In [2]:

```
import pandas as pd
import glob,os

allFiles = glob.glob("data/*.csv")
df = pd.concat((pd.read_csv(f) for f in allFiles))
```

In [3]:

```
from itertools import chain

bookieArray = ['B365', 'BS', 'PSC', 'GB', 'IW', 'LB', 'PS', 'SB', 'SJ', 'VC',
bookieArray = [(x+'H', x+'A', x+'D') for x in bookieArray]
bookieArray = list(chain(*bookieArray))

bookiedf = df[bookieArray]

df = df[['Date','HomeTeam','AwayTeam','FTHG','FTAG','FTR','HS','AS','HST','AS
df = df.reset_index().drop('index',1)
df.head()
```

Out[3]:

	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HS	AS	HST	AST	HF	AF	HC	AC	HY
0	07/08/09	Wolfsburg	Stuttgart	2	0	H	13	14	7	4	12	12	6	3	0
1	08/08/09	Dortmund	FC Koln	1	0	H	24	7	11	0	8	10	16	1	0
2	08/08/09	Hertha	Hannover	1	0	H	10	15	4	3	16	20	5	3	3
3	08/08/09	Hoffenheim	Bayern Munich	1	1	D	9	9	1	3	10	28	3	10	0
4	08/08/09	Mainz	Leverkusen	2	2	D	8	13	4	7	22	28	3	5	1

```
In [4]: def histstats(hist, team):
            goals, shots, targets, fouls, corners, yellows, reds = 0,0,0,0,0,0,0
            results = ''
            for i, row in hist.iterrows():
                if row['HomeTeam']==team:
                    if row['FTR']=='H':
                        results += ',W'
                    elif row['FTR']=='A':
                        results += ',L'
                    elif row['FTR']=='D':
                        results += ',D'
                    goals += row['FTHG']
                    shots += row['HS']
                    targets += row['HST']
                    fouls += row['HF']
                    corners += row['HC']
                    yellows += row['HY']
                    reds += row['HR']
                elif row['AwayTeam']==team:
                    if row['FTR']=='H':
                        results += ',L'
                    elif row['FTR']=='A':
                        results += ',W'
                    elif row['FTR']=='D':
                        results += ',D'
                    goals += row['FTAG']
                    shots += row['AS']
                    targets += row['AST']
                    fouls += row['AF']
                    corners += row['AC']
                    yellows += row['AY']
                    reds += row['AR']
            return [results[1:], goals, shots, targets, fouls, corners, yellows, rec
```

In [5]:
```
newdf = pd.DataFrame()

for current, row in df.iterrows():
    h = df.iloc[current]['HomeTeam']
    a = df.iloc[current]['AwayTeam']
    date = df.iloc[current]['Date']
    ftr = df.iloc[current]['FTR']


    df2 = df.iloc[range(current-1,-1,-1)]
    homehist = df2[(df2['HomeTeam']==h) | (df2['AwayTeam']==h)].head(5)
    awayhist = df2[(df2['HomeTeam']==a) | (df2['AwayTeam']==a)].head(5)
    newrow = pd.DataFrame([[date,h,a,ftr] + histstats(homehist, h) + histsta
    newdf = newdf.append(newrow)

newdf = newdf.reset_index().drop('index',1)
newdf.head()
```

Out[5]:

	Date	HTeam	ATeam	FTR	HResults	HGoals	HShots	HTargets	HFouls	HCorners
0	07/08/09	Wolfsburg	Stuttgart	H		0	0	0	0	0
1	08/08/09	Dortmund	FC Koln	H		0	0	0	0	0
2	08/08/09	Hertha	Hannover	H		0	0	0	0	0
3	08/08/09	Hoffenheim	Bayern Munich	D		0	0	0	0	0
4	08/08/09	Mainz	Leverkusen	D		0	0	0	0	0

In [6]:
```
bookieArray = ['B365', 'BS', 'PSC', 'GB', 'IW', 'LB', 'PS', 'SB', 'SJ', 'VC'
for i in bookieArray:
    bookieSum = bookiedf[i+'H'] + bookiedf[i+'D'] + bookiedf[i+'A']
    bookiedf[i+'H'] = bookiedf[i+'H'] / bookieSum
    bookiedf[i+'D'] = bookiedf[i+'D'] / bookieSum
    bookiedf[i+'A'] = bookiedf[i+'A'] / bookieSum
bookiedf = bookiedf[bookiedf.columns[bookiedf.isnull().any() == False].tolis
bookiedf = bookiedf.reset_index().drop('index',1)
bookiedf.head()
```

Out[6]:

	B365H	B365A	B365D	LBH	LBA	LBD	VCH	VCA	VCD	W
0	0.211957	0.407609	0.380435	0.218579	0.409836	0.371585	0.222857	0.388571	0.388571	0.211
1	0.149034	0.505980	0.344986	0.176346	0.475185	0.348469	0.165839	0.496524	0.337637	0.163
2	0.183673	0.459184	0.357143	0.189485	0.438116	0.372399	0.195652	0.434783	0.369565	0.184
3	0.441640	0.200841	0.357518	0.423729	0.203390	0.372881	0.441989	0.204420	0.353591	0.423
4	0.413043	0.217391	0.369565	0.380952	0.238095	0.380952	0.404494	0.213483	0.382022	0.359

In [7]:

```
import numpy as np
cdf = pd.concat([newdf, bookiedf], axis=1)
cdf = cdf.iloc[45:]

df1 = cdf['HResults'].apply(lambda x: pd.Series(x.split(',')))
df1.columns = ['H1','H2','H3','H4','H5']
df2 = cdf['AResults'].apply(lambda x: pd.Series(x.split(',')))
df2.columns = ['A1','A2','A3','A4','A5']

cdf = pd.concat([cdf,df1,df2], axis=1)
cdf = cdf.drop('HResults',1)
cdf = cdf.drop('AResults',1)
cdf.head()
```

Out[7]:

	Date	HTeam	ATeam	FTR	HGoals	HShots	HTargets	HFouls	HCorners	HYellows
45	18/09/09	Schalke 04	Wolfsburg	A	7	51	15	114	35	8
46	19/09/09	Bayern Munich	Nurnberg	H	11	64	28	80	30	3
47	19/09/09	Bochum	Mainz	A	5	65	15	83	27	13
48	19/09/09	Hannover	Dortmund	D	3	69	22	92	29	5
49	19/09/09	M'gladbach	Hoffenheim	A	7	71	30	89	28	7

5 rows × 40 columns

In [8]:

```
formColumns = ['H'+str(x) for x in range(1,6)] + ['A'+str(x) for x in range(
newcdf = cdf.dropna(axis=0, how='any')
newcdf = pd.get_dummies(newcdf, columns= formColumns)
newcdf.head()
```

Out[8]:

	Date	HTeam	ATeam	FTR	HGoals	HShots	HTargets	HFouls	HCorners	HYellows
45	18/09/09	Schalke 04	Wolfsburg	A	7	51	15	114	35	8
46	19/09/09	Bayern Munich	Nurnberg	H	11	64	28	80	30	3
47	19/09/09	Bochum	Mainz	A	5	65	15	83	27	13
48	19/09/09	Hannover	Dortmund	D	3	69	22	92	29	5
49	19/09/09	M'gladbach	Hoffenheim	A	7	71	30	89	28	7

5 rows × 60 columns

In [9]:

```
df = newcdf.drop('Date',1).drop('HTeam',1).drop('ATeam',1)
indices = np.random.rand(len(df))<0.8
train_df = df[indices]
test_df = df[~indices]
```

In [59]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
train_df = train_df[train_df.FTR != 'D']
test_df = test_df[test_df.FTR != 'D']
trFts = train_df.drop('FTR',1)
trLbs = train_df['FTR']
tsFts = test_df.drop('FTR',1)
tsLbs = test_df['FTR']

logit = LogisticRegression()
logit = logit.fit(trFts,trLbs)
svm1 = SVC()
svm1 = svm1.fit(trFts,trLbs)
#print(svm.score(trFts,trLbs))
rfc = RandomForestClassifier()
rfc = rfc.fit(trFts,trLbs)
#parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] }
clf = GridSearchCV(logit, param_grid)
clf.fit(trFts,trLbs)

print('Test_Score_SVM:',svm1.score(tsFts,tsLbs))
print('Test_Score_LR:',logit.score(tsFts,tsLbs))
print('Test_Score_RFC:',rfc.score(tsFts,tsLbs))
print('Test_Score_GRIDSEARCHCV_LR:',clf.score(tsFts,tsLbs))

#print(logit.score(trFts,trLbs))
#print(rfc.score(trFts,trLbs))
#print(clf.score(trFts,trLbs))
pd.DataFrame(index=tsFts.columns.values, data=rfc.feature_importances_, colu
```

```
Test_Score_SVM: 0.604294478528
Test_Score_LR: 0.705521472393
Test_Score_RFC: 0.656441717791
Test_Score_GRIDSEARCHCV_LR: 0.708588957055
```

Out[59]:

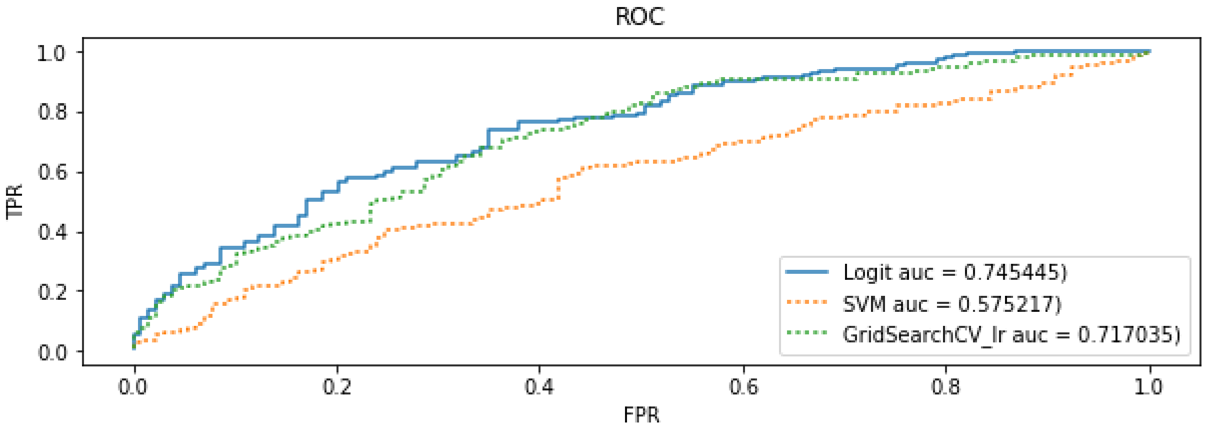
	importance
VCH	0.061832
LBH	0.053714
WHA	0.050593
B365H	0.047892
WHH	0.046648
HShots	0.040289
VCD	0.039916
HCorners	0.038927

	importance
AFouls	0.036393
LBD	0.035508
LBA	0.034137
ATargets	0.032224
AShots	0.032130
ACorners	0.032023
HFouls	0.031743
B365A	0.031422
B365D	0.029556
WHD	0.028274
HTargets	0.024846
AYellows	0.023693
VCA	0.023466
HYellows	0.020297
HGoals	0.019540
AGoals	0.018432
HReds	0.014292
H4_D	0.008376
A1_L	0.008322
A4_W	0.007212
A5_L	0.007142
A2_W	0.006566
AReds	0.006257
A4_L	0.006071
A3_W	0.006028
H2_D	0.005895
A3_L	0.005874
H3_L	0.005818
H3_D	0.005288
H5_L	0.005057
H1_D	0.004784
H1_L	0.004660
H4_L	0.004544
H5_D	0.004534
A4_D	0.004358

	importance
A3_D	0.004248
H1_W	0.004186
H2_W	0.004141
H4_W	0.004058
H3_W	0.004023
H5_W	0.003854
A1_D	0.003429
H2_L	0.003255
A2_L	0.003214
A5_W	0.003030
A2_D	0.002797
A5_D	0.002693
A1_W	0.002503

In [54]:

```

score = np.array(logit.decision_function(tsFts))
score2 = np.array(svm1.decision_function(tsFts))
#score3 = np.array(rfc.decision_function(tsFts))[:,0]
score4 = np.array(clf.decision_function(tsFts))
#y = np.array(tsLbs)
truth = pd.get_dummies(tsLbs)
truth = truth['H']
fpr,tpr,_ = roc_curve(truth, score)
fpr2,tpr2,_ = roc_curve(truth, score2)
#fpr3,tpr3,_ = roc_curve(truth, score3)
fpr4,tpr4,_ = roc_curve(truth, score4)

roc_auc = auc(fpr, tpr)
roc_auc2 = auc(fpr2, tpr2)
#roc_auc3 = auc(fpr3, tpr3)
roc_auc4 = auc(fpr4, tpr4)

plt.figure(figsize=(10, 3))
plt.plot(fpr, tpr, label='Logit auc = %f)' % roc_auc)
plt.plot(fpr2, tpr2, ':', label='SVM auc = %f)' % roc_auc2)
#plt.plot(fpr3, tpr3, ':', label='Random Forest auc = %f)' % roc_auc3)
plt.plot(fpr4, tpr4, ':', label='GridSearchCV_lr auc = %f)' % roc_auc4)

plt.legend(loc="lower right")
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC')
plt.show()
"""
"""
```



Out[54]: '\n'

In [57]:

```
from sklearn.cross_validation import *
from sklearn.metrics import roc_auc_score
from sklearn import svm
def xValSVM(dataset, label_name, k, cs):
    '''
    Perform k-fold cross validation on SVM across a range of C,
    returns mean and var of auc avg.
    '''
    n_samp = dataset.shape[0]
    cv = KFold(n = n_samp, n_folds = k)
    aucs = {}

    for train_index, test_index in cv:
        tr_k = dataset.iloc[train_index]
        va_k = dataset.iloc[test_index]

        for c in cs:
            svm_k = svm.SVC(kernel = 'linear', C = c)
            svm_k.fit(tr_k.drop(label_name, 1), tr_k[label_name])
            met = roc_auc_score(va_k[label_name], svm_k.decision_function(va

            if (c in aucs):
                aucs[c].append(met)
            else:
                aucs[c] = [met]

    return aucs
```

In [64]:

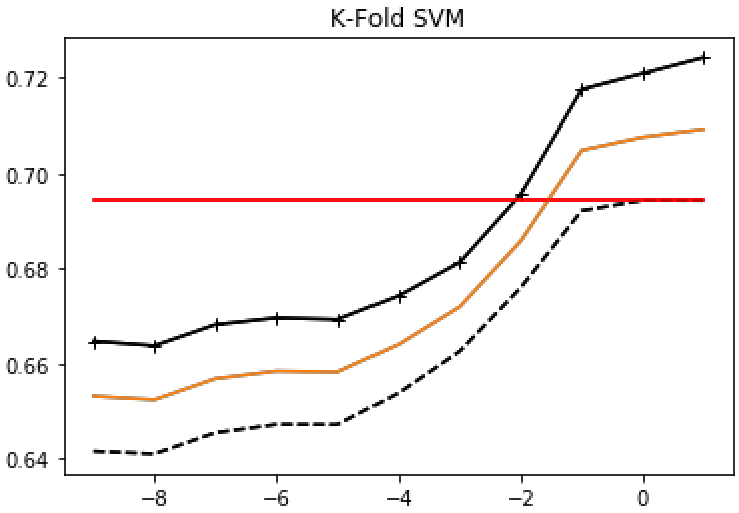
```
xval_dict = {'e':[], 'mu':[], 'sig':[]}
k = 10
er = range(-9, 2)
cs = [10**i for i in er ]

aucs_sv = xValSVM(df, 'FTR', k, cs)

for i in er:
    xval_dict['e'].append(i)
    xval_dict['mu'].append(np.array(aucs_sv[10**i]).mean())
    xval_dict['sig'].append(np.sqrt(np.array(aucs_sv[10**i]).var()/10))

means = np.array(xval_dict['mu'])
cs = np.array(xval_dict['e'])
stderr = np.array(xval_dict['sig'])

low = (means-stderr).max()
best_c_ser = np.array(cs)[(means>low)].min()
best_c_max = np.array(cs)[(means==max(means))]

plt.plot(xval_dict['e'], means)
plt.plot(xval_dict['e'], means+stderr, 'k+-')
plt.plot(xval_dict['e'], means-stderr, 'k--')
plt.plot(xval_dict['e'], low*np.ones(len(means)), 'r')
```

Out[64]:

```
[<matplotlib.lines.Line2D at 0x10217f780>]
```

```
In [65]: plt.title('K-Fold SVM')
         plt.legend()
         plt.show()
```

```
/Users/monilshah/anaconda3/lib/python3.5/site-packages/matplotlib/axes/_a
xes.py:545: UserWarning: No labelled objects found. Use label='...' kwarg
on individual plots.
  warnings.warn("No labelled objects found. "
```



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```