Investment and Trading:
Stock Predictor

Definition

Project Overview

The stock market is a collection of buyers and sellers of fractions of company ownership. Buyers and sellers can take a variety of forms from individuals to investment firms to funds. Their goal is to understand how to make more profitable decisions with their investments. Copious data is available in the form of historical prices. These prices will be used to generate training data for machine learning algorithms.

The endeavor of this project will be to provide return predictions for investments over a user defined period from the present day.

Data used in this project is retrieved from the Yahoo Finance API provided through the `pandas` library. Historical data including the information of concern, adjusted closing prices, are retrieved using a company symbol, an all uppercase group of characters representing how the company is referred to in the market ("AAPL" for Apple, "ABC" for Google, "IBM" for IBM). The historical adjusted closing prices are requested for all trading days in the past year up to the day the program is run as specified in the project `README.md` file.

Problem Statement

Accurate predictions of future stock returns provide the opportunity to make profitable and responsible investments. Stock price movements, increases and decreases in stock value, are known to be difficult to predict. Stock prices are subject to a lot of context created by participants in the stock market. While saturated with noise, there is still information that can be exploited to make more responsible market strategies.

Metrics

The metrics used to measure the performance of the models will be Root Mean Squared Error (RMSE), Mean Average Percentage Error (MAPE), and Correlation.

RMSE makes sense to use in most contexts because it gives a scale of the regression error in the same units as the prediction itself. MAPE, on the other hand, makes sense to use when the outcome in question is the future stock price and not the return. MAPE gives an estimate of the error as a percentage of the overall price of a stock. Since the outcome this project will focus on is investment return (future price / present price - 1) which is normally on a scale from -1 to +1, this project will display RMSE for aesthetic purposes. In other words, RMSE displays a less intimidating view of the error.

Correlation is also displayed to show how well the predictions trend with the true value of the returns. The benchmark for this is set in the analysis section by analyzing how well the current price correlates with future results.

Analysis

Data Exploration

The complex nature of presenting this data is that trends in the stock market are constantly changing. In this project, a variety of Company symbols will be used to display how the analysis was conducted. The figure below is an example of how random the market returns seem.

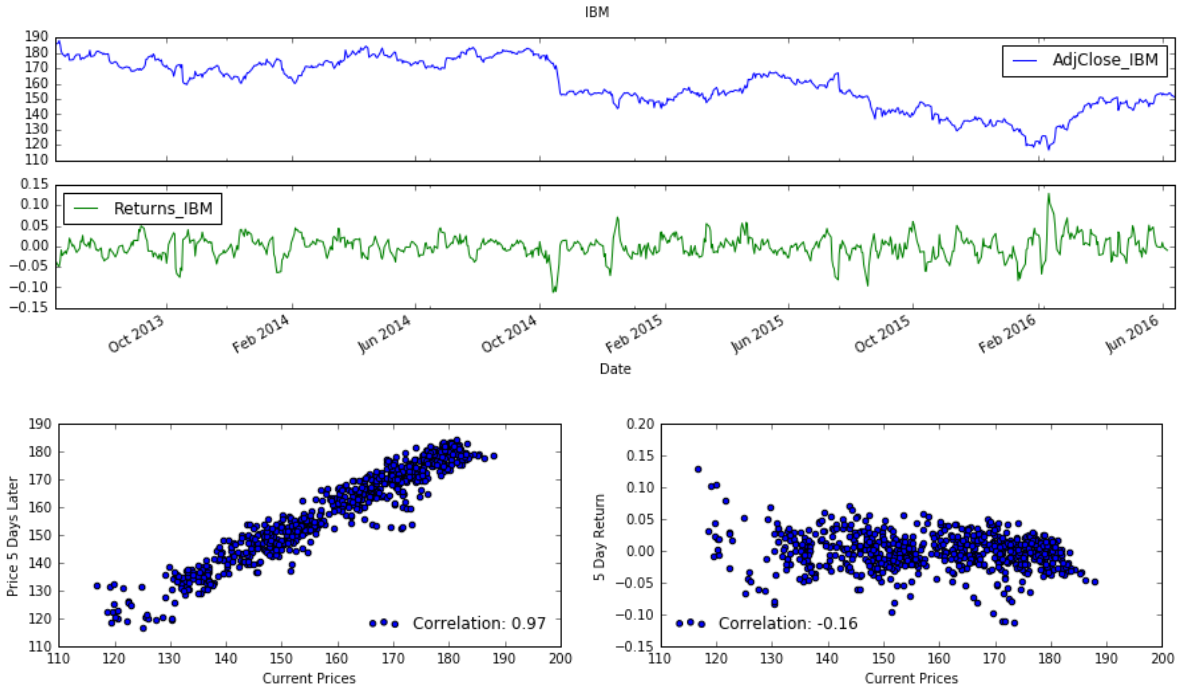
In [1]:
```
# Import Python Libraries necessary for the report
import datetime as dt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
# Import personal libraries developed for project available on github for exploration:
# https://github.com/Seananigans/Finance/tree/master/StockPredictor
from dataset_construction import create_input, create_output, get_and_store_web_data
from error_metrics import rmse
from learners.LinRegLearner import LinRegLearner as lrl
from learners.KNNLearner import KNNLearner as knn
from normalization import mean_normalization
from predict_future import predict_spy_future
```

In [2]:
```
# Create features dataset and output dataset
ibm = create_input("IBM")
ibm_future_prices = create_output("IBM", use_prices=True)
ibm_future_returns = create_output("IBM", use_prices=False)
ibm_future_prices.columns = ["Future_IBM"]
ibm_future_returns.columns = ["Returns_IBM"]
# Display the joined Dataset
ibm_price_returns_df = ibm.join(ibm_future_prices).join(ibm_future_
returns)
num_days = ibm_price_returns_df.shape[0]
print ibm_price_returns_df.iloc[:4]
print
# Display Statistics about output variable
print "There are {} days/observations in the data.".format(ibm_pric
e_returns_df.shape[0])
print "{} or {:.2f}% of those days observe a 5 day future return AB
OVE zero.".format(
    ibm_price_returns_df[ibm_price_returns_df.Returns_IBM>0.0].shap
e[0],
    ibm_price_returns_df[ibm_price_returns_df.Returns_IBM>0.0].shap
e[0]/float(num_days)*100.)
print "{} or {:.2f}% of those days observe a 5 day future return EQ
UAL TO or BELOW zero.".format(
    ibm_price_returns_df[ibm_price_returns_df.Returns_IBM<=0.0].sha
pe[0],
    ibm_price_returns_df[ibm_price_returns_df.Returns_IBM<=0.0].sha
pe[0]/float(num_days)*100.)
print "{} of those days observe a 5 day future return that is not a
number (eg. NaN).".format(
    ibm_price_returns_df[np.isnan(ibm_price_returns_df.Returns_IB
M)].shape[0])
print "The average 5 day future return for all days in this \
dataset is {:.4f} with a standard deviation of {:.4f}.".format(
    ibm_price_returns_df.Returns_IBM.mean(), ibm_price_returns_df.R
eturns_IBM.std()
)
```

```
            AdjClose_IBM  Future_IBM  Returns_IBM
Date
2013-06-14    185.494397  179.311259    -0.033333
2013-06-17    186.264993  177.549876    -0.046789
2013-06-18    187.943802  178.870906    -0.048275
2013-06-19    185.255883  178.760825    -0.035060

There are 755 days/observations in the data.
375 or 49.67% of those days observe a 5 day future return ABOVE ze
ro.
375 or 49.67% of those days observe a 5 day future return EQUAL TO
or BELOW zero.
5 of those days observe a 5 day future return that is not a number
(eg. NaN).
The average 5 day future return for all days in this dataset is -
0.0009 with a standard deviation of 0.0270.
```

Exploratory Visualization

In [3]:
```
# Display Prices and returns over whole timeline for example datase
t IBM
ibm.join(ibm_future_returns).plot(subplots=True,figsize=(15,4), tit
le="IBM")
# Display how current price relates to 5 day future returns and pri
ces
prices_corr = ibm.join(ibm_future_prices).corr()["AdjClose_IBM"]["F
uture_IBM"]
returns_corr = ibm.join(ibm_future_returns).corr()["AdjClose_IBM"]
["Returns_IBM"]
f, axarr = plt.subplots(1,2, figsize=(15,3))
axarr[0].scatter(ibm, ibm_future_prices, label="Correlation: {}".fo
rmat(round(prices_corr,2)))
axarr[0].set_xlabel('Current Prices')
axarr[0].set_ylabel('Price 5 Days Later')
axarr[0].legend(loc="lower right", frameon=False)
axarr[1].scatter(ibm, ibm_future_returns, label="Correlation: {}".f
ormat(round(returns_corr,2)))
axarr[1].set_xlabel('Current Prices')
axarr[1].set_ylabel('5 Day Return')
axarr[1].legend(loc="lower left", frameon=False)

plt.show()
```



The high correlation between the current and future price is easy to explain for a short prediction period. When prices are low, prices 5 days later are likely to remain low by comparison. It is unlikely for prices to make many huge jumps up or down in price. On the other hand, there is a moderate negative correlation between current price and 5 day returns. This implies that when prices are low, a positive return would be expected, while when prices are high, a negative return is somewhate more likely.

While model performance will be explored for predicting future price, the main concern of any investor is the future return. For that reason, the overall goal of the project is to out-perform the benchmark for predicting returns.

Algorithms and Techniques

A variety of indicators are explored to analyze the movement of prices in the market. This section will discuss some of the indicators used and how they were tested to determine which would be most effective.

The list of indicators tested were:

- Window based Indicators:
  - N-Day Bollinger Value
    - *(Price[t] - N-day Moving Average) / (2 * N-day Standard Deviation of the Price)*
  - Exponential Moving Average (EMA)
    - *(Price[t] - EMA[t-1]) * (2/(N+1)) + EMA[t-1]*
  - N-day Simple Moving Average (SMA)
    - Average(*Price[t-N] to Price[t]*)
  - N-day Momentum
    - *Price[t] / Price[t-N] - 1*
  - Relative Strength Index (RSI)
    - RSI = *100 - 100/(1+RS)*
    - RS = *N-day Average Gain / N-day Average Loss*
  - Volatility
    - N-day Standard Deviation of the Price
- One non-window based indicator:
  - Weekdays
    - Is it Monday? [0 or 1]
    - Is it Tuesday? [0 or 1],
    - etc.

Every window indicator was analyzed for varying window size from a one-day window to a thirty-day window.

For every member of the S&P 500 list found in `spylist.csv`:

```
1) A dataset was created with the specified indicator and specified win
dow.
2) The dataset was split into training and testing data.
3) A linear regression model was trained using training data.
4) The linear regression model predictions were compared the test data.
5) Test error for the indicator was added to a running total over the S
&P 500 list.
6) Test errors were sorted to find the best performing indicator windo
w.
```

The top three performing windows for each indicator were added to another list. All combinations up to a size of 4 indicators were then tested in the same way against the above S&P500 to determine the best set of indicators.

As for the models, this is a supervised machine learning regression task. Therefore, it will require the use of regression models. Given the endeavor is to predict returns for approximately 500 stocks, the regression models will be limited to less complex models that can be trained on smaller amounts of data in a more limited time period. This rules out the use of Neural Networks, as they require lots of data to train them and also have a large number of hyperparameters to adjust. In summation, the models used in this project will be limited to linear regression and k-Nearest Neighbors. These models are quick to train and quick to query on the amount of data they will have to work.

Benchmark

There are a few benchmarks that this project will endeavor to beat. The first is to achieve a better test root mean squared error (RMSE) than predicting zero. This benchmark is essentially the equivalent of the standard deviation of the returns considering the average is essentially zero. Returns are well known to average out to 0 over short periods. Short term returns are known to be very noisy. Aside from that discovered in research, the data exploration section showed roughly 50% of the 5 day returns to be above and below zero.

Another benchmark is to achieve a better correlation than that of the adjusted close to the 5 day returns. A correlation cannot be formed from a comparison between all zeros and the actual returns. Further, always predicting 0.0 future returns never allows for any trading opportunities. This is why the Adjusted close correlation will be compared against the returns.

Methodology

Data Preprocessing

With regard to abnormalities in the data, there were not many. Occasionally, there will be gaps in Adjusted Closing data from one date to another. It is common practice to simply feed the last observed price forward to a date where there is data. That is one of the many functionalities of the Pandas library in python, specifically `DataFrame.bfill()` and `DataFrame.ffill()`.

Other than the `NaN` values discussed above, there were some that were produced by the indicators developed in this project along with in the construction of the output data. Some indicators were constructed from a defined number of trading days into the past. This meant that for some days not all the information was available to create the indicator and `NaN` values were produced. These rows were simply eliminated using `DataFrame.dropna()`.

In [4]:
```
# Creates input data with a 4 day window bollinger value indicator
from indicators.Bollinger import Bollinger
df = create_input("IBM", indicators=[Bollinger(3)])
# Display how feature data appears before removing NA values.
print df.iloc[:5]
```

```
            AdjClose_IBM  Bollinger_3_AdjClose_IBM
Date
2013-06-14    185.494397                       NaN
2013-06-17    186.264993                       NaN
2013-06-18    187.943802                       NaN
2013-06-19    185.255883                 -0.565978
2013-06-20    181.045108                 -0.407173
```

In [5]:
```
# Display how feature data appears after removing NA values.
print df.iloc[:6].dropna()
# Display mean normalized feature data
print "====================After Normalizing===================="
df = (df-df.mean())/ df.std()
print df.iloc[:6].dropna()
```

```
            AdjClose_IBM  Bollinger_3_AdjClose_IBM
Date
2013-06-19    185.255883                 -0.565978
2013-06-20    181.045108                 -0.407173
2013-06-21    179.311259                  0.447208
====================After Normalizing====================
            AdjClose_IBM  Bollinger_3_AdjClose_IBM
Date
2013-06-19      1.614492                 -1.354219
2013-06-20      1.351512                 -0.966460
2013-06-21      1.243226                  1.119701
```

The above data shows features that are on largely different scales. To some machine learning algorithms, the size of different each feature will matter, while output data does not need to be normalized. Subsequently, the data was normalized using mean normalization.

In [ ]:
```
```

In [6]:
```
# Creates output data of 3 day future returns
df_output = create_output("IBM",horizon=3)
# Display output data as it appears before removing NA values.
print df_output.iloc[-6:]
# Display output data as it appears before removing NA values.
print "=========After Processing==========="
print df_output.iloc[-6:].dropna()
```

```
                y_IBM
Date
2016-06-06  0.004518
2016-06-07 -0.006261
2016-06-08 -0.017662
2016-06-09       NaN
2016-06-10       NaN
2016-06-13       NaN
=========After Processing==========
                y_IBM
Date
2016-06-06  0.004518
2016-06-07 -0.006261
2016-06-08 -0.017662
```

Implementation

Separating data into training and test sets was completed using the earlier data to train and the later data to test. Training and test data cannot be randomly selected because it would incur a *look-ahead bias*, which simply means that the model would have access to information that it normally would not have access. For this reason, data was split using the first 80% of the data for training, and the last 20% for testing.

Occasionally, models had numerical problems providing a linear model to particular companies' historical prices. As any `NaN` values are removed prior to building a model, the companies were removed from the dataset to avoid complications in testing. No clear solution was discovered for addressing the problem in the imported libraries.

After running exhaustive tests with `testindicator.py`, indicators did not appear to provide clear benefit to a Linear Model's predictive capabilities. The table below shows some, but not all, of the indicator's correlation with the returns. They have very low and mostly negative correlation with the output.

Linear Regression also has difficulties with multicolinearity, essentially if two features are highly collinear there will be some confusion about how much weight to apply to either. For this reason, Open, High, Low, and Close will also not be used directly. Instead, the differences between the *high* and the *low*, and *open* and *close*, are known to provide extra information about trader confidence in the price of a stock. The difference also has a vastly more limited range than using the prices themselves. This works out well because the output also has a limited range.

In [7]:
```
from indicators import Bollinger, SimpleMA, ExponentialMA, RSI, Vol
atility, Weekdays, Lag
ibm_indicators = create_input("IBM",[
        Bollinger.Bollinger(i) for i in range(3,20)] + [
        SimpleMA.SimpleMA(i) for i in range(3,20)] + [
        RSI.RSI(i) for i in range(3,20)] + [
        ExponentialMA.ExponentialMA(i) for i in range(3,20)] + [
        Weekdays.Weekdays() ] + [
        Lag.Lag(i) for i in range(5)])
overall_data = get_and_store_web_data("IBM", online=False)
ibm_indicators = ibm_indicators.join( overall_data[
        [col for col in overall_data.columns if not col.startswith
("Adj")]] )
print (ibm_indicators.join(ibm_future_returns).corr().ix[:,-1])[np.
abs(
        ibm_indicators.join(ibm_future_returns).corr().ix[:,-1])>0.
08]
```

```
AdjClose_IBM          -0.162257
SMA_15_AdjClose_IBM    0.125734
EMA_4_AdjClose_IBM     0.146601
Open_IBM              -0.144284
High_IBM              -0.144782
Low_IBM               -0.147613
Close_IBM             -0.149263
Returns_IBM            1.000000
Name: Returns_IBM, dtype: float64
```

Refinement

Predicting Future Returns with Linear Regression

The code below shows the results of using linear regression to predict returns five days into the future for all Data in the S&P 500 list held in `spy_list.csv`. This is the function that can be used in the command line as `python predict_future.py` after populating the `webdata` folder. It will make predictions for a specified number of trading days in the future and present the 10 predictions with the lowest test RMSE. All of the results can be viewed in the `return_results.csv` file in the same folder.

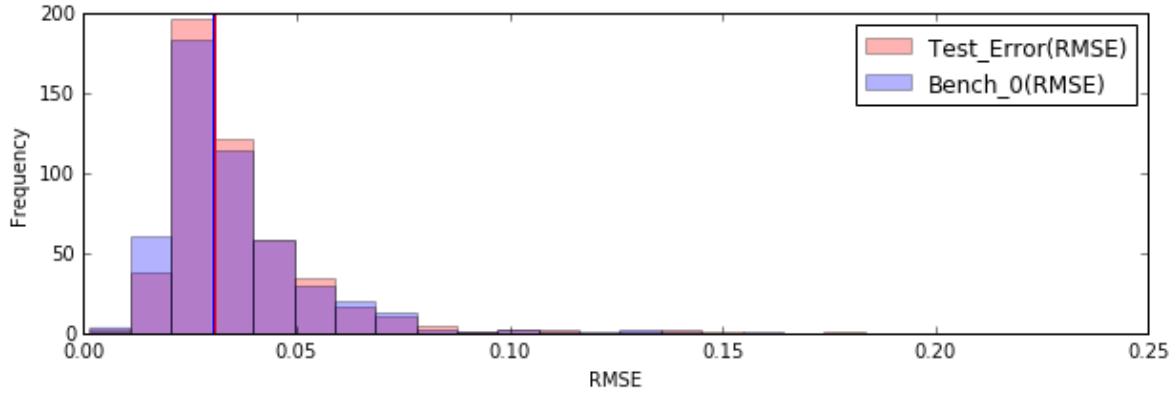
In [8]:
```
# Reports 5 symbols with the lowest test prediction error (RMSE)
predictions = predict_spy_future(horizon=5, learner=lrl)
print predictions[[col for col in predictions.columns if not col.st
artswith("Return_Date")]].iloc[:5]
```

```
           Symbol  Return(%)  Test_Error(RMSE)  Bench_0(RMSE)  Tes
t_Corr
Date
2016-06-13     TE   0.002059          0.003465       0.002900  -0.
069360
2016-06-13    GAS   0.001575          0.007862       0.004787   0.
173005
2016-06-13    CVC   0.006690          0.013348       0.011674   0.
239181
2016-06-13    MMM  -0.012610          0.014789       0.012345   0.
267725
2016-06-13     PG   0.004258          0.015582       0.016028   0.
282982

/Users/seanhegarty/Library/Enthought/Canopy_64bit/User/lib/python
2.7/site-packages/scipy/linalg/basic.py:884: RuntimeWarning: inter
nal gelsd driver lwork query error, required iwork dimension not r
eturned. This is likely the result of LAPACK bug 0038, fixed in LA
PACK 3.2.2 (released July 21, 2010). Falling back to 'gelss' drive
r.
  warnings.warn(mesg, RuntimeWarning)
```

Below, statistics for the data above is calculated and presented in tables. There is also a histogram to show how well predicted RMSE matches the benchmark RMSE.

In [9]:
```
# Display performance statistics as they relate to the benchmark fo
r all S&P 500 companies in spy_list.csv
predicted_returns_results = pd.read_csv("return_results.csv", index
_col="Date")
# Calculate meaningful statistics
averages = predicted_returns_results[["Test_Error(RMSE)","Bench_0(R
MSE)","Test_Corr"]].mean(
    axis=0).to_frame(name="Average")
medians = predicted_returns_results[["Test_Error(RMSE)","Bench_0(RM
SE)","Test_Corr"]].median(
    axis=0).to_frame(name="Median")
stdevs = predicted_returns_results[["Test_Error(RMSE)","Bench_0(RMS
E)","Test_Corr"]].std(
    axis=0).to_frame(name="Standard Deviation")
df = averages.join(medians).join(stdevs)
print df
# Plot histogram to show performance against benchmark.
predicted_returns_results[["Test_Error(RMSE)","Bench_0(RMSE)"]].plo
t.hist(color=["r","b"], bins=30,

alpha=0.3, figsize=(10,3))
plt.axvline(predicted_returns_results.median(axis=0)[["Test_Error(R
MSE)"]].values, color="r")
plt.axvline(predicted_returns_results.median(axis=0)[["Bench_0(RMS
E)"]].values, color="b")
plt.xlabel("RMSE")
plt.xlim((0,.25))
plt.show()
```

	Average	Median	Standard Deviation
Test_Error(RMSE)	0.036466	0.030800	0.022261
Bench_0(RMSE)	0.035533	0.030249	0.021356
Test_Corr	0.211033	0.221939	0.200044



After failing to discover distinctly effective indicators, focus shifted. Using information provided from the initial grab from Yahoo Finance, like Volume, Open, Close, High, and Low, a lower RMSE was realized. Extra effort was also put toward training models on few features and optimizing the predictions. Since different model types have different strengths, the below section explores averaging the outputs of the models as an ensemble prediction.

Ensemble: Predicting Future Returns with Linear and k-Nearest Neighbors Regression

In [10]:
```
# Create full dataset
dataset = get_and_store_web_data("IBM", online=False).join(ibm_futu
re_returns).dropna()
# Calculate Open minus Close and High minus Low
dataset["HmL_IBM"] = dataset["High_IBM"]-dataset["Low_IBM"]
dataset["OmC_IBM"] = dataset["Open_IBM"]-dataset["Close_IBM"]
# Choose features that will be used for training model
dataset = create_input("IBM", []).join(
    dataset[["Volume_IBM","HmL_IBM"]]).join(
    create_output("IBM", use_prices=False)).dropna()
# Change actual Adjusted Close and Volume to percent change to deal
with trends in prices and
# improve stationarity (decrease time dependence)
dataset[["AdjClose_IBM","Volume_IBM"]] = dataset[["AdjClose_IBM","V
olume_IBM"]].pct_change()
# Remove any NaN values
dataset = dataset.dropna()
# Create testing set
num_rows = dataset.shape[0]
test_rows = range(num_rows-int(0.2*num_rows), num_rows)
testingX, testingY = (dataset.ix[test_rows,:-1], dataset.ix[test_ro
ws,-1])
# Remove test rows
features = dataset.iloc[range(0, num_rows-int(0.2*num_rows))]
# Create datasets to use for cross-validation
dividend = 3
n_rows = features.shape[0]
section = int(float(n_rows)/dividend + 1)
print "There are {} datasets with {} rows each, from the total {} r
ows in the dataset.".format(
    dividend, section, n_rows)
d_rows = [(section*(i-1), section*i) for i in range(1,dividend+1)]
print "Feature dataset indices: {}".format(d_rows)
features = [features.iloc[i[0]:i[1]] for i in d_rows]
print "Test dataset indices: {} to {}".format(test_rows[0],test_row
s[-1])
```

```
There are 3 datasets with 201 rows each, from the total 600 rows i
n the dataset.
Feature dataset indices: [(0, 201), (201, 402), (402, 603)]
Test dataset indices: 600 to 748
```

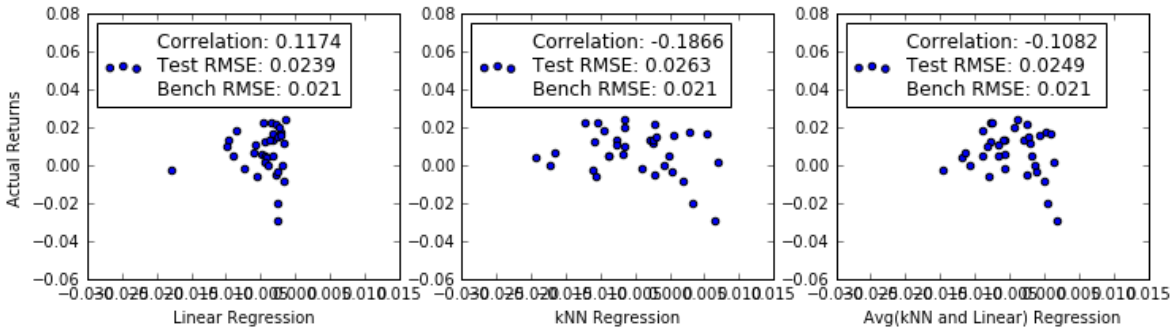
In [11]:

```
for dataset in features:
    # Create a training and validation set
    train_rows = range(0, int(0.8*dataset.shape[0]))
    valid_rows = range(int(0.8*dataset.shape[0]), dataset.shape[0])
    print "Trained from {} to {}.".format(dataset.iloc[train_rows].
index[0].strftime("%B %d, %Y"),
                                          dataset.iloc[train_rows].i
ndex[-1].strftime("%B %d, %Y"))
    print "Tested from {} to {}.".format(dataset.iloc[valid_rows].i
ndex[0].strftime("%B %d, %Y"),
                                          dataset.iloc[valid_rows].i
ndex[-1].strftime("%B %d, %Y"))
    # Split into training and validation sets.
    trainX, trainY = (dataset.ix[train_rows,:-1].values, dataset.ix
[train_rows,-1].values)
    validX, validY = (dataset.ix[valid_rows,:-1].values, dataset.ix
[valid_rows,-1].values)
    # Normalize training and validation set by the values of the tr
aining data
    trainX, validX = mean_normalization(trainX, validX)
    # Create two learners (Linear Regression and k-Nearest Neighbor
s Regression)
    learner = lrl()
    learner2 = knn(k=15)
    # Train the models on the training data
    learner.addEvidence(trainX, trainY)
    learner2.addEvidence(trainX, trainY)
    # Predict the future data in the validation set
    resultslrl = learner.query(validX)
    resultsknn = learner2.query(validX)
    # Average prediction for an ensemble prediction
    resultsavg = np.add(resultslrl , resultsknn)/2
    # Draw plots to compare predictions vs actual values
    f, (plot1, plot2, plot3) = plt.subplots(1,3, figsize=(12,3), sh
arex=True)
    # Linear Regression Prediction plot
    plot1.scatter(x = resultslrl, y = validY,
                  label="Correlation: {}\nTest RMSE: {}\nBench RMS
E: {}".format(
            round(np.corrcoef(resultslrl, validY)[0,1],4),
            round(rmse(resultslrl, validY),4),
            round(rmse(np.zeros(validY.shape), validY),4)))
    plot1.legend(loc="upper left")
    plot1.set_xlabel("Linear Regression")
    plot1.set_ylabel("Actual Returns")
    # k-Nearest Neighbors Regression Prediction plot
    plot2.scatter(x = resultsknn, y = validY,
                  label="Correlation: {}\nTest RMSE: {}\nBench RMS
E: {}".format(
            round(np.corrcoef(resultsknn, validY)[0,1],4),
            round(rmse(resultsknn, validY),4),
            round(rmse(np.zeros(validY.shape), validY),4)))
    plot2.legend(loc="upper left")
    plot2.set_xlabel( "kNN Regression" )
```

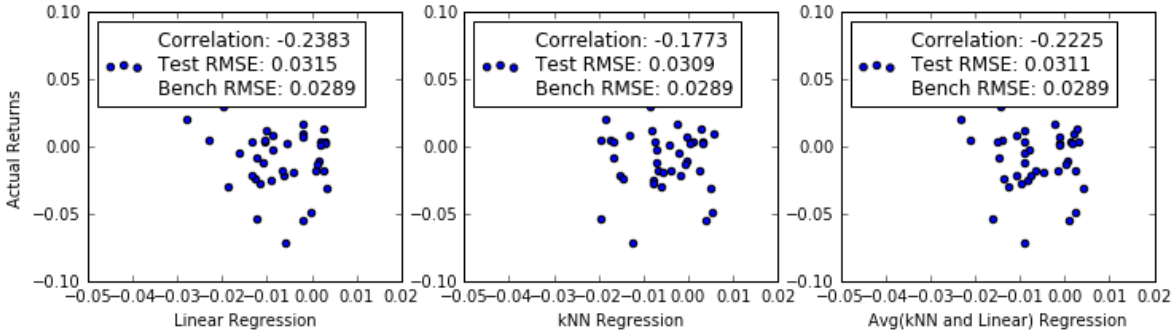
```
    # Avg(kNN and Linear) Regression Prediction plot
    plot3.scatter(x = resultsavg, y = validY,
                  label="Correlation: {}\nTest RMSE: {}\nBench RMS
E: {}".format(
            round(np.corrcoef(resultsavg, validY)[0,1],4),
            round(rmse(resultsavg, validY),4),
            round(rmse(np.zeros(validY.shape), validY),4)))
    plot3.legend(loc="upper left")
    plot3.set_xlabel( "Avg(kNN and Linear) Regression" )
    plt.show()
# Retreive data comparable in rows to those previously trained on.
trainingX, trainingY = (dataset.ix[-section:,:-1], dataset.ix[-sect
ion:,-1])
validX, validY = trainingX.values, trainingY.values
# Mean Normalize by data used to train the model
validX, testX = mean_normalization( validX, testingX )
# Print dates used for training and testing
print "Trained from {} to {}".format(dataset.iloc[-section:].index
[0].strftime("%B %d, %Y"),
                                     dataset.iloc[-section:].index
[-1].strftime("%B %d, %Y"))
print "Tested from {} to {}".format(testX.index[0].strftime("%B %d,
%Y"),
                                    testX.index[-1].strftime("%B %
d, %Y"))
# Create, train, and query Linear Regression Learner
learner = lrl()
learner.addEvidence(validX, validY)
resultslrl = learner.query(testX.values)
# Create, train, and query Linear Regression Learner
learner = knn()
learner.addEvidence(validX, validY)
resultsknn = learner.query(testX.values)
# Calculate Avg(kNN and Linear) Regression Predictions
resultsavg = np.add(resultslrl , resultsknn)/2
# Plot correlation between predicted and actual returns
plt.figure(figsize=(12,4))
plt.scatter(resultsavg,
            testingY,
            color="r",
            label="Correlation: {}\nTest RMSE: {}\nBench RMSE: {}".
format(
        round(np.corrcoef(resultsavg, testingY)[0,1],4),
        round(rmse(resultsavg, testingY),4),
        round(rmse(np.zeros(testingY.shape), testingY),4)))
plt.legend(loc="upper left")
plt.xlabel( "Avg(kNN and Linear) Regression Test Results" )
plt.ylabel("Actual Returns")
plt.axvline(0.0)
plt.axhline(0.0)
plt.show()
# Join returns data into one dataframe for plotting.
pred = pd.DataFrame(resultsavg, columns = ["Predicted"], index=test
ingY.index)
```

```
validY = pd.DataFrame(validY, columns = ["Past"], index=trainingY.i
ndex)
dframe = pd.DataFrame(pred.join(trainingY.to_frame(name="Past").joi
n(testingY, how="outer"), how="outer"))
# Calculate sensitivity.
print "Only {:.2f}% of the predicted returns were greater than 0.0,
\
while {:.2f}% of the actual test data returns were greater than 0.
0.".format(
    (np.sum(pred>0)*100./testingY.shape[0]).values[0], np.sum(testi
ngY>0)*100./testingY.shape[0] )
print "{:.2f}% of the time the return was positive, the model predi
cted it would be positive.".format(
    (dframe[dframe.Predicted>0])[dframe.y_IBM>0].shape[0]*100./dfra
me[dframe.y_IBM>0].shape[0] )
print "{:.2f}% of the time the return was negative, the model predi
cted it would be negative.".format(
    (dframe[dframe.Predicted<=0])[dframe.y_IBM<=0].shape[0]*100./df
rame[dframe.y_IBM<=0].shape[0] )
# Plot returns dataframe.
dframe.plot(figsize=(12,4))
plt.ylabel("Returns")
plt.axhline(0.0, color='k')
plt.show()
```

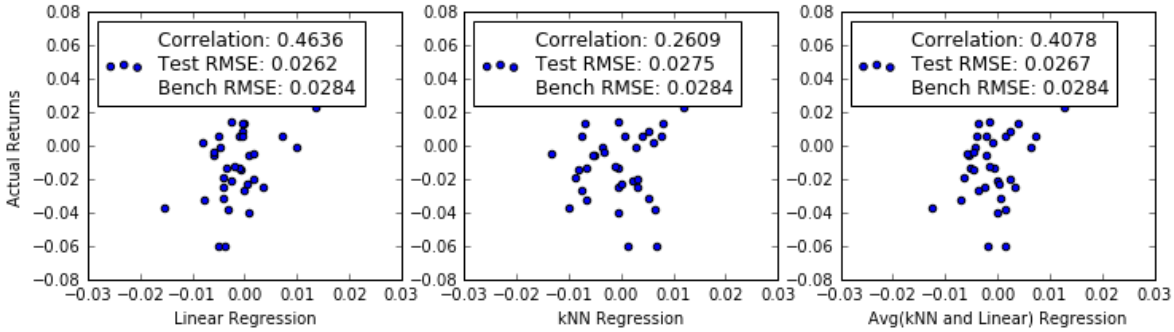
Trained from June 17, 2013 to February 03, 2014.
Tested from February 04, 2014 to April 02, 2014.



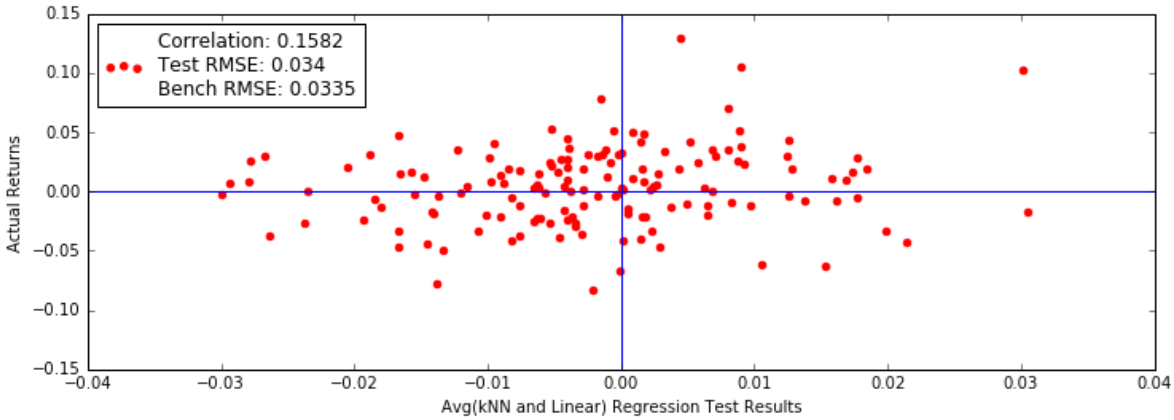
Trained from April 03, 2014 to November 18, 2014.
Tested from November 19, 2014 to January 20, 2015.



Trained from January 21, 2015 to September 03, 2015.
Tested from September 04, 2015 to October 30, 2015.

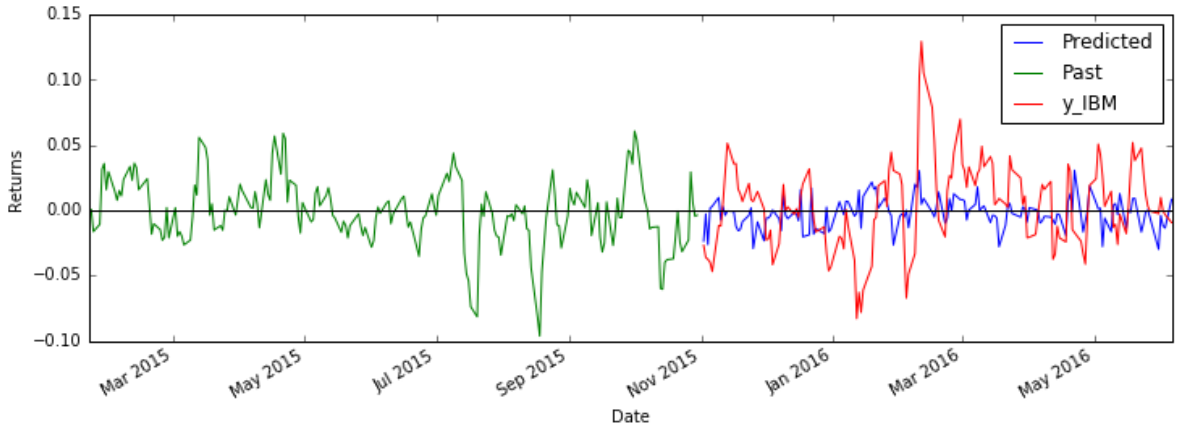


Trained from January 21, 2015 to October 30, 2015
Tested from November 02, 2015 to June 06, 2016



```
Only 40.27% of the predicted returns were greater than 0.0, while
56.38% of the actual test data returns were greater than 0.0.
44.05% of the time the return was positive, the model predicted it
would be positive.
64.62% of the time the return was negative, the model predicted it
would be negative.
```

```
/Users/seanhegarty/Library/Enthought/Canopy_64bit/User/lib/python
2.7/site-packages/pandas/core/frame.py:1997: UserWarning: Boolean
Series key will be reindexed to match DataFrame index.
  "DataFrame index.", UserWarning)
```



The above first three sets of graphs explore the effects of averaging the output of a linear regression model and a k-Nearest Neighbors regression model. In some situations it appears that linear regression outperforms k-nearest neighbors, while not so much in others. Using an ensemble of models can be beneficial to estimating future return and is used in the final model.

The final two graphs are used to express how well the final model has performed on the task. The first graph is a scatter plot that expresses the correlation between the predicted and actual results for the test set. The legend shows the correlation, which exceeds that of the benchmark by being both larger in magnitude and positive. On the other hand, RMSE for the predicted is still slightly greater than that of the benchmark. Still, having a better correlation implies that trading opportunities can be recognized at all, while a prediction of 0.0 returns will never produce recognize any opportunities.

The statistics shown above the last graph, express how often the predicted and test returns were above zero and also how sensitive the model is to predicting positive and negative values. It appears that the model performs better at predicting negative values than it does at predicting positive values. This information can be utilized in a trading strategy to short stocks more accurately than random chance.

Lastly, the bottom graph shows the 5 day returns recognized over the training data on the left side in green. In the fourth quarter on the right of the graph are the predicted returns over the test data and the actual 5 day returns. Future explorations of this type can search to classify whether or not the return will be greater than or less than 0.0 returns.

Results

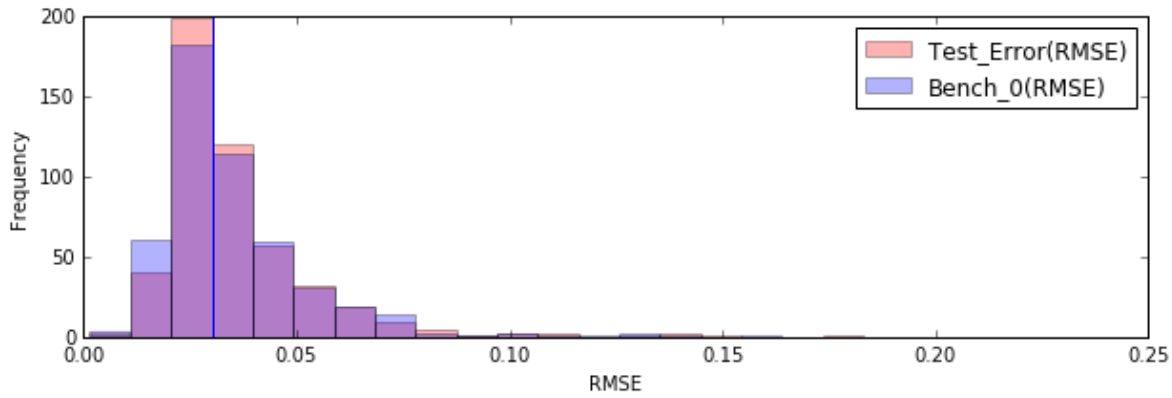
Model Evaluation and Validation

Ensemble Models

The below predictions use an ensemble of regression models. Specifically, there are two types of regression models, k-Nearest Neighbors and Linear Regression. As in the previous section, the model is run for all ticker symbols in `spy_list.csv` and the results are analyzed using the below statistics.

In [12]:
```
# Predict 5 day returns using Ensemble of Regression Models
predict_spy_future(horizon=5, use_prices=False)
# Display performance statistics as they relate to the benchmark fo
r all S&P 500 companies in spy_list.csv
predicted_returns_results = pd.read_csv("return_results.csv", index
_col="Date")
# Calculate meaningful statistics
averages = predicted_returns_results[
    ["Test_Error(RMSE)","Bench_0(RMSE)","Test_Corr"]].mean(axis=0).
to_frame(name="Average")
medians = predicted_returns_results[
    ["Test_Error(RMSE)","Bench_0(RMSE)","Test_Corr"]].median(axis=
0).to_frame(name="Median")
stdevs = predicted_returns_results[
    ["Test_Error(RMSE)","Bench_0(RMSE)","Test_Corr"]].std(axis=0).t
o_frame(name="Standard Deviation")
df = averages.join(medians).join(stdevs)
print df
# Plot histogram to show performance against benchmark.
predicted_returns_results[
    ["Test_Error(RMSE)","Bench_0(RMSE)"]].plot.hist( color=
["r","b"], bins=30, alpha=0.3, figsize=(10,3) )
plt.axvline(predicted_returns_results.median(axis=0)[["Test_Error(R
MSE)"]].values, color="r")
plt.axvline(predicted_returns_results.median(axis=0)[["Bench_0(RMS
E)"]].values, color="b")
plt.xlabel("RMSE")
plt.xlim((0,.25))
plt.show()
```

```
                   Average    Median  Standard Deviation
Test_Error(RMSE)  0.036385  0.030461            0.022293
Bench_0(RMSE)     0.035533  0.030249            0.021356
Test_Corr         0.215578  0.229058            0.198156
```



Justification

The results, as shown below, are comparable to the benchmark over a number of weeks. Unfortunately, when the test error beats the benchmark, the overall error is too substantial to consider as a feasible predictor of returns. However, to retiterate, the benchmark estimate does not offer any trading opportunities while the predictions do.

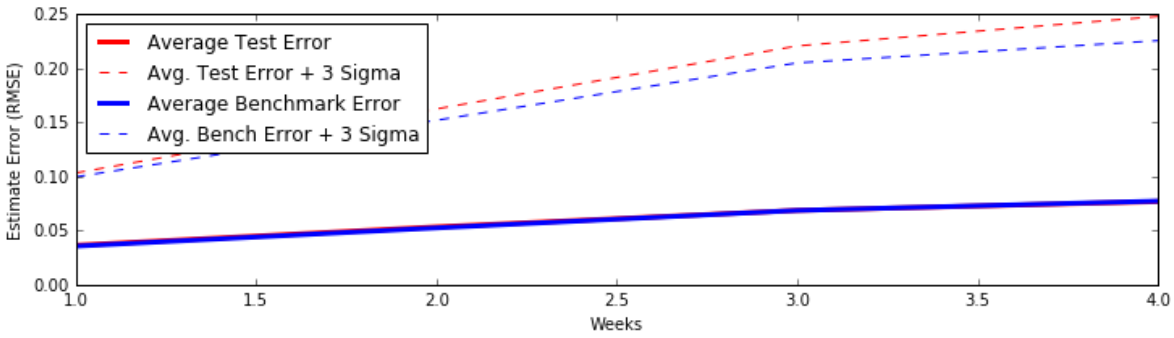
It is important to understand that the market is a very noisy collection of data that is a conglomeration of a variety of factors. Certainly, not all indicators are known about and most investors underperform against the market. This statement is make clear that predicting the returns of any stock using only historical pricing data has a very low success rate.

This model uses the most readily available information, historical pricing data. Companies today use metadata from new feeds, twitter, etc. to explore the value of public sentiment as a feature in predicting price movements for a given company. However, the predictions calculated by this model can help in a trading strategy. There is a moderate positive correlation betweent the predictions and the actual results. The information provided in the predictions can be used to develop a trading strategy by buying stocks for the specified horizon when the prediction is strongly positive.

In [13]:
```
bench_list, error_list = [], []
bench_std_list, error_std_list = [], []
weeks = range(1,5)
for i in range(len(weeks)):
    try:
        predict_spy_future(horizon=5*weeks[i])
        test = pd.read_csv("return_results.csv", index_col="Date")
        error_list.append(test.mean(axis=0)[["Test_Error(RMSE)"]])
        bench_list.append(test.mean(axis=0)[["Bench_0(RMSE)"]])
        error_std_list.append(test.std(axis=0)[["Test_Error(RMS
E)"]])
        bench_std_list.append(test.std(axis=0)[["Bench_0(RMSE)"]])
    except:
        weeks[i] = None
        continue
weeks = [w for w in weeks if w]
```

In [14]:
```
error_upper_band = [mn + 3*std for mn, std in zip(error_list, error
_std_list)]
bench_upper_band = [mn + 3*std for mn, std in zip(bench_list, bench
_std_list)]
```

In [15]:
```
fig, ax = plt.subplots(figsize=(12,3))

test_error_plot, = plt.plot(weeks, error_list, 'r', lw=3)
test_error_upper_plot, = plt.plot(weeks, error_upper_band, 'r--')
bench_error_plot, = plt.plot(weeks, bench_list, 'b', lw=3)
bench_error_upper_plot, = plt.plot(weeks, bench_upper_band, 'b--')
plt.legend([test_error_plot, test_error_upper_plot, bench_error_plo
t, bench_error_upper_plot],
           ["Average Test Error",
            "Avg. Test Error + 3 Sigma",
            "Average Benchmark Error",
            "Avg. Bench Error + 3 Sigma"],
           loc="upper left")
plt.xlabel("Weeks")
plt.ylabel("Estimate Error (RMSE)")

plt.show()
```



Conclusion

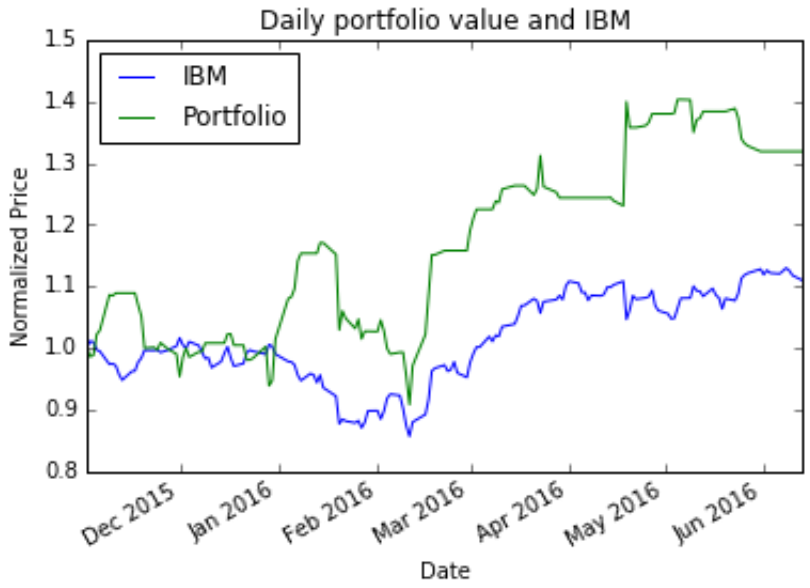
Free Form Visualization

In [16]:
```
from learner_strategy import learner_strategy
from marketsim import compute_portvals, test_code
```

The IBM predicted returns are explored for the following trading strategy. The same horizon that returns are predicted for is used to buy and sell stocks over the same time. The starting value as well as the number of shares to buy and sell are defined so that someone does not have to be very wealthy to see the potential benefits of using these predictions. The threshold is used to maximize the number of trades made while not simply following the market.

In [17]:
```
returns = dframe.Predicted.dropna()
returns = returns.to_frame()
returns.columns = ["Returns"]

symbol = "IBM"
horizon = 5
num_shares = 10
orders_file = "./orders/learner_orders.csv"
start_val = 1000
learner_strategy(data = returns,
                 threshold = 0.01,
                 sym = symbol,
                 horizon = horizon,
                 num_shares = num_shares,
                 shorting = True)
test_code(of = orders_file, sv = start_val)
# test_code(of = orders_file, sv = start_val) uses compute_portvals
(orders_file, start_val, allowed_leverage=2.0)
# to compute the portfolio value.
```

```
Starting Cash Value = $1000
Ending Cash Value = $1318.81143
Date Range: 2015-11-02 to 2016-06-13

Sharpe Ratio of Fund: 1.36939433855
Sharpe Ratio of IBM: 0.897396863544

Cumulative Return of Fund: 0.31881143
Cumulative Return of IBM: 0.109549461307

Standard Deviation of Fund: 0.0243537337046
Standard Deviation of IBM: 0.013671678585

Average Daily Return of Fund: 0.00210084402857
Average Daily Return of IBM: 0.000772869407028

Final Portfolio Value: 1318.81143
```



The plot above shows both the value of IBM and the portfolio normalized by the starting cash value, $1000, of the investor. The strategy utilized in this portfolio is to make a trade for the number of shares, either Buy or Sell, if the absolute value of the 5 day return prediction exceeds the threshold value, 0.01, and then do the reciprocal, either Sell or Buy, after the horizon, 5 trading days, has passed.

The analysis of these trades occurs over the testing data, which means the model is operating on data it has not been trained on. According to this representation of how the model can be utilized, it appears that a successful trading strategy is possible even with the formerly stated flaws in model performance.

In this instance, the model achieved a ~32% return while the actual value of the company rose only ~11%. That is a 20% better performance than the company itself. Furthermore, the sharpe ratio for the portfolio following the prediction-based trading strategy is roughly 1.5x the sharpe ratio of IBM returns. Sharpe ratio is a measure commonly used in trading for calculating the return as adjusted by the risk.

Sharpe Ratio = Avg(returns - risk free rate) / Std. Dev. of Returns * SquareRoot(Number of Trading Days Per Year)

Reflection

The process of developing this program began with retrieving the data from Yahoo Finance. After that, there were many potential indicators to explore, but the information that proved most useful was the information almost immediately provided by the data retrieval. The data needed to be processed so that it was on the same scale given the dynamic nature of the market. Given the processed data, predictions were improved by observing the combination of two regression models, Linear and k-Nearest Neighbors. While the results did not immediately appear useful, when observed in the context of a trading strategy, the results proved that they could functionas a method to achieve greater returns than the market in the instance explored in this project.

Unfortunately, while a great amount of time was used to explore the effectiveness of indicators, none proved to be very useful. This is not to say that indicators do not serve a purpose in a trading strategy, but they were not useful for predicting future returns.The most difficult aspect of this project is how noisy the returns are well-known to be in the short and long term. It was difficult to find meaningful features that improved the accuracy of the model.

While the final solution does not exceed the accuracy of the benchmark, I am satisfied that it can contribute to a moderately successful trading strategy. As a disclaimer, stock trading is a very volatile way of making money. In other words, it is just as likely that a trader will lose money as a trader will make money. In short, stock trading can be very much like gambling, and all investor can engage in trading at their own risk.

Improvement

There are a myriad of improvements that can be made to this product. There are hundreds, if not thousands, of technical indicators used in inter- and intra-day trading. There is continuous analysis and debate over what indicators are most effective, just as there is the same analysis and debate over which models are most effective in this realm.

In the same vein, sentiment analysis can be explored as a feature to better predict stock returns. There is not enough storage space or computing power on one computer to explore that effectiveness in a reasonable amount of time. Information from a large number of sources would need to be stored over time to explore sentiments effectiveness in predicting returns for even one company.

Another way to enhance the usefulness of this program is to use it in conjunction with a reinforcement learner. The trading strategy can be controlled by a reinforcement learning agent that can adjust to learn an optimal trading strategy and maximize returns.