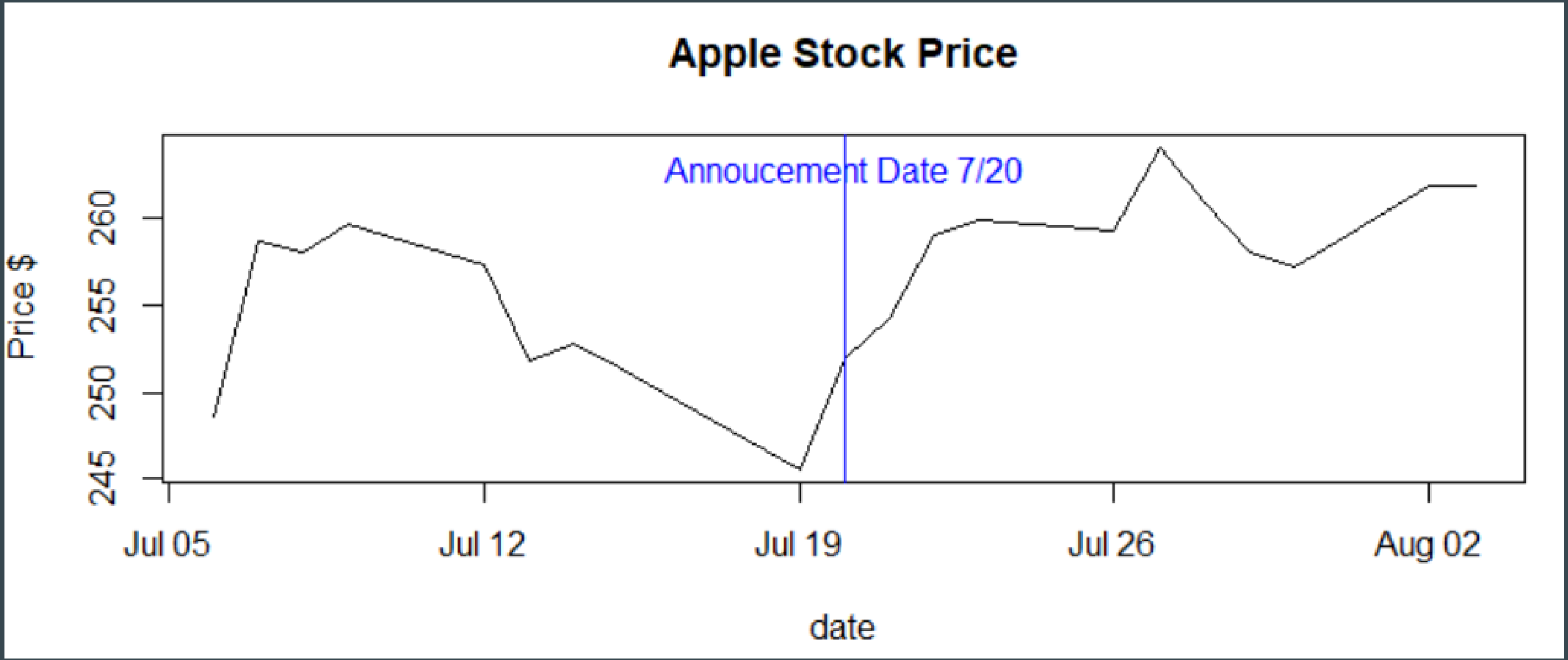
Predicting Post-Earnings Announcement Drift

•••

Example: Apple Announcing Earning 2010/07/20



Overview

U.S. stocks have been shown to earn higher returns during earnings announcement months than during non-announcement months.

The magnitude of this earnings announcement premium has been estimated by Frazzini and Lamont (2007) to be over 7% per year. While a number of potential explanations for the premium have been put forward, uncertainty remains over the reasons for its existence.

There are two goals of the projects.

- The first is to investigate premium extends over the companies traded in the three major exchanges in the U.S., thereby providing out-of-sample evidence of its existence.
- The second is to exploit observed variations in the direction of the premium in order to gain insights into the factors driving this difference.

Data

Selection and formation of data;
Variables and intuitions

$$Ret_{it} = \alpha_t + \beta_{1t} ifpositiveDrift_{it} + \beta_{2t} Past30Vol_{it}$$
$$+\beta_{3t} Past10Vol_{it} + \beta_{4t} PastMktVol_{it} + \beta_{5t} Ret5dbefore_{it}$$
$$+\beta_{6t} Ret1dbefore_{it} + \beta_{7t} past200dSK_{it}$$
$$+\beta_{8t} past200dKur_{it} + \beta_{9t} PR_{it} + \sum_{j=1}^{3} \gamma_{jt} Quarter_{jt}$$

Data

CRSP Daily Stock

- Period of 2010 - 2018
- Return ±5, 10, 30 day
- Vol -10 day
- Lag_Returns + 5 day
- Lag_Market Cap
- - 200 day Skewness

Compustat Quarterly

- Earnings report date
- Fiscal Quarter
- Change in Cash
- Shares Repurchase

CRSP Daily Market

- Past 3 Month Market Volatility

Variables

$$Ret_{it} = \alpha_t + \beta_{1t} ifpositiveDrift_{it} + \beta_{2t} Past30Vol_{it} + \beta_{3t} Past10Vol_{it} + \beta_{4t} PastMktVol_{it} + \beta_{5t} Ret5dbefore_{it} + \beta_{6t} Ret1dbefore_{it} + \beta_{7t} past200dSK_{it} + \beta_{8t} past200dKur_{it} + \beta_{9t} PR_{it} + \sum_{j=1}^{3} \gamma_{jt} Quarter_{jt}$$

IfPositiveDrift, Quarter

- IfPositiveDrift=1, if lag(Ret5dayAfter)>0; IfPositiveDrift=0, if lag(Ret5dayAfter)<0
- Quarter_j: 3 dummies to count the seasonality effect.

PastVol

- PastVol: use past 30 and past 10 days' volatilities as inputs.
- PastMktVol: past quarter's market volatility before announcement day.

PR

- From Ball and Kothari(1991), PR measures investor sophistication.

PR_i=[log(max MV)−log(MV_i)]/[log(max MV)−log(min MV)], max MV=\$99 billion, min MV=\$1 million, and MV_i=market value of equity at the beginning of firm-quarter q for firm i, in millions.

past200dSK & past200dKur

- From McNichols(1988), earnings reports causes more 'extreme bad news' to be reflected in stock prices.
- Use past 200 days stock returns before announcement days to calculate skewness and kurtosis.

Models

Panel Regression

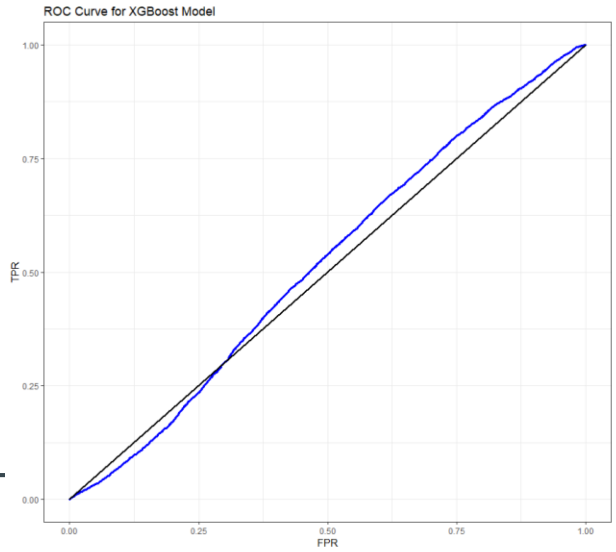
R2 score is 3.6%, but this is on individual stocks, not the portfolio

```
Call:
   felm(formula = signs ~ x | fyearq + gvkey | 0 | fyearq + gvkey,      data = w1)

Residuals:
    Min      1Q  Median      3Q     Max
-1.1897 -0.4051  0.0818  0.3753  1.1421

Coefficients:
                          Estimate Cluster s.e. t value Pr(>|t|)
xfqtr                     1.179e-02    3.341e-02   0.353   0.7241
xRet5DayBefore            1.327e-01    7.806e-02   1.701   0.0890 .
xPast10dayVol             7.098e-01    4.306e-01   1.648   0.0993 .
xifPositiveDrift_Lag      2.988e-03    3.713e-03   0.805   0.4209
xPast30dayVol            -1.515e+00    6.671e-01  -2.271   0.0232 *
xPastMktVol3Month        -3.090e+00    2.195e+00  -1.408   0.1593
xLaggedME                -2.315e-09    1.055e-09  -2.195   0.0282 *
xChangeInCashInvestment -1.064e-07    2.652e-06  -0.040   0.9680
xSharesRepurchased        1.684e-04    7.793e-04   0.216   0.8289
xLTDebtDue1Year           1.577e-06    5.063e-06   0.311   0.7555
xAmortization             2.071e-05    1.082e-04   0.191   0.8483
xRet1DayBefore           -1.384e-01    1.066e-01  -1.298   0.1945
xPast200SK                1.823e-03    2.346e-03   0.777   0.4371
xPast200Kur              -5.138e-04    2.576e-04  -1.995   0.0461 *
xPR                       1.452e+00    2.238e-01   6.485  9.1e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4645 on 18112 degrees of freedom
Multiple R-squared(full model): 0.2311   Adjusted R-squared: 0.1231
Multiple R-squared(proj model): 0.05514   Adjusted R-squared: -0.07752
F-statistic(full model, *iid*): 2.14 on 2543 and 18112 DF, p-value: < 2.2e-16
F-statistic(proj model): 81.44 on 15 and 5 DF, p-value: 6.074e-05
```

XGBoost

```
          Reference
Prediction    0    1
         0 1968 6162
         1 1714 7118

               Accuracy : 0.5357
                 95% CI : (0.5281, 0.5432)
```

```
> lift1
    deciles y_data_test mean_response
 1:       1   0.4502062     0.8646151
 2:       2   0.4752358     0.9126841
 3:       3   0.5247642     1.0078026
 4:       4   0.5212264     1.0010084
 5:       5   0.5353774     1.0281851
 6:       6   0.5495283     1.0553618
 7:       7   0.5854953     1.1244359
 8:       8   0.5784198     1.1108475
 9:       9   0.5318396     1.0213909
10:      10   0.4549204     0.8736687
```



Logistic Regression

Best	Worst	Accuracy	F1 score
0.61	0.57	0.76	0.56

```
clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(x_train, y_train)
```

```
/Users/ycli/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarnin
led to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

```
from sklearn.metrics import f1_score
```

```
y_pred = clf.predict(x_test)
```

```
f1_score(y_test,y_pred)
```

```
0.6815907522429262
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.5647149006427266
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

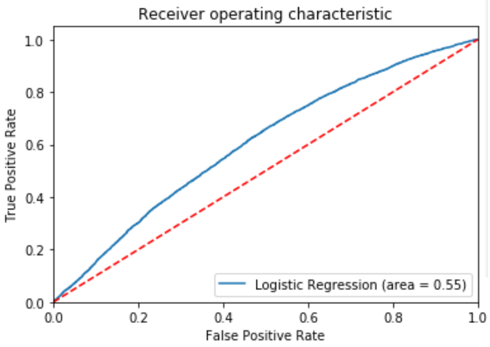
```
[[1676 6451]
 [ 931 7901]]
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.21      0.31      8127
           1       0.55      0.89      0.68      8832

    accuracy                           0.56     16959
   macro avg       0.60      0.55      0.50     16959
weighted avg       0.59      0.56      0.50     16959
```

TN 1696	FP 6451
FN 931	TP 7901

```
library(dplyr)
phat <- predict(out, type = "response")
phat <- predict(out, data = ww2, type = 'response')
deciles <- ntile(phat, n = 10)
df <- data.table(deciles=deciles,phat=phat,signs=ww1$signs)
lift <- aggregate(df,by=list(deciles),FUN="mean",data=df)
lift <- lift[,c(2,4)]
lift[,3] <- lift[,2]/mean(w$signs)
names(lift) <- c("decile","Mean Response","Lift Factor")
lift
```




```
> lift
   decile Mean Response Lift Factor
1       1     0.3296225   0.6059885
2       2     0.4574056   0.8409092
3       3     0.5104116   0.9383572
4       4     0.5488867   1.0090911
5       5     0.5840194   1.0736800
6       6     0.5934172   1.0909574
7       7     0.6305085   1.1591471
8       8     0.6413359   1.1790526
9       9     0.6852300   1.2597490
10     10     0.7495644   1.2941801
```

Decision Trees

Max depth	Best	Worst	Accuracy	F1 score
3 gini	0.67	0.60	0.546	0.682
5 gini	0.62	0.68	0.55	0.66
3 entropy	0.67	0.59	0.543	0.683

```
from sklearn.tree import DecisionTreeClassifier

tree_clf2 = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf2.fit(x_test, y_test)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=42, splitter='best')
```

```
y_pred2 = tree_clf2.predict(x_test)
accuracy_score(y_test, y_pred2)
```

```
0.6150126776342945
```

```
f1_score(y_test,y_pred2)
```

```
0.672944948154085
```

```
cross_val_score(tree_clf2, x_test, y_test, cv=10)
```

```
array([0.60518562, 0.608132  , 0.59316038, 0.60554245, 0.6120283 ,
       0.60495283, 0.61497642, 0.60117994, 0.60530973, 0.620059  ])
```

```
tree_clf10 = DecisionTreeClassifier(max_depth=10, random_state=42)
tree_clf10.fit(x_test, y_test)
```

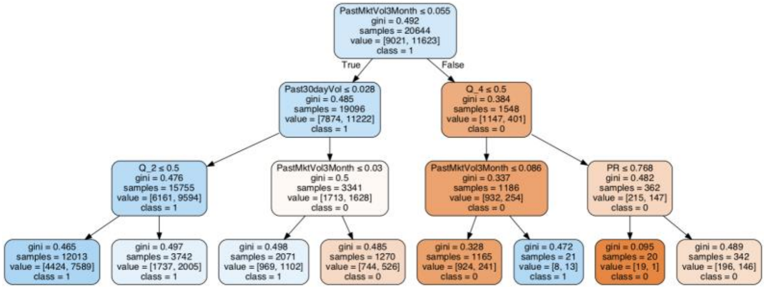
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=42, splitter='best')
```

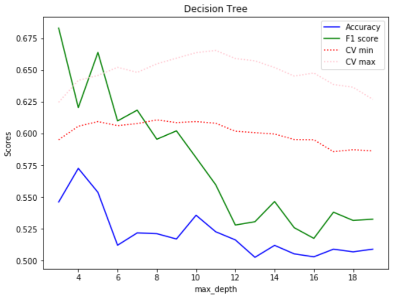
```
y_pred10 = tree_clf10.predict(x_test)
accuracy_score(y_test, y_pred10)
```

```
0.7128958075358217
```

```
acc, f1, cross_min, cross_max = [], [], [], []
for i in np.arange(3,20,1):
    tree_clfx = DecisionTreeClassifier(random_state=42,criterion='gini', max_depth = i)
    tree_clfx.fit(x_train,y_train)
    y_predx = tree_clfx.predict(x_test)
    print(i, accuracy_score(y_test, y_predx), f1_score(y_test,y_predx), \
          min(cross_val_score(tree_clfx, x_test, y_test, cv=10)), \
              max(cross_val_score(tree_clfx, x_test, y_test, cv=10)))
    acc.append(round(accuracy_score(y_test, y_predx),4))
    f1.append(round(f1_score(y_test,y_predx),4))
    cross_min.append(round(min(cross_val_score(tree_clfx, x, y, cv=10)),4))
    cross_max.append(round(max(cross_val_score(tree_clfx, x, y, cv=10)),4))
```

```
3 0.5461406922577983 0.6830031712038219 0.5990566037735849 0.671386430678466
4 0.5725573441830296 0.6204115829711473 0.6267688679245284 0.671386430678466
5 0.5537472728344832 0.663823738450604 0.6206489675516225 0.6821933962264151
6 0.5121174597558819 0.6099009900990099 0.6281673541543901 0.6816037735849056
7 0.52184680700513 0.6183820415078358 0.629345904537419 0.6892688679245284
8 0.5211981838551801 0.595536959553696 0.6202830188679245 0.6892688679245284
9 0.5169526505100537 0.6020596521908093 0.6240424278137 89 0.6827830188679245
10 0.5357037561176956 0.5810365010109609 0.6140247495580436 0.6792452830188679
11 0.52273129311869 8 0.5597258485639687 0.6114386792452831 0.6739386792452831
12 0.5163040273601038 0.5280478683620046 0.6096698113207547 0.6686320754716981
13 0.5026239754702518 0.5306326859941015 0.5955188679245284 0.6550707547169812
14 0.5119995282740728 0.5464708461201228 0.5984669811320755 0.6533018867924528
15 0.5053363995518604 0.5259112743712914 0.5880966411314084 0.6379716981132075
16 0.502977769915679 0.517488121815788 0.5934001178550383 0.6344339622641509
17 0.5089922754879415 0.5380817662395296 0.5875073659398939 0.6344339622641509
18 0.5068695088153783 0.5316157938952674 0.5849056603773585 0.6344339622641509
19 0.5089922754879415 0.5325849003648611 0.5837264150943396 0.6373820754716981
```





Cross validation = 10, worse results longer depths -> overfitting. 4, 5 are the 'best' in out-of-sample, but overall the same in cross-validation, pick priority

Random Forest Classification

Max depth	Best	Worst	Accuracy	F1 score
3 gini	0.65	0.60	0.553	0.688
10 gini	0.67	0.65	0.559	0.662
3 entropy	0.67	0.59	0.541	0.550

```
from sklearn.ensemble import RandomForestClassifier
rnd_clf = RandomForestClassifier(random_state=42,max_depth=3)
```

```
rnd_clf.fit(x_train, y_train)
y_pred = rnd_clf.predict(x_test)
accuracy_score(y_test, y_pred)
```

```
/Users/ycli/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/for
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
0.5525679580163925
```

```
f1_score(y_test,y_pred)
```

```
0.6880190773785051
```

```
cross_val_score(rnd_clf, x_test, y_test, cv=10)
```

```
array([0.6010607 , 0.63523866, 0.62971698, 0.60318396, 0.61674528,
       0.61851415, 0.61025943, 0.65663717, 0.63362832, 0.64070796])
```

```
rnd_clf2 = RandomForestClassifier(random_state=42,max_depth=10)
rnd_clf2.fit(x_train,y_train)
y_pred2 = rnd_clf2.predict(x_test)
accuracy_score(y_test, y_pred2)
```

```
/Users/ycli/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/for
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
0.5599976413703638
```

```
cross_val_score(rnd_clf2, x_test, y_test, cv=10)
```

```
array([0.62816735, 0.67000589, 0.64150943, 0.64917453, 0.6692217 ,
       0.65860849, 0.64622642, 0.6660767 , 0.6560472 , 0.6259587 ])
```

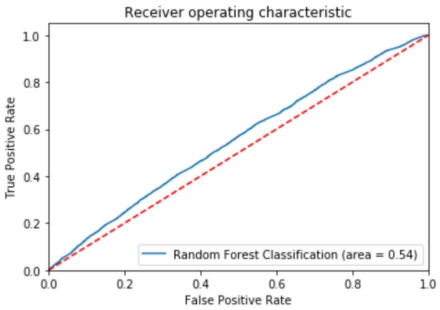
```
f1_score(y_test,y_pred2)
```

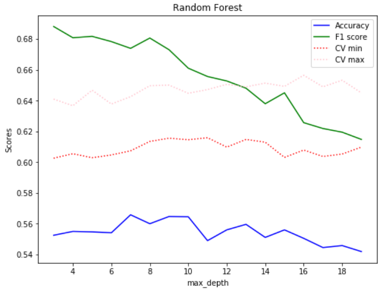
```
0.6622612473974835
```

```
import warnings
warnings.filterwarnings('ignore')
acc, f1, cross_min, cross_max = [], [], [], []
for i in np.arange(3,20,1):
    rnd_clfx = RandomForestClassifier(random_state=42,criterion='entropy', max_depth = i)
    rnd_clfx.fit(x_train,y_train)
    y_predx = rnd_clfx.predict(x_test)
    print(i, accuracy_score(y_test, y_predx), f1_score(y_test,y_predx), \
          min(cross_val_score(rnd_clfx, x_test, y_test, cv=10)), \
              max(cross_val_score(rnd_clfx, x_test, y_test, cv=10)))
    acc.append(round(accuracy_score(y_test, y_predx),4))
    f1.append(round(f1_score(y_test,y_predx),4))
    cross_min.append(round(min(cross_val_score(rnd_clfx, x, y, cv=10)),4))
    cross_max.append(round(max(cross_val_score(rnd_clfx, x, y, cv=10)),4))
```

```
3 0.5523910607936788 0.6881137269403016 0.5992928697701827 0.6513274336283186
4 0.5548676219116693 0.6807898854074168 0.6096698113207547 0.6631268436578172
5 0.5546317589480512 0.6816975009482068 0.6205067766647024 0.6647024160282852
6 0.5540421015390058 0.6782660484111116 0.6287566293459045 0.671184443134944
7 0.5657173182381037 0.6738697250143912 0.6287566293459045 0.6702064896755162
8 0.5599386756294593 0.6805632838248513 0.6340601060695344 0.6745283018867925
9 0.5645969691609175 0.673013904879993 0.6317030053034767 0.6721698113207547
10 0.5644200719382039 0.6610379479649429 0.6299351797289334 0.6774764150943396
11 0.5489120820803114 0.6555915721231766 0.6216853270477313 0.6709905660377359
12 0.555929005247951 0.6527092460225963 0.6216853270477313 0.6851415094339622
13 0.559466949702223 0.6480425872709286 0.6057748968768415 0.6682380671773719
14 0.5510348487528746 0.6378769142965851 0.6157925751325869 0.6686320754716981
15 0.5558700395070464 0.6449849170437406 0.6311137301119623 0.6586084905660378
16 0.5504451913438292 0.6256138283244943 0.5969357690041249 0.6627358490566038
17 0.5444306857715667 0.6217936166046603 0.6205067766647024 0.6739386792452831
18 0.5456689663305619 0.6194122005433439 0.6149764150943396 0.6662735849056604
19 0.5418361931717672 0.6146980065456709 0.6031839622641509 0.6609669811320755
```

Best performance:
max_depth 8.9 gini,
greater is overfitting





Bagging

```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier, ExtraTreesClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

seed = 42
# Create classifiers
rf = RandomForestClassifier()
et = ExtraTreesClassifier()
knn = KNeighborsClassifier()
lg = LogisticRegression()
clf_array = [rf, et, knn, lg]

for clf in clf_array:
    vanilla_scores = cross_val_score(clf, x_train, y_train, cv=10, n_jobs=-1)
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    fscore=f1_score(y_test,y_pred)
    bagging_clf = BaggingClassifier(clf, max_samples=0.4, max_features=10, random_state=seed)
    bagging_scores = cross_val_score(bagging_clf, x_train, y_train, cv=10, n_jobs=-1)
    bagging_clf.fit(x_train,y_train)
    y_pred2 = bagging_clf.predict(x_test)
    fscore2=f1_score(y_test,y_pred2)

    print ("Mean: {1:.3f}, std: (+/-) {2:.3f} [{0}]".format(clf.__class__.__name__,\
           vanilla_scores.mean(), vanilla_scores.std()))
    print ("Mean: {1:.3f}, std: (+/-) {2:.3f} [Bagging {0}]" .format(clf.__class__.__name__,\
           bagging_scores.mean(), bagging_scores.std()))
    print ("f1_score: {1:.3f} [{0}]".format(clf.__class__.__name__, fscore))
    print ("f1_score: {1:.3f} [Bagging {0}]\n".format(clf.__class__.__name__, fscore2))
```

```
Mean: 0.590, std: (+/-) 0.020 [RandomForestClassifier]
Mean: 0.621, std: (+/-) 0.017 [Bagging RandomForestClassifier]
f1_score: 0.546 [RandomForestClassifier]
f1_score: 0.651 [Bagging RandomForestClassifier]

Mean: 0.585, std: (+/-) 0.019 [ExtraTreesClassifier]
Mean: 0.625, std: (+/-) 0.017 [Bagging ExtraTreesClassifier]
f1_score: 0.544 [ExtraTreesClassifier]
f1_score: 0.637 [Bagging ExtraTreesClassifier]

Mean: 0.525, std: (+/-) 0.010 [KNeighborsClassifier]
Mean: 0.550, std: (+/-) 0.008 [Bagging KNeighborsClassifier]
f1_score: 0.570 [KNeighborsClassifier]
f1_score: 0.615 [Bagging KNeighborsClassifier]

Mean: 0.592, std: (+/-) 0.014 [LogisticRegression]
Mean: 0.584, std: (+/-) 0.010 [Bagging LogisticRegression]
f1_score: 0.683 [LogisticRegression]
f1_score: 0.681 [Bagging LogisticRegression]
```

Voting - soft/hard

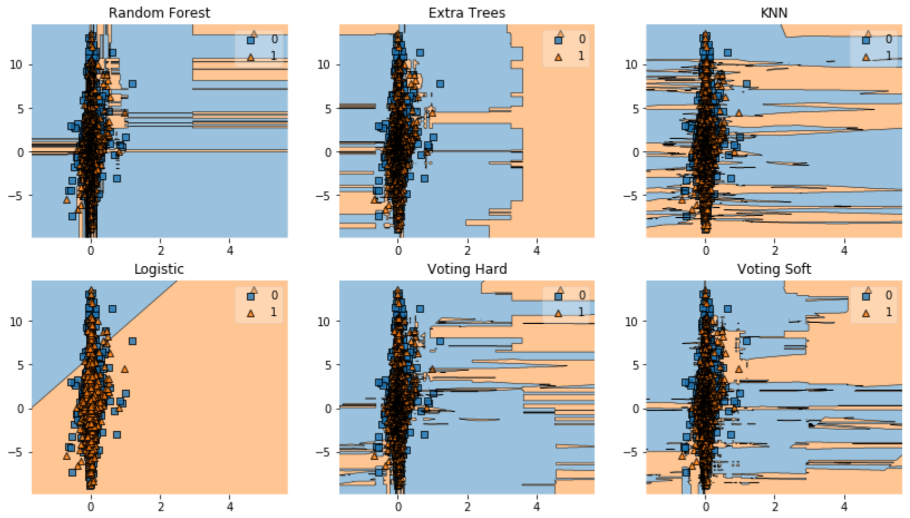
```python
# hard/soft voting

import warnings
warnings.filterwarnings("ignore")

from sklearn.ensemble import VotingClassifier
eclf = VotingClassifier(estimators=[('Random Forests', rf), ('Extra Trees', et), \
                                    ('KNeighbors', knn),('Logistic Classifier', lg)], voting='hard')
eclf2 = VotingClassifier(estimators=[('Random Forests', rf), ('Extra Trees', et), \
                                     ('KNeighbors', knn), ('Logistic Classifier', lg)], voting='soft')
for clf, label in zip([eclf,eclf2], ['Voting Hard','Voting Soft']):
    scores = cross_val_score(clf, x_train, y_train, cv=10, scoring='accuracy')
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    fscore=f1_score(y_test,y_pred)

    print("Mean: {1:.3f}, std: (+/-) {2:.3f} [{0}]" .format(label,scores.mean(), scores.std()))
    print ("f1_score: {1:.3f} [{0}]\n".format(label, fscore))
```

```
Mean: 0.595, std: (+/-) 0.020 [Voting Hard]
f1_score: 0.559 [Voting Hard]

Mean: 0.602, std: (+/-) 0.017 [Voting Soft]
f1_score: 0.635 [Voting Soft]
```

Decision Boundaries

```python
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
import itertools

gs = gridspec.GridSpec(3, 3)
fig = plt.figure(figsize=(14, 12))
labels = ['Random Forest', 'Extra Trees', 'KNN', 'Logistic', 'Voting Hard','Voting Soft']
for clf, lab, grd in zip([rf, et, knn, lg, eclf,eclf2], labels, \
                         itertools.product([0, 1, 2], repeat = 2)):
    clf.fit(x_train[['Ret5DayBefore', 'Past200SK']], y_train)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=np.array(x_train[['Ret5DayBefore', 'Past200SK']]), \
                                y=np.array(y_train), clf=clf)
    plt.title(lab)
```

Decision Boundaries



References

- Ball, R., Kothari, S., 1991. Security returns around earnings announcements. The Accounting Review 66, 718–738.
- Barber, B., De George, E., Lehavy, R., Trueman, B., 2013. The earnings announcement premium around the globe. Journal of Financial Economics 108, 118–138.
- Frazzini, A., Lamont, O., 2007. The earnings announcement premium and trading volume. NBER Working Paper No. 13090
- McNichols, M., 1988. A comparison of the skewness of stock return distributions at earn- ings and non-earnings announcement dates. Journal of Accounting and Economics 10, 239 273.