Modeling High-Frequency Limit Order Book Dynamics Using Machine Learning

In [2]:
```
%pylab inline
import pandas as pd
import numpy as np
import time
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import (RandomForestClassifier, ExtraTreesClassifier,
                              GradientBoostingClassifier)
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [10]:
```
def read_csv(day_trade):
    data_up = []
    data_down = []
    path = '/home/rory/Demo/Data_Transformation/Train_Test_Builder/order_boo
    for j,i in enumerate(day_trade):
        for k in range(0,len(i),1):
            path_up = path + '_' + str(j+1) + '_' + str(i[k]) + '_' + 'UP' +
            path_down = path + '_' + str(j+1) + '_' + str(i[k]) + '_' + 'DOW
            data_up.append(pd.read_csv(path_up))
            data_down.append(pd.read_csv(path_down))
            #print path_down
    return data_up,data_down
```

Example : 2014/1/2

In [11]:
```
day_trade = [[2]]
```

In [12]:
```
data_2014_up, data_2014_down = read_csv(day_trade)
```

Column = 0 : label[0 : not traded,1 : traded] & Column = 1~ : Features values

In [15]:
```
data_2014_up[0].head(10)
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	5
0	0.0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	...	2.56976
1	0.0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	...	2.56976
2	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.32098
3	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.32098
4	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.03606
5	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.03606
6	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.03606
7	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.03744
8	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.03744
9	1.0	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	0.21097	...	0.04299

10 rows × 65 columns

Machine learning algorithms

In [16]:
```
models = {
    'RandomForestClassifier': RandomForestClassifier(random_state = 0),
    'ExtraTreesClassifier': ExtraTreesClassifier(random_state = 0),
    'AdaBoostClassifier': AdaBoostClassifier(base_estimator = DecisionTreeCl
                                             n_estimators = 10,random_state
    'GradientBoostingClassifier': GradientBoostingClassifier(random_state =
    'SVC': SVC(probability=True,random_state = 0),
}
```

Grids for Hyperparameter Tuning

In [17]:
```
model_grid_params = {
    'RandomForestClassifier': {'max_features':[None],'n_estimators':[10],'ma
                               'min_samples_split':[2],'criterion':['entropy
                               'min_samples_leaf':[3]},
    'ExtraTreesClassifier': {'max_features':[None],'n_estimators':[10],'max_
                             'min_samples_split':[2],'criterion':['entropy']
                             'min_samples_leaf':[3]},
    'AdaBoostClassifier': {"base_estimator__criterion" : ["entropy"],\
                           "base_estimator__max_depth": [None],\
                           "base_estimator__min_samples_leaf" : [3],\
                           "base_estimator__min_samples_split" : [2],\
                           "base_estimator__max_features" : [None]},
    'GradientBoostingClassifier': {'max_features':[None],'n_estimators':[10]
                                   'min_samples_split':[2],'min_samples_leaf
                                   'learning_rate':[0.1],'subsample':[1.0]},
    'SVC': [{'kernel':['rbf'],'gamma':[1e-1],'C':[1]},\
            {'kernel':['linear'],'C':[1,10]}]
}
```

Model Selection Pipline

In [26]:
```
class Model_Selection:

    def __init__(self,models,model_grid_params,data_2014,latest_sec,pred_sec

        self.models = models
        self.model_grid = model_grid_params
        self.data_2014 = data_2014
        self.latest_sec = latest_sec
        self.pred_sec = pred_sec
        self.day = day
        self.keys = models.keys()
        self.best_score = {}
        self.grid = {}
        self.predict_values = {}
        self.cv_acc = {}
        self.acc = {}
        self.fscore = {}
        self.true_values = {}
        self.predict_values_day = {}
        self.cv_acc_day = {}
        self.acc_day = {}
        self.fscore_day = {}
        self.true_values_day = {}
        self.summary_day = []

    def Grid_fit(self,X_train,y_train,cv = 5,scoring = 'accuracy'):

        for key in self.keys:
            print "Running GridSearchCV for %s." %(key)
            model = self.models[key]
            model_grid = self.model_grid[key]
            Grid = GridSearchCV(model, model_grid, cv = cv, scoring = scorin
            Grid.fit(X_train,y_train)
            self.grid[key] = Grid
            print Grid.best_params_
            print 'CV Best Score = %s'%(Grid.best_score_)
            self.cv_acc[key].append(Grid.best_score_)

    def model_fit(self,X_train, y_train, X_test, y_test):

        for key in self.keys:
            print "Running training & testing for %s." %(key)
            model = self.models[key]
            model.set_params(**self.grid[key].best_params_)
            model.fit(X_train, y_train)
            predictions = model.predict(X_test)
            #print 'Prediction latest 15 second = %s'%(predictions)
            self.predict_values[key].append(predictions.tolist())
            self.true_values[key].append(y_test.tolist())
            acc = metrics.accuracy_score(y_test,predictions)
            f_score = metrics.f1_score(y_test,predictions)
            print 'Accuracy = %s'%(acc)
            self.acc[key].append(acc)
            self.fscore[key].append(f_score)

            if key == 'SVC':
```

```
            if self.grid[key].best_params_.values()[0] == 'linear':
                feature_imp = dict(zip([i for i in range(0,64,1)],model.
                Top_five = sorted(feature_imp.items(),key = lambda x : x
                #print 'Kernel is linear and top five importance feature
            else:
                #print 'Kernel is rbf'
                pass
        else:
            feature_imp = dict(zip([i for i in range(0,64,1)],model.feat
            Top_five = sorted(feature_imp.items(),key = lambda x : x[1]
            #print 'Top five importance features = %s'%(Top_five)
            pass

    def pipline(self):

        self.set_list_day() # store day values
        for day in range(0,self.day,1):
            self.set_list() # store values
            print 'Day = %s'%(day+1)
            for i in range(0,10,self.pred_sec):#9000-self.latest_sec-600,sel

                print '--------------------Rolling Window Time = %s---------
                # Train data
                data_train = self.data_2014[day][i:i+self.latest_sec]
                X_train = data_train.drop(['0'],axis=1)#,'65','66','67'],axi
                y_train = data_train['0']

                # Test data
                data_test = self.data_2014[day][i + self.latest_sec:i + self
                X_test = data_test.drop(['0'],axis=1)#,'65','66','67'],axis=
                y_test = data_test['0']

                #start = time.time()
                self.Grid_fit(X_train, y_train, cv = 5, scoring = 'accuracy'
                self.model_fit(X_train, y_train,X_test,y_test)
                #end = time.time()
                #print 'Total Time = %s'%(end - start)

            for key in self.keys:

                self.cv_acc_day[key].append(self.cv_acc[key])
                self.acc_day[key].append(self.acc[key])
                self.fscore_day[key].append(self.fscore[key])
                self.true_values_day[key].append(self.true_values[key])
                self.predict_values_day[key].append(self.predict_values[key]

            self.summary_day.append(self.score_summary(sort_by = 'Accuracy_m

    def set_list(self):

        for key in self.keys:
            self.predict_values[key] = []
            self.cv_acc[key] = []
            self.acc[key] = []
            self.fscore[key] = []
            self.true_values[key] = []
```

```
    def set_list_day(self):

        for key in self.keys:
            self.predict_values_day[key] = []
            self.cv_acc_day[key] = []
            self.acc_day[key] = []
            self.fscore_day[key] = []
            self.true_values_day[key] = []

    def score_summary(self,sort_by):

        summary = pd.concat([pd.DataFrame(self.acc.keys()),pd.DataFrame(map(
                             pd.DataFrame(map(lambda x: std(self.acc[x]), se
                             pd.DataFrame(map(lambda x: max(self.acc[x]), se
                             pd.DataFrame(map(lambda x: min(self.acc[x]), se
                             pd.DataFrame(map(lambda x: mean(self.fscore[x])
        summary.columns = ['Estimator','Accuracy_mean','Accuracy_std','Accur
        summary.index.rename('Ranking', inplace=True)
        return summary.sort_values(by = [sort_by], ascending=False)

    def print_(self):
        print self.predict_values
```

In [27]:
```
latest_sec = 60 * 30
pred_sec = 10
day = 1
data_2014_up, data_2014_down = read_csv(day_trade)
data_2014 = data_2014_up
pip = Model_Selection(models,model_grid_params,data_2014,latest_sec,pred_sec
```

Start Machine Learning Pipline

In [28]:
```
start = time.time()
pip.pipline()
end = time.time()
print 'Total Time = %s'%(end-start)
```

```
Day = 1
--------------------Rolling Window Time = 0--------------------
Running GridSearchCV for SVC.
{'kernel': 'rbf', 'C': 1, 'gamma': 0.1}
CV Best Score = 0.818333333333
Running GridSearchCV for AdaBoostClassifier.
{'base_estimator__min_samples_split': 2, 'base_estimator__criterion': 'en
tropy', 'base_estimator__max_depth': None, 'base_estimator__min_samples_l
eaf': 3, 'base_estimator__max_features': None}
CV Best Score = 0.710555555556
Running GridSearchCV for GradientBoostingClassifier.
{'subsample': 1.0, 'learning_rate': 0.1, 'min_samples_leaf': 3, 'n_estima
tors': 10, 'min_samples_split': 2, 'max_features': None, 'max_depth': 10}
CV Best Score = 0.717222222222
Running GridSearchCV for ExtraTreesClassifier.
{'min_samples_leaf': 3, 'n_estimators': 10, 'min_samples_split': 2, 'crit
erion': 'entropy', 'max_features': None, 'max_depth': 10}
CV Best Score = 0.740555555556
Running GridSearchCV for RandomForestClassifier.
{'min_samples_leaf': 3, 'n_estimators': 10, 'min_samples_split': 2, 'crit
erion': 'entropy', 'max_features': None, 'max_depth': 10}
CV Best Score = 0.722777777778
Running training & testing for SVC.
Accuracy = 1.0
Running training & testing for AdaBoostClassifier.
Accuracy = 1.0
Running training & testing for GradientBoostingClassifier.
Accuracy = 1.0
Running training & testing for ExtraTreesClassifier.
Accuracy = 1.0
Running training & testing for RandomForestClassifier.
Accuracy = 1.0
Total Time = 16.1451020241
```

Metrics

In [236]:
```
pip.summary_day[0]#.reset_index(drop = True)
```

Out[236]:

Ranking	Estimator	Accuracy_mean	Accuracy_std	Accuracy_max	Accuracy_min	F_
1	AdaBoostClassifier	0.929091	0.188139	1.0	0.0	0.6
0	ExtraTreesClassifier	0.923939	0.203313	1.0	0.0	0.6
4	RandomForestClassifier	0.914394	0.215931	1.0	0.0	0.6
3	GradientBoostingClassifier	0.896667	0.242876	1.0	0.0	0.6
2	SVC	0.805000	0.333106	1.0	0.0	0.6

In [235]:
```
pip.summary_day[1]#.reset_index(drop = True)
```

Out[235]:

Ranking	Estimator	Accuracy_mean	Accuracy_std	Accuracy_max	Accuracy_min	F_
0	ExtraTreesClassifier	0.956212	0.147322	1.0	0.0	0.6
1	AdaBoostClassifier	0.949545	0.155456	1.0	0.1	0.6
4	RandomForestClassifier	0.942273	0.176629	1.0	0.0	0.6
3	GradientBoostingClassifier	0.922121	0.198544	1.0	0.0	0.6
2	SVC	0.842727	0.307125	1.0	0.0	0.6

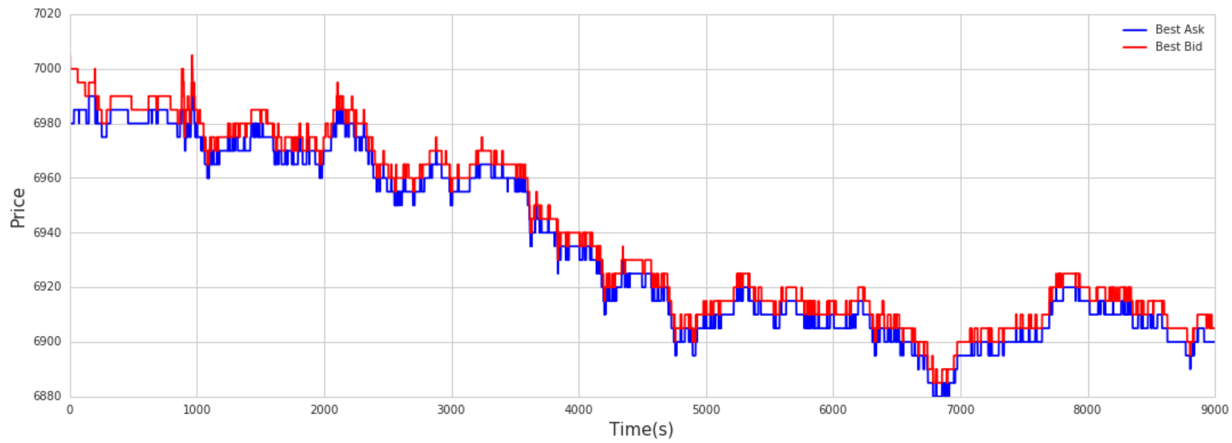
In [279]:
```
pip.summary_day[2]#.reset_index(drop = True)
```

Out[279]:

Ranking	Estimator	Accuracy_mean	Accuracy_std	Accuracy_max	Accuracy_min	F_
0	ExtraTreesClassifier	0.954091	0.162084	1.0	0.0	0.8
1	AdaBoostClassifier	0.950303	0.167105	1.0	0.0	0.8
4	RandomForestClassifier	0.947879	0.169165	1.0	0.0	0.8
3	GradientBoostingClassifier	0.930758	0.201380	1.0	0.0	0.7
2	SVC	0.896970	0.258240	1.0	0.0	0.7

In [383]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (18,6))
color_ = ['r','b']
plot(data_2014[1]['66'],label = 'Best Ask',color = color_[1])
plot(data_2014[1]['67'],label = 'Best Bid',color = color_[0])
plt.legend(loc=0)
plt.xlabel('Time(s)',size = 15)
plt.ylabel('Price',size = 15)
```

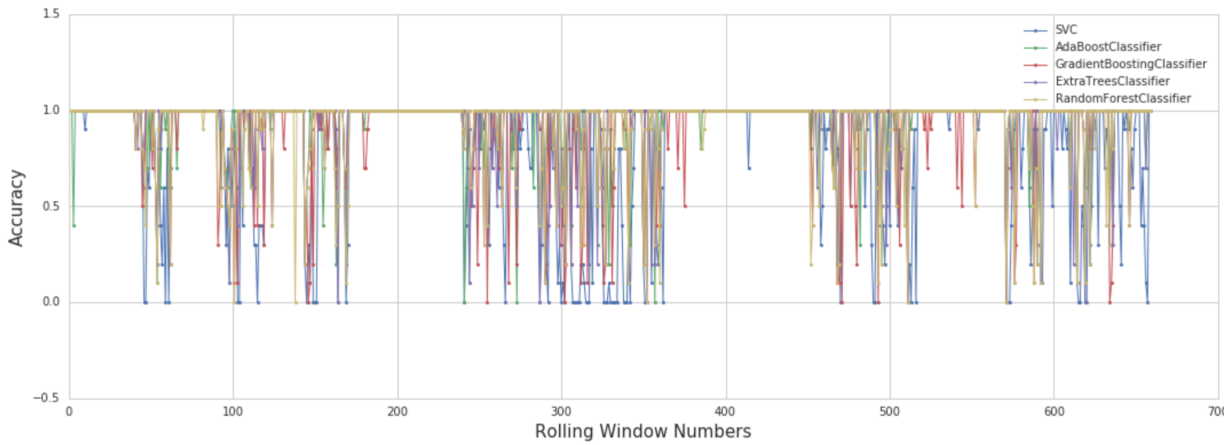
Out[383]: <matplotlib.text.Text at 0x7f680dc3d710>



Accuracy in one day

In [420]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (18,6))
color = []
for key in pip.keys:
    plot(np.array(pip.acc_day[key])[0],'-o',label = key,lw = 1,markersize =
    plt.legend(loc=0)
plt.ylim(-0.5,1.5)
plt.legend(loc=0)
plt.xlabel('Rolling Window Numbers',size = 15)
plt.ylabel('Accuracy',size = 15)
```

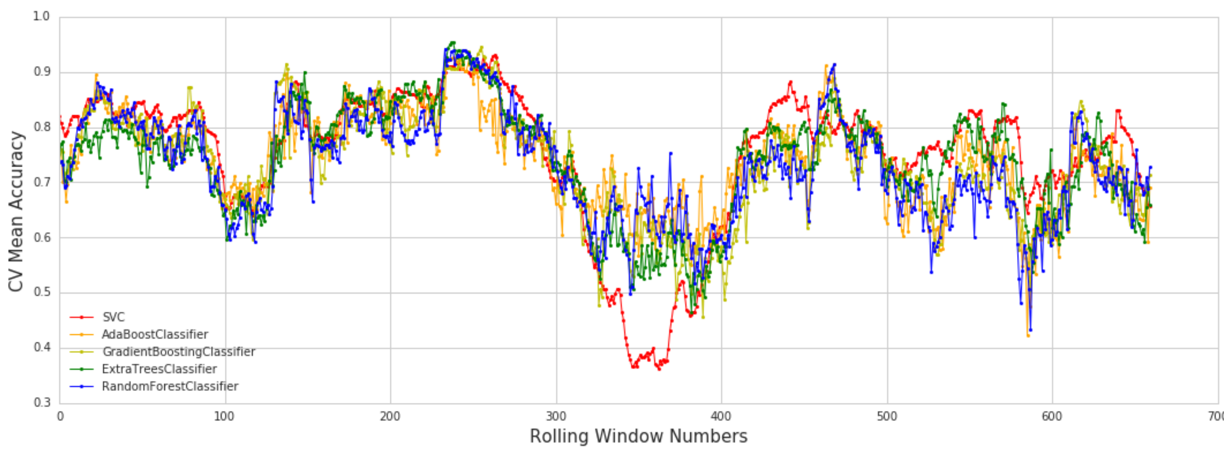
Out[420]: <matplotlib.text.Text at 0x7f6801138250>



Cross Validation

In [247]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (18,6))
color_ = ['r','orange','y','g','b']
for index,key in enumerate(pip.keys):
    plot(np.array(pip.cv_acc_day[key])[0],'-o',label = key,color = color_[in
#plot(best_cv_score,'-v',label = 'Best cv 5 folds score',color = 'violet',lw
plt.legend(loc=0)
plt.xlabel('Rolling Window Numbers',size = 15)
plt.ylabel('CV Mean Accuracy',size = 15)
```

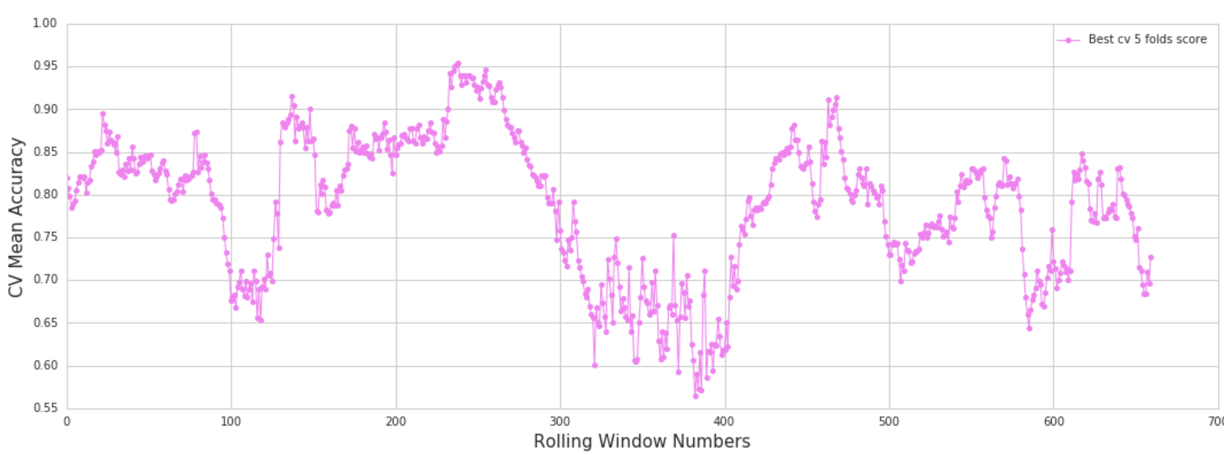
Out[247]: <matplotlib.text.Text at 0x7f6812428350>



Best Model

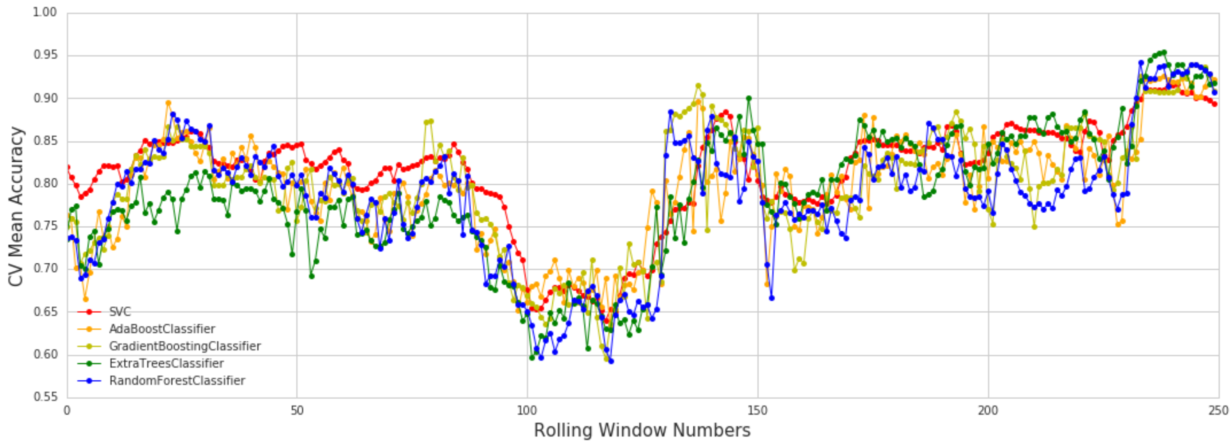
In [282]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (18,6))
plot(best_cv_score,'-o',label = 'Best cv 5 folds score',color = 'violet',lw
plt.legend(loc=0)
plt.xlabel('Rolling Window Numbers',size = 15)
plt.ylabel('CV Mean Accuracy',size = 15)
```

Out[282]: <matplotlib.text.Text at 0x7f68110e5cd0>



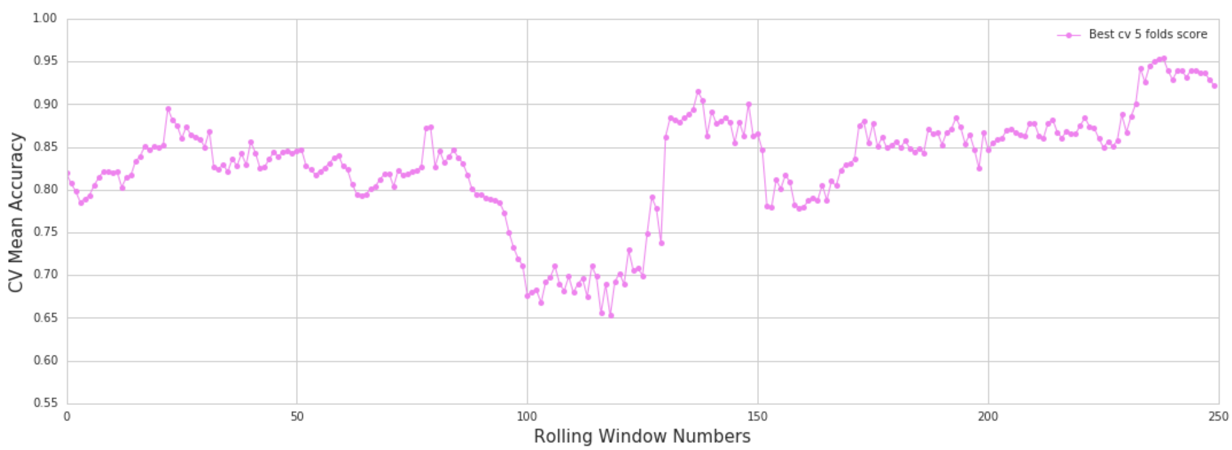
In [365]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (18,6))
color_ = ['r','orange','y','g','b']
for index,key in enumerate(pip.keys):
    plot(np.array(pip.cv_acc_day[key])[0][0:250],'-o',label = key,color = co
#plot(best_cv_score,'-v',label = 'Best cv 5 folds score',color = 'violet',lw
plt.legend(loc=0)
plt.xlabel('Rolling Window Numbers',size = 15)
plt.ylabel('CV Mean Accuracy',size = 15)
plt.ylim(0.55,1)
```

Out[365]: (0.55, 1)



In [366]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (18,6))
plot(best_cv_score[0:250],'-o',label = 'Best cv 5 folds score',color = 'viol
plt.legend(loc=0)
plt.xlabel('Rolling Window Numbers',size = 15)
plt.ylabel('CV Mean Accuracy',size = 15)
plt.ylim(0.55,1)
```

Out[366]: (0.55, 1)



Profit & Loss

In [344]:
```
# compute cum_profit and Best_cv_score
dict_ = {}
dict_['cum_profit'] = []
dict_['Best_cv_score'] = []

for day in range(0,1,1):
    cum_profit_label = []
    cum_profit = []
    best_cv_score = []
    spread = 0.2 * data_2014[day]['65'][1800:][9::10].values
    loss = 0.2*(data_2014[0]['67'][1800:9000-600][9::10].values - data_2014[
    for j in range(0,len(pip.cv_acc_day.values()[0][day]),1):
        max_al = {}
        for i in range(0,len(pip.keys),1):
            max_al[pip.keys[i]] = np.array(pip.cv_acc_day[pip.keys[i]])[day]
        # select best algorithm in cv = 5
        top_cv_acc = sorted(max_al.items(),key = lambda x : x[1], reverse =
        best_cv_score.append(top_cv_acc[1])
        submission = pip.predict_values_day[top_cv_acc[0]][day][j][-1]
        true_value = pip.true_values_day[top_cv_acc[0]][day][j][-1]

        if submission == true_value:
            if submission == 1:
                cum_profit_label.append(1)
                cum_profit.append(spread[j])
            elif submission == 0:
                cum_profit_label.append(0)
                cum_profit.append(0)
        elif submission != true_value:
            if submission == 1:
                cum_profit_label.append(-1)
                cum_profit.append(loss[j])
            elif submission == 0:
                cum_profit_label.append(0)
                cum_profit.append(0)

    dict_['cum_profit'].append(cum_profit)
    dict_['Best_cv_score'].append(best_cv_score)
```

In [414]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (20,8))
plt.subplot(211)
plot(cum_profit,'-o',label = 'Profit & Loss',lw = 1,markersize = 3)
plt.ylabel('Tick',size = 15)
plt.legend(loc=0)
plt.ylim(-7.5,2.5)
plt.subplot(212)
plot(cumsum(cum_profit),'-o',label = 'Cum Profit',lw = 1,markersize = 2)
plt.legend(loc=0)
plt.xlabel('Rolling Window Numbers',size = 15)
plt.ylabel('Profit',size = 15)
```

Out[414]: <matplotlib.text.Text at 0x7f680164e210>