > df.lower.shift(1)
             crit2 = df[col] < df.lower
             return df[col][(crit1) & (crit2)]

         bb_down = get_down_cross(bb_df, 'price')
         bb_up = get_up_cross(bb_df, 'price')

         f, ax = plt.subplots(figsize=(11,8))

         bb_df.loc['2014':].plot(ax=ax, alpha=.5)
         bb_up.loc['2014':].plot(ax=ax, ls='', marker='^', markersize=7,
                                 alpha=0.75, label='upcross', color='g')
         bb_down.loc['2014':].plot(ax=ax, ls='', marker='v', markersize=7,
                                   alpha=0.75, label='downcross', color='r')
         ax.legend()

Out[42]: <matplotlib.legend.Legend at 0x7f6527e28f28>
```

2.5.1 (a) Derive meta-labels for `ptSl=[0,2]` and `t1` where `numdays=1`. Use as `trgt` dailyVol.

```
In [43]: bb_side_up = pd.Series(-1, index=bb_up.index) # sell on up cross for mean reversion
         bb_side_down = pd.Series(1, index=bb_down.index) # buy on down cross for mean reversion
         bb_side_raw = pd.concat([bb_side_up,bb_side_down]).sort_index()
         cprint(bb_side_raw)

         minRet = .01
         ptsl=[0,2]
         bb_events = getEvents(close,tEvents,ptsl,target,minRet,cpus,t1=t1,side=bb_side_raw)
         cprint(bb_events)

         bb_side = bb_events.dropna().side
         cprint(bb_side)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                     0
2018-02-22 13:34:29  1
2018-02-22 14:20:25  1
2018-02-22 14:44:33  1
2018-02-23 13:41:26 -1
```

```
2018-02-23 14:40:49 -1
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2040 entries, 2009-10-06 09:29:52 to 2018-02-23 14:40:49
Data columns (total 1 columns):
0    2040 non-null int64
dtypes: int64(1)
memory usage: 31.9 KB
None
--------------------------------------------------------------------------------


2018-10-09 19:40:27.517390 100.0% applyPtSlOnT1 done after 0.0 minutes. Remaining 0.0 minutes.


--------------------------------------------------------------------------------
dataframe information
--------------------------------------------------------------------------------
                     side                  t1      trgt
2018-02-13 13:43:37  -1.0 2018-02-14 13:53:59  0.014365
2018-02-14 10:30:48   NaN 2018-02-15 10:42:27  0.012136
2018-02-14 13:36:02   NaN 2018-02-15 13:42:09  0.011688
2018-02-15 09:31:56   NaN 2018-02-16 09:42:36  0.011244
2018-02-15 14:05:41   NaN 2018-02-16 14:15:08  0.010183
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 929 entries, 2009-10-05 14:55:48 to 2018-02-15 14:05:41
Data columns (total 3 columns):
side    139 non-null float64
t1      929 non-null datetime64[ns]
trgt    929 non-null float64
dtypes: datetime64[ns](1), float64(2)
memory usage: 29.0 KB
None
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
dataframe information
--------------------------------------------------------------------------------
                     side
2016-07-07 10:17:10  -1.0
2016-07-08 09:30:57  -1.0
2018-02-06 10:18:08   1.0
2018-02-06 14:19:57   1.0
2018-02-13 13:43:37  -1.0
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
```

```
DatetimeIndex: 139 entries, 2009-10-06 09:29:52 to 2018-02-13 13:43:37
Data columns (total 1 columns):
side    139 non-null float64
dtypes: float64(1)
memory usage: 2.2 KB
None
-------------------------------------------------------------------------------
```

```
In [44]: bb_side.value_counts()

Out[44]:  1.0    72
         -1.0    67
         Name: side, dtype: int64

In [45]: bb_bins = getBins(bb_events,close).dropna()
         cprint(bb_bins)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                          ret  bin
2016-07-07 10:17:10 -0.003791  0.0
2016-07-08 09:30:57 -0.010571  0.0
2018-02-06 10:18:08  0.025085  1.0
2018-02-06 14:19:57  0.028779  1.0
2018-02-13 13:43:37 -0.010108  0.0
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 139 entries, 2009-10-06 09:29:52 to 2018-02-13 13:43:37
Data columns (total 2 columns):
ret    139 non-null float64
bin    139 non-null float64
dtypes: float64(2)
memory usage: 3.3 KB
None
-------------------------------------------------------------------------------
```

```
In [46]: bb_bins.bin.value_counts()

Out[46]: 0.0    79
         1.0    60
         Name: bin, dtype: int64
```

2.5.2 (b) train random forest to decide to trade or not. Use features: volatility, serial correlation, and the crossing moving averages from exercise 2.

```
In [47]: def returns(s):
             arr = np.diff(np.log(s))
             return (pd.Series(arr, index=s.index[1:]))

         def df_rolling_autocorr(df, window, lag=1):
             """Compute rolling column-wise autocorrelation for a DataFrame."""

             return (df.rolling(window=window)
                     .corr(df.shift(lag))) # could .dropna() here

         #df_rolling_autocorr(d1, window=21).dropna().head()
```

```
In [48]: srl_corr = df_rolling_autocorr(returns(close), window=window).rename('srl_corr')
         cprint(srl_corr)
```

```
--------------------------------------------------------------------------------
dataframe information
--------------------------------------------------------------------------------
                     srl_corr
2018-02-26 15:31:06  0.028037
2018-02-26 15:40:15  0.015957
2018-02-26 15:49:42  0.032877
2018-02-26 15:59:04  0.046014
2018-02-26 16:16:14  0.109129
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 30859 entries, 2009-09-28 10:06:04 to 2018-02-26 16:16:14
Data columns (total 1 columns):
srl_corr    30809 non-null float64
dtypes: float64(1)
memory usage: 482.2 KB
None
--------------------------------------------------------------------------------
```

```
In [49]: features = (pd.DataFrame()
                     .assign(vol=bb_events.trgt)
                     .assign(ma_side=ma_side)
                     .assign(srl_corr=srl_corr)
                     .drop_duplicates()
                     .dropna())
         cprint(features)
```

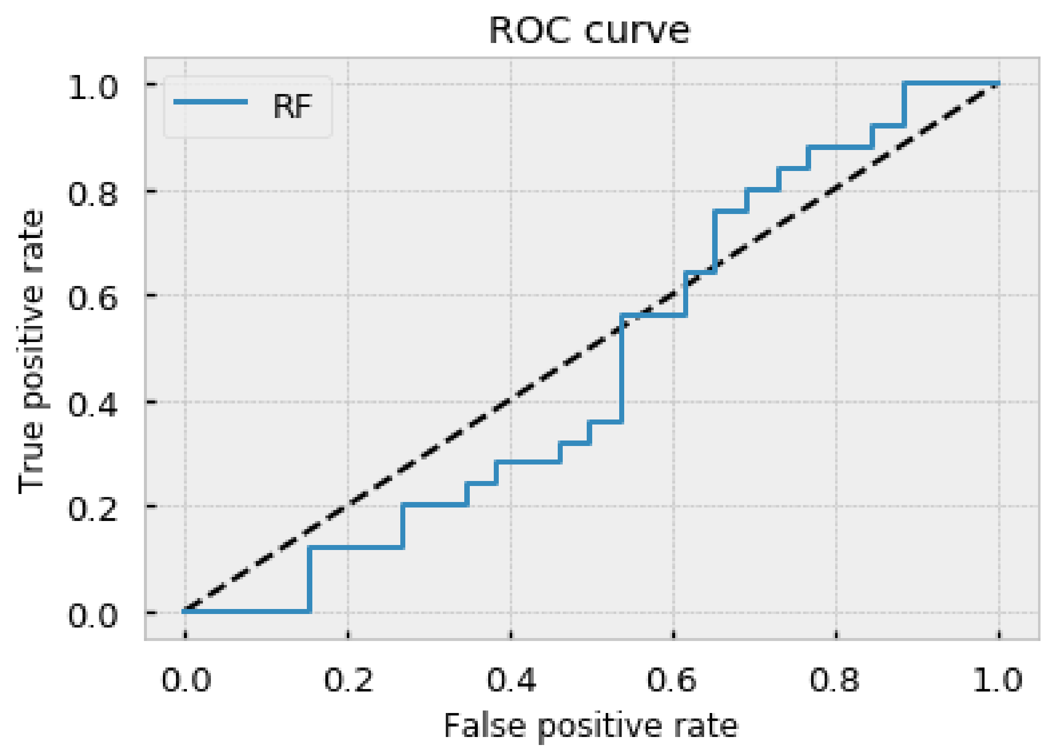
```
--------------------------------------------------------------------------------
dataframe information
```

```
-------------------------------------------------------------------------------
                          vol  ma_side  srl_corr
2016-07-07 14:28:00  0.012624     -1.0  0.251865
2016-07-08 09:30:57  0.011944      1.0  0.238590
2018-02-06 10:18:08  0.013317     -1.0  0.123961
2018-02-07 15:28:09  0.024870      1.0 -0.005597
2018-02-13 09:30:00  0.017363     -1.0  0.198935
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 102 entries, 2009-10-29 13:40:22 to 2018-02-13 09:30:00
Data columns (total 3 columns):
vol         102 non-null float64
ma_side     102 non-null float64
srl_corr    102 non-null float64
dtypes: float64(3)
memory usage: 3.2 KB
None
-------------------------------------------------------------------------------
```

```
In [50]: Xy = (pd.merge_asof(features, bb_bins[['bin']],
                             left_index=True, right_index=True,
                             direction='forward').dropna())
         cprint(Xy)
```

```
-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                          vol  ma_side  srl_corr  bin
2016-07-07 14:28:00  0.012624     -1.0  0.251865  0.0
2016-07-08 09:30:57  0.011944      1.0  0.238590  0.0
2018-02-06 10:18:08  0.013317     -1.0  0.123961  1.0
2018-02-07 15:28:09  0.024870      1.0 -0.005597  0.0
2018-02-13 09:30:00  0.017363     -1.0  0.198935  0.0
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 102 entries, 2009-10-29 13:40:22 to 2018-02-13 09:30:00
Data columns (total 4 columns):
vol         102 non-null float64
ma_side     102 non-null float64
srl_corr    102 non-null float64
bin         102 non-null float64
dtypes: float64(4)
memory usage: 4.0 KB
None
-------------------------------------------------------------------------------
```

```
In [51]: Xy.bin.value_counts()

Out[51]: 0.0    60
         1.0    42
         Name: bin, dtype: int64

In [52]: X = Xy.drop('bin',axis=1).values
         y = Xy['bin'].values

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, shuffle=False)

         n_estimator = 10000
         rf = RandomForestClassifier(max_depth=2, n_estimators=n_estimator,
                                     criterion='entropy', random_state=RANDOM_STATE)
         rf.fit(X_train, y_train)

         # The random forest model by itself
         y_pred_rf = rf.predict_proba(X_test)[:, 1]
         y_pred = rf.predict(X_test)
         fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)
         print(classification_report(y_test, y_pred, target_names=['no_trade','trade']))

         plt.figure(1)
         plt.plot([0, 1], [0, 1], 'k--')
         plt.plot(fpr_rf, tpr_rf, label='RF')
         plt.xlabel('False positive rate')
         plt.ylabel('True positive rate')
         plt.title('ROC curve')
         plt.legend(loc='best')
         plt.show()

              precision    recall  f1-score   support

    no_trade       0.47      0.73      0.58        26
       trade       0.36      0.16      0.22        25

   micro avg       0.45      0.45      0.45        51
   macro avg       0.42      0.45      0.40        51
weighted avg       0.42      0.45      0.40        51
```



2.5.3 (c) What is accuracy of predictions from primary model if the secondary model does not filter bets? What is classification report?

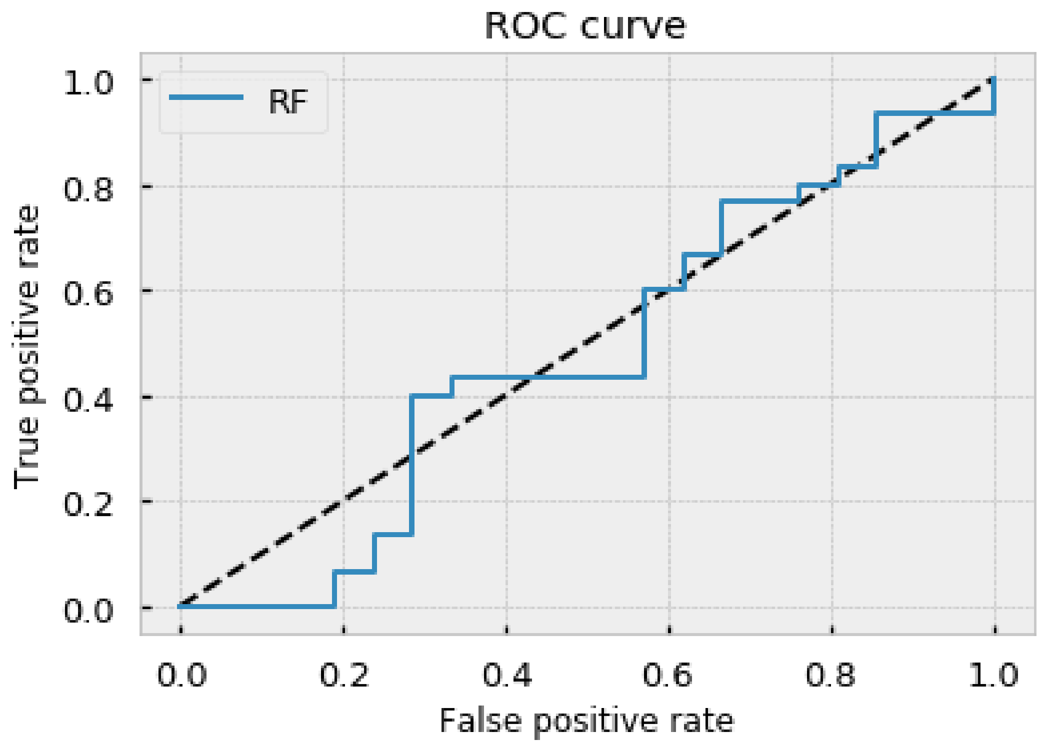
In [53]:
```
minRet = .01
ptsl=[0,2]
bb_events = getEvents(close,tEvents,ptsl,target,minRet,cpus,t1=t1)
cprint(bb_events)

bb_bins = getBins(bb_events,close).dropna()
cprint(bb_bins)

features = (pd.DataFrame()
            .assign(vol=bb_events.trgt)
            .assign(ma_side=ma_side)
            .assign(srl_corr=srl_corr)
            .drop_duplicates()
            .dropna())
cprint(features)

Xy = (pd.merge_asof(features, bb_bins[['bin']],
                    left_index=True, right_index=True,
                    direction='forward').dropna())
cprint(Xy)
```

```python
### run model ###
X = Xy.drop('bin',axis=1).values
y = Xy['bin'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, shuffle=False)

n_estimator = 10000
rf = RandomForestClassifier(max_depth=2, n_estimators=n_estimator,
                            criterion='entropy', random_state=RANDOM_STATE)
rf.fit(X_train, y_train)

# The random forest model by itself
y_pred_rf = rf.predict_proba(X_test)[:, 1]
y_pred = rf.predict(X_test)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)
print(classification_report(y_test, y_pred))

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

```
2018-10-09 19:40:36.802669 100.0% applyPtSlOnT1 done after 0.0 minutes. Remaining 0.0 minutes.


-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                                     t1      trgt
2018-02-13 13:43:37 2018-02-14 13:53:59  0.014365
2018-02-14 10:30:48 2018-02-15 10:42:27  0.012136
2018-02-14 13:36:02 2018-02-15 13:42:09  0.011688
2018-02-15 09:31:56 2018-02-16 09:42:36  0.011244
2018-02-15 14:05:41 2018-02-16 14:15:08  0.010183
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 929 entries, 2009-10-05 14:55:48 to 2018-02-15 14:05:41
Data columns (total 2 columns):
t1      929 non-null datetime64[ns]
trgt    929 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 21.8 KB
None
```

```
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                          ret  bin
2018-02-13 13:43:37  0.010108  1.0
2018-02-14 10:30:48  0.010876  1.0
2018-02-14 13:36:02  0.005056  1.0
2018-02-15 09:31:56  0.003964  1.0
2018-02-15 14:05:41  0.004842  1.0
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 929 entries, 2009-10-05 14:55:48 to 2018-02-15 14:05:41
Data columns (total 2 columns):
ret    929 non-null float64
bin    929 non-null float64
dtypes: float64(2)
memory usage: 21.8 KB
None
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                          vol  ma_side  srl_corr
2016-07-07 14:28:00  0.012624     -1.0  0.251865
2016-07-08 09:30:57  0.011944      1.0  0.238590
2018-02-06 10:18:08  0.013317     -1.0  0.123961
2018-02-07 15:28:09  0.024870      1.0 -0.005597
2018-02-13 09:30:00  0.017363     -1.0  0.198935
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 102 entries, 2009-10-29 13:40:22 to 2018-02-13 09:30:00
Data columns (total 3 columns):
vol         102 non-null float64
ma_side     102 non-null float64
srl_corr    102 non-null float64
dtypes: float64(3)
memory usage: 3.2 KB
None
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
dataframe information
-------------------------------------------------------------------------------
                          vol  ma_side  srl_corr  bin
2016-07-07 14:28:00  0.012624     -1.0  0.251865  1.0
```

```
2016-07-08 09:30:57  0.011944      1.0  0.238590  1.0
2018-02-06 10:18:08  0.013317     -1.0  0.123961  1.0
2018-02-07 15:28:09  0.024870      1.0 -0.005597 -1.0
2018-02-13 09:30:00  0.017363     -1.0  0.198935  1.0
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 102 entries, 2009-10-29 13:40:22 to 2018-02-13 09:30:00
Data columns (total 4 columns):
vol         102 non-null float64
ma_side     102 non-null float64
srl_corr    102 non-null float64
bin         102 non-null float64
dtypes: float64(4)
memory usage: 4.0 KB
None
-------------------------------------------------------------------------------

              precision    recall  f1-score   support

        -1.0       0.39      0.43      0.41        21
         1.0       0.57      0.53      0.55        30

   micro avg       0.49      0.49      0.49        51
   macro avg       0.48      0.48      0.48        51
weighted avg       0.50      0.49      0.49        51
```


ROC curve
1.0
0.8
0.6
0.4
0.2
0.0
True positive rate
RF
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate

In [ ]: