FIN30290

Recent Research Topics in Finance

Stacked Ensembling and Algorithmic Trading Project

Group 3 - Winx

April 2021

Name	Student Number
John Purcell	17370751
Thomas Smyth	17462644
Kevin Gannon	17340596

Table of Contents

1. Introduction

1.1 Mandate

In this report we fit five classification and regression models to Visa log returns data using various lags of log returns and trading volumes. We then perform five-fold cross validation on each of the classification techniques and build a stacked ensemble with these models. We then test how this ensemble performs out-of-sample. The aim of this project is to test whether we can accurately predict directional changes in Visa's daily closing stock price using a meta model of base models. We have used the Julia programming language to perform our analysis citing its speed and versatility in fitting models as a clear advantage over other dynamic programming languages.

1.2 Visa

Visa Inc. (NYSE:V) is an American financial services firm which facilitates electronic funds transfers worldwide, most commonly through Visa-branded credit cards, debit cards and prepaid cards. It is one of the most valuable companies in the world and is one of the most recognizable credit card companies globally. Visa has a market capitalisation of approximately $486 billion and has a monthly beta of 0.98 (last five years). It has performed very well in recent years, particularly in 2020. Given Visa's strong historical trading performance and relatively low beta, we thought it would be an interesting equity to test an ensemble of classification models on.

1.3 Data

Visa's initial public offering took place on 18 March 2008. Thus, we have analysed daily adjusted closing prices and trading volume of Visa from 19 March 2008 to the present, giving us over 3,200 data points for each variable.

1.4 Data Preparation and Pre-processing

We cleaned the data by removing rows with non-trading days and miscellaneous null values for either closing price or trading volume. We calculated the daily log returns on the adjusted closing prices and truncate the log returns data. We remove the earliest observation of trading volume as we do not have a log return figure for that day. We then store ten lags of log returns and log volume, removing the first ten trading days from our dataset. We also produce a column of ones and minus ones corresponding to positive and negative daily returns respectively for each trading day stored. Finally, we remove the final 252 trading days (one trading year) from the dataset and keep these aside as "testing" data for analysing pseudo-out-of-sample performance of our meta model. The remainder of our data is our "training" data which we use to build and validate our models. The reason we split our data prior to model fitting was to ensure that we would be testing our ensemble on true out of sample data.

2. Methodology

2.1 K-fold Cross Validation

Cross validation allows us to use our data more efficiently. It gives us more information on our fitted learning models, as we can calculate our cv error/accuracy and see how the class of model performs out-of-sample. We use 5-fold cross validation due to its compatibility with the stacked ensemble, its simplicity and for the results in skill estimates that generally have a lower bias than other methods. Given our variety of models, the very general and flexible nature of this method was very attractive, as opposed to LOOCV which would have been more computationally expensive. LOOCV can also result in higher bias than k-fold. The procedure we followed is as follows:

i. The training data is then split randomly into 5 folds

ii. A base model (e.g SVM) is fitted on the k-1 folds and predictions are made for the kth fold

iii. This process is iterated until every fold has been predicted

iv. We repeat this for every model and collate the predictions for later use in the neural network

2.2 Stacking

We used stacked generalisation or "stacking" to configure our meta model. This method of ensembling allows us to train a learning model (via deep neural network) on the out-of-fold predictions of our various base models. We seek to optimise the output of our five base models in making a single combined forecast, to improve out-of-sample predictions. Stacked ensemble models are frequently used in modelling and forecasting competitions globally and generally perform better when a diverse suite of base models is used with low multicollinearity in forecasts. Our over-arching aim in training a stacked ensemble is to reduce both bias and variance, however we could be in danger of overfitting due to the large number of parameters used to generate our single combined forecasts. We also incorporate the original input variables in the training process alongside our "expert" forecasts. We train the model against actual outcomes; however, given that our base models vary in type of output, i.e. probabilities, predicted returns (continuous), and binary classifier labels, we converted the output of each model to a 1/-1 binary output if it predicts a positive/negative return. We must acknowledge however that a stacked ensemble generally performs better if it is trained on probabilities as opposed to classifiers, but this option was not available to us due to the models we were restricted to using. We used the Flux package in Julia to train our neural network on the expert forecasts and lagged returns and volumes. We train a three layered deep neural network and use the $(m(x)y-1)^2$ loss function. When forecasting with our ensemble, we used base models fitted on the whole training sample, to maximise the incorporation of information in our data.

2.3 Proportional vs Long/Short Investing

For proportional investing, we invested/shorted a proportion of our wealth equal to the neural network scores which were between -1 and 1. This meant that we scaled our investment amount based on the strength of our ensemble predictions. For our long/short strategy, we invested our whole wealth if we predicted a gain and shorted our whole wealth if we predicted a loss.

3. Base Models

We have incorporated five different classes of base models in our ensemble.

3.1 Lasso Regression Model

We chose to fit a lasso model as opposed to a ridge regression since we were dealing with a large number (20) explanatory variables and that the lasso can force some coefficients to be exactly equal to zero as opposed to simply shrinking insignificant coefficients toward zero, to improve out of sample performance. We thought that a sparse regression model would perform better for our data. It differs from a standard linear regression by introduction of an ℓ_1 penalty term and λ parameter. The lasso model is fitted and chosen to minimise the sum of the residual sum of squares and lambda times the sum of the absolute value of model coefficients. When $\lambda = 0$, our lasso model is equal to an ordinary least squares model. In our model we fit the β_i coefficients first, and then choose our λ coefficient by minimising the 5-fold cross-validation error each time we fit the model. This means that each time the model is fit in each fold, it inherently applies 5-fold cross validation within each fold.

After converting our class of continuous predictions from the lasso to binary classifiers, our cross-validation prediction accuracy for all five-folds was only 54.331%. The lowest mean squared error for the five models fit was 0.00016. When fitting the lasso model on the whole training data, we get the following output in Figure 1:

	df	pct_dev	λ
[1]	7	0.0155001	0.000162982

Figure 1 - Lasso Regression Training Output

Our optimised regularisation coefficient λ is 0.000163 and our model only has three degrees of freedom. This model has an in-sample prediction accuracy of 56.053%. This model generates the cumulative log returns using a long/short trading strategy during the in-sample and out-of-sample periods illustrated in Figure 2 and Figure 3 below.

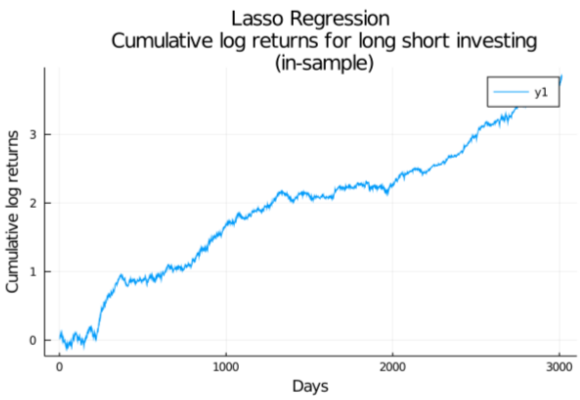


Figure 2 - Lasso Regression Training Returns

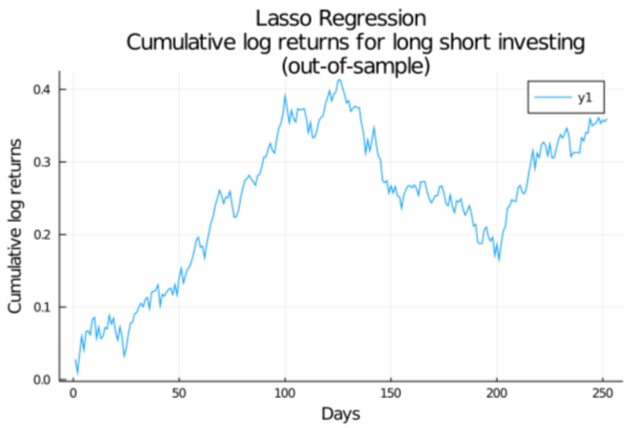


Figure 3 - Lasso Regression Test Returns

This model and strategy give an in-sample annualised Sharpe ratio of 1.61 and a higher out-of-sample Sharpe ratio of 1.76.

3.2 Logistic Regression Model

Logistic regression is an alternative method to use other than the simpler linear regression model. logistic regression does not look at the relationship between the variables as a straight line. Instead, logistic regression uses the natural logarithm function to find the relationship between the variables and uses test data to find the coefficients. The function can then predict the future results using these coefficients in the logistic equation.

This final equation is the logistic curve for the regression. It models the non-linear relationship between the x variables and y with an 'S'-like curve for the probabilities that y = 1, i.e that event that y occurs.

$$P(y = 1|x) = \frac{e^{\beta_0+\beta_1+....+\beta_{20}}}{1+e^{\beta_0+\beta_1+....+\beta_{20}}}$$

For the purpose of this model, we used 10 day lagged returns and volume, giving us 20 explanatory variables.

After having fitted our model, we took our probabilities and rounded them to generate "Buy/Sell" indicators. Using these rounded values, we were able to model this trading strategy and see how our signals performed. We can see how the model performs on our training and testing data if the investor was to invest both long and short below based on the model's predictions below:

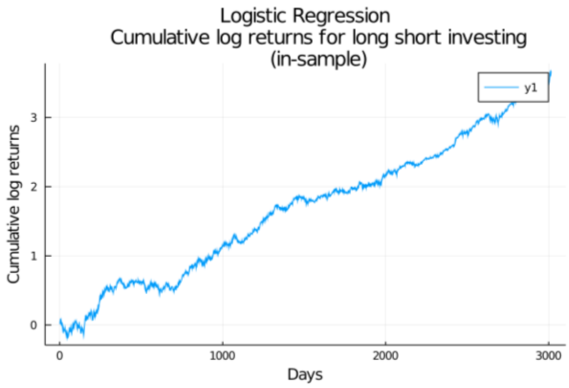


Figure 4 - Logistic Regression Training Returns

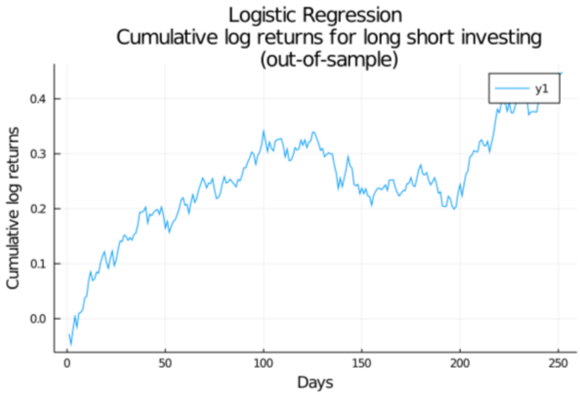


Figure 5 - Logistic Regression Test Returns

This model and strategy give an in-sample annualised Sharpe ratio of 1.53 and a higher out-of-sample Sharpe ratio of 2.19.

3.3 Support Vector Machine

The support vector machine (SVM) is used in a binary classification setting to predict the class of an object. It was developed in the 1990s in the computer science community and has gained popularity ever since. It is an extension of the support vector classifier that instead uses kernels to expand feature space. A non-linear kernel allows for a flexible decision boundary, whereas a linear kernel (which is equivalent to a support vector classifier) is linear in its features. The advantage of using kernels over simply using functions of the original features to enlarge feature space comes down the computational storage that is saved in doing so. It is interesting to note that using kernels to enlarge feature space is not unique to the SVM, it can be applied to other classification methods such as logistic regression. However, for historical reasons, it is more commonly used in SVM.

When we applied the support vector machine using the 5-fold approach, we achieved a cross validation prediction accuracy of 54.789%. The model that was built on the training sample achieved a training predictive accuracy score of 54.66%. These both outperform the random classification case, assuming that random classification achieves a predictive accuracy score of 50%. We can therefore see the predictive power of the model. We can see how the model performs on our training and testing data if the investor was to invest both long and short in Figure 6 and Figure 7 below:

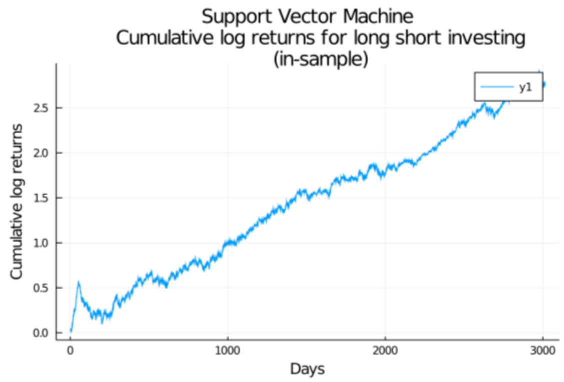


Figure 6 - SVM Training Returns

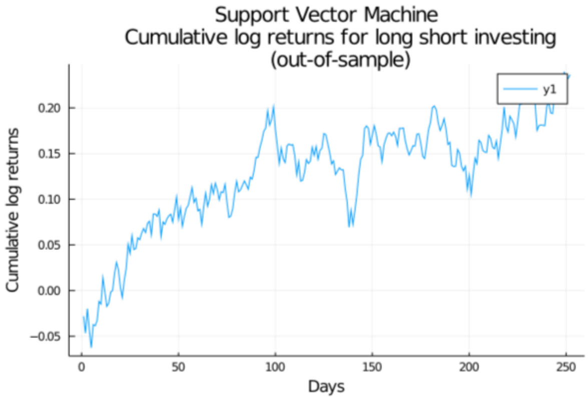


Figure 7 - SVM Testing Returns

This model and strategy give an in-sample annualised Sharpe ratio of 1.15 and an approximately equal out-of-sample Sharpe ratio of 1.15.

3.4 Gradient Boosting

Gradient boosting attempts to improve the predictive performance of a decision tree. These decision trees can be used in a regression or classification context. In our case, we use a regression decision tree within the gradient boosting model to predict the log returns of our equity. We then transform this output into a binary class (of +/-1) based on whether we expect the stock to go up or down. Boosting improves the predictive accuracy of decision trees by "learning slowly" thus, reducing overfitting. Each tree is grown sequentially using the residuals from previously grown trees rather than the outcome Y. We then add this new decision tree into the fitted function in order to update the residuals. The aggregation of many decision trees improves the overall performance of the model.

Our highest prediction accuracy for the five-fold cross validation was found using this model with a predictive accuracy score of 57.676%. The highest training predictive accuracy is produces from this model also with a score of 61.692%. Furthermore, this model performs best over the training data period based off cumulative log returns. Performance over the testing and training periods for a long/short strategy based off of these model predictions can be seen in Figure 8 and Figure 9 below:

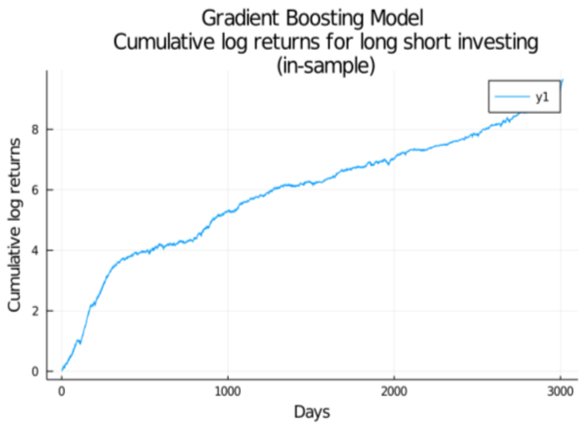


Figure 8 - Gradient Boosting Training Returns

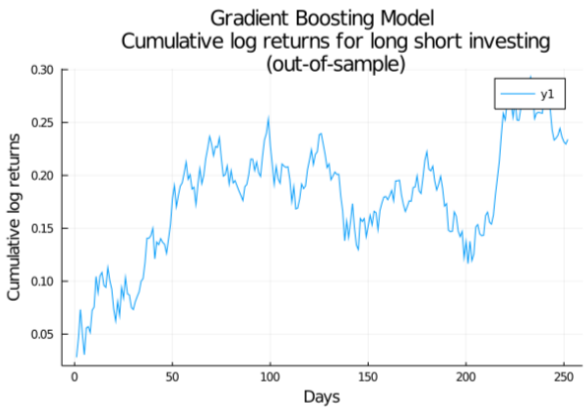


Figure 9 - Gradient Boosting Testing Returns

This model and strategy give an in-sample annualised Sharpe ratio of 4.14 and a much lower and relatively disappointing out-of-sample Sharpe ratio of 1.14, indicating that the model may in fact be overfitted.

3.5 Linear Discriminant Analysis

Linear discriminant analysis, as a classification method, is an alternative to and less direct than logistic regression. As opposed to modelling the probability that our return is positive given the predictor variables, we model the distribution of the predictors separately given the log return, and then use Bayes' theorem to convert these around into estimates for the probabilities of a positive return given the predictors. For our multivariate model, we assume that the predictors follow a multivariate Gaussian distribution (normal) however, which is quite a strong assumption. We include the discriminant analysis model as well as the logistic regression in our ensemble as when the classes (positive/negative return) are well-separated or when there are few examples from which to estimate the parameters, the parameter estimates for the logistic regression model are surprisingly unstable. The LDA model does not suffer from these shortfalls.

When we applied the LDA model using the 5-fold approach, we achieved a poor cross validation prediction accuracy of 54.894%. The model that was built on the training sample achieved a similar predictive accuracy score of 56.119%. These both outperform the expected random classification case of 50% accuracy. We can see how the model performs on our training and testing data if the investor was to invest both long and short in Figure 10 and Figure 11 below:

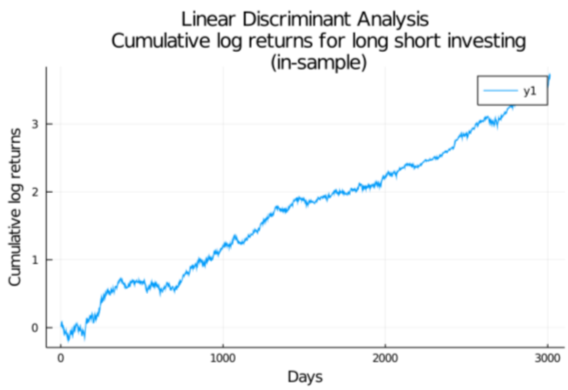


Figure 10 - LDA Training Returns

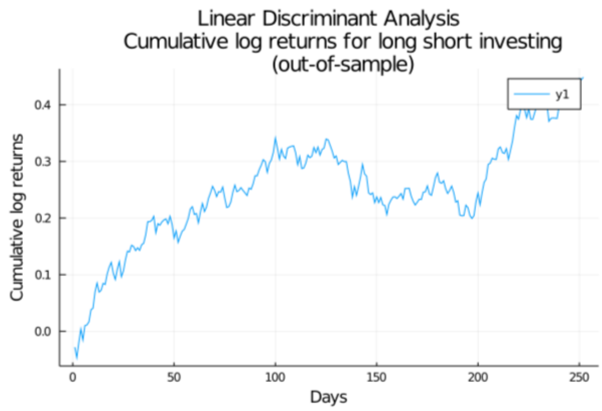


Figure 11 - LDA Testing Returns

This model and strategy give an in-sample annualised Sharpe ratio of 1.55 and a higher out-of-sample Sharpe ratio of 2.19.

4. Results

4.1 In-sample Ensemble

The forecast for our in-sample ensemble was produced by running the training data of the 20 original features of lagged log returns and lagged volume alongside our 5 "expert" forecasts into our deep neural network model. Based off the whole training sample, we found the stacked ensemble's prediction accuracy to be 58.133%. This result cuts both ways; on one hand, the in-sample ensemble outperformed the random classification case quite significantly. This once again assumes that random classification achieves a predictive accuracy score of 50%. However, based off the training data, the neural network results did not manage to outperform all the base models individually. The gradient boosting produces a predictive accuracy score of 61.692% on the training data. Our results are illustrated in Figure 12:

In-Sample	Annualised Sharpe	Annualised Returns
Long/Short	2.36	47.05%
Proportional	2.85	28.77%
Buy and Hold	0.72	14.59%
Lasso Regression	1.61	32.25%
Logistic Regression	1.53	30.65%
Support Vector Machine	1.15	23.19%
Gradient Boosting	4.14	80.75%
LDA	1.55	31.14%

**Figure 12 – In Sample Returns Stats*

4.2 Out-of-sample Ensemble

Rather disappointingly, the out-of-sample prediction accuracy performs noticeably worse than the in-sample result. Based off training data of the most recent 252 trading days, our ensemble had a predictive accuracy of 54.762%. This is only slightly better than randomly predicting whether the equity will rise or fall in value. Our results are illustrated in Figure 13:

Out-of-Sample	Annualised Sharpe	Annualised Returns
Long/Short	1.71	34.99%
Proportional	1.92	19.69%
Buy and Hold	1.15	23.64%
Lasso Regression	1.76	35.99%
Logistic Regression	2.19	44.87%
Support Vector Machine	1.15	23.64%
Gradient Boosting	1.14	23.39%
LDA	2.19	44.87%

**Figure 13 – Out-of-Sample Returns Stats*

**Note the period for the in sample returns for the different trading strategies is not the same period as the period for the in sample returns for the base models. This is due to how five-fold cross validation being used to train the Neural Network model as opposed to the full training sample for the base models.*

4.3 Long/Short Trading Strategy Performance

We encoded a long/short trading strategy in terms of 1's and -1's. When we believe the stock will rise in value, our model will assign a 1 for that trading day, hence we will go long on the stock. Conversely, when we believe the stock will fall in value, our model will assign a -1 for that trading day, hence we will short the stock. If our predictions are accurate, we will be rewarded in both instances. However, there in inherent risk in short selling as there is the theoretically potential for infinite loss. This strategy produced an annual log return of 34.99% and an annualised Sharpe ratio of 2.36. Note that, for simplicity's sake, we opted not to include the risk-free rate in the Sharpe ratio calculations, given that rates have been approximately zero for the last year making the difference between returns and excess returns negligible. The cumulative returns for the long/short trading strategy are illustrated in Figure 14 below:

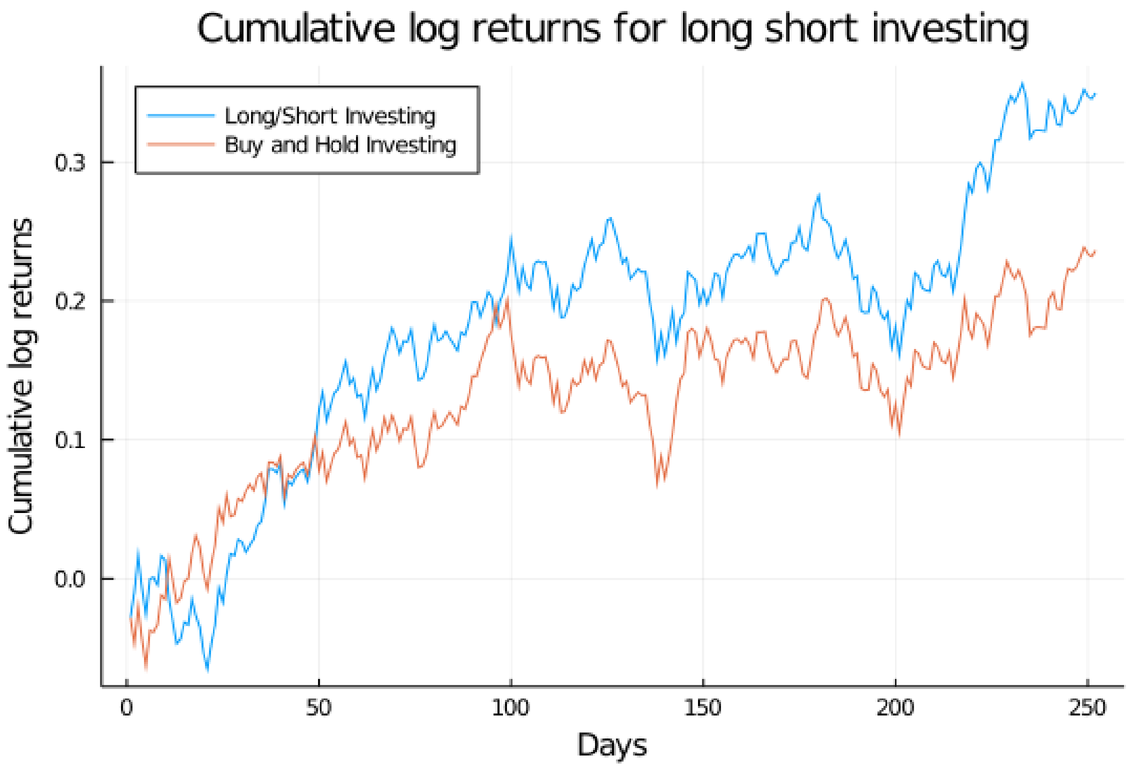


Figure 14 – Long/Short Testing Returns

The long/short investing strategy has a higher terminal wealth than passive buy and hold investing. This strategy also appears to be less volatile than regular investing. The Sharpe ratio summaries this return to risk trade off. We find that long/short investing does indeed have a higher Sharpe ratio than buy and hold investing with ratios of 1.71 and 1.15 respectively (See Figure 13)

4.4 Proportional Trading Strategy Performance

We employed a proportional trading strategy by investing an amount corresponding to how confident our model was of an up/down day. When we are more confident of an up day, our model will predict a value that is closer to 1. The max testing value our model predicted was 0.86. Conversely, when we confidently believe that the stock will fall in value, our model will assign a value closer to -1 for that trading day. The minimum testing value our model predicted was -0.56. This strategy is inherently less risky than the long/short strategy as we are no longer taking as hard a stance on our less confident predictions that are close to zero. However, given that we are only ever investing in fractions, we expect a lower cumulative return and lower standard deviation of returns than the long/short strategy. The cumulative returns for the proportional trading strategy are illustrated in Figure 15 below:

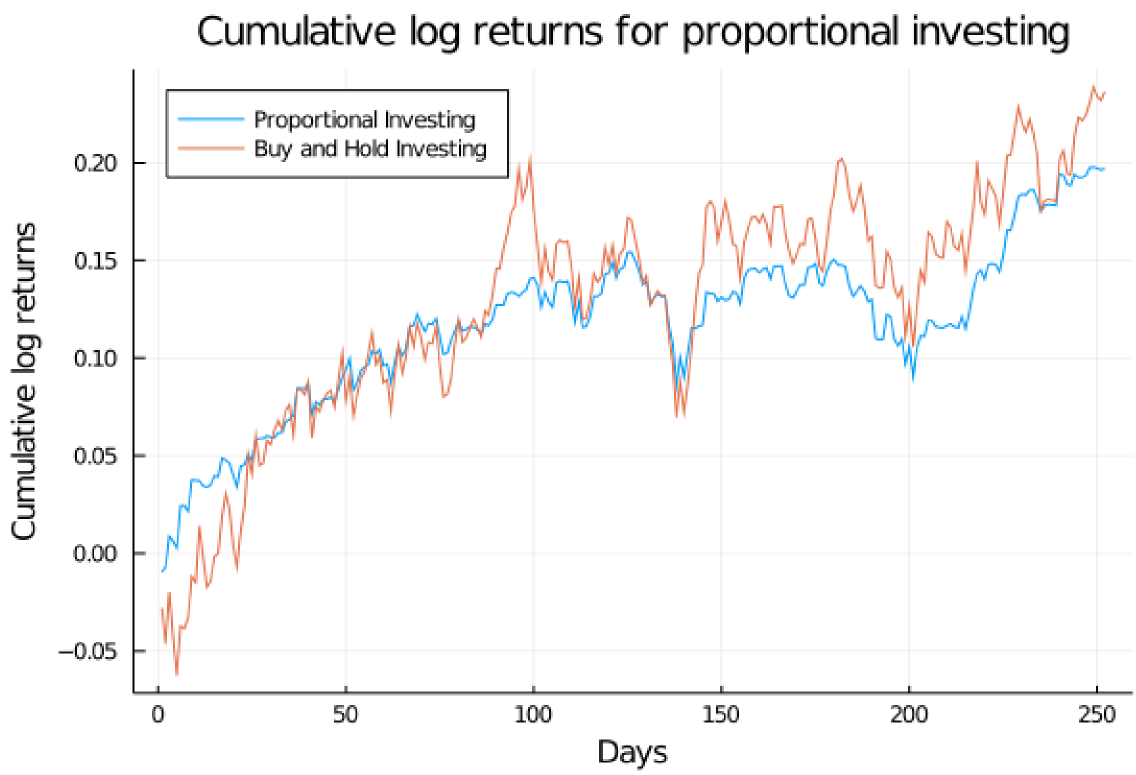


Figure 15 – Proportional Trading Testing Returns

This strategy produced an annual log return of 19.69% and an annualised Sharpe ratio of 1.92 for our out of sample period. While our returns weren’t as high as the other two investment strategies, we found that it resulted in undertaking much less risk. This can be seen in the higher Sharpe ratio. This is in line with what we expected. Therefore, this strategy would be more suited to a more risk averse investor (See Figure 13).

5. Conclusions

We wanted to compare the out-of-sample accuracy of our ensemble model to each of our individual models. From Figure 16 we can see that our ensemble model underperforms some of our individual models when they are tested out-of-sample.

Model Type	Accuracy
Ensemble	**54.76%**
SVM	**53.97%**
Logistic Regression	**59.13%**
Lasso Regression	**59.92%**
LDA	**59.13%**
Gradient Boosting	**54.37%**

Figure 16 – Out-of-Sample Model Accuracy

In Section 4 we saw that our proportional investing strategy outperformed our long-short trading strategy on an annualised Sharpe ratio basis. However, to conclude, we wanted to compare our trading strategies to a buy and hold strategy, where one goes long the stock at beginning of the period and holds it throughout our testing sample.

Figure 17 – Buy and Hold Testing Returns vs Long/Short Testing Returns

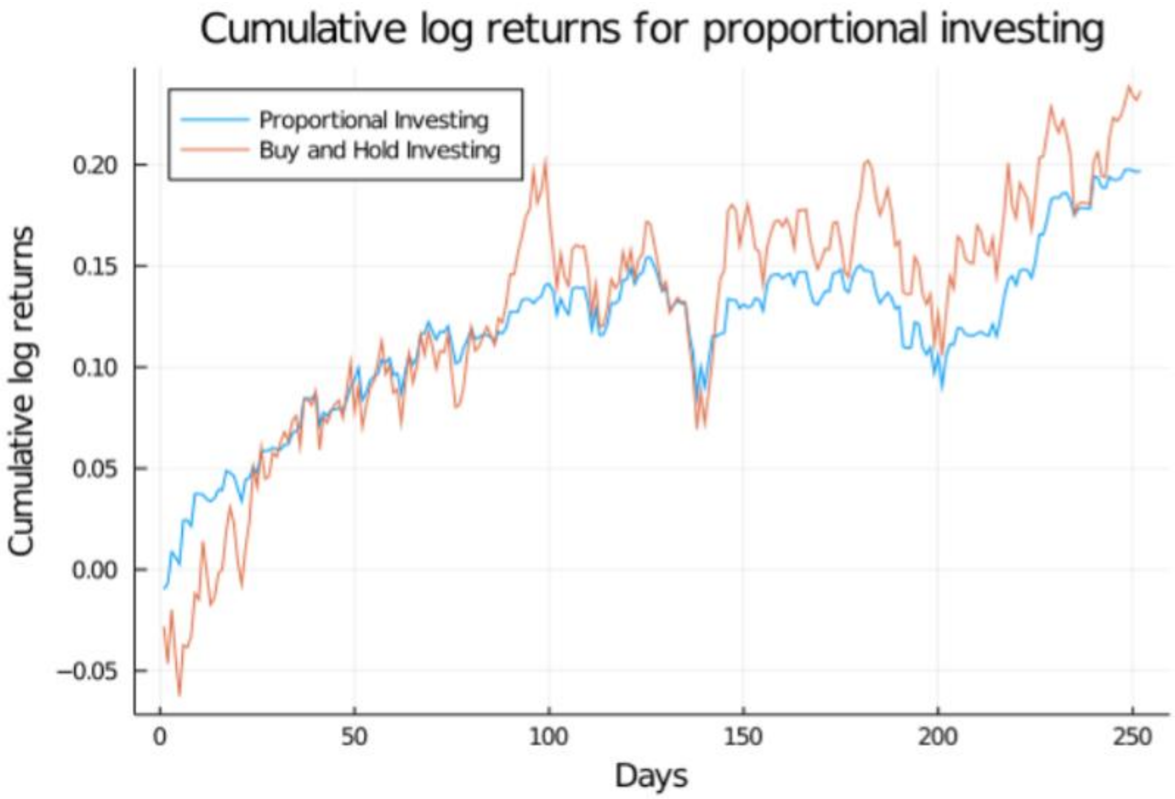


Figure 18 – Buy and Hold Testing Returns vs Proportional Testing Returns

The buy and hold strategy returned annualised returns of approximately 24% over our testing period, this compares to 35% for our long-short strategy and 20% for our proportional trading strategy. If we look at the return profiles above it is clear that the proportional investing strategy provides the least volatile returns and as a result it may be more pertinent to use risk-adjusted measures to evaluate our strategies

The buy and hold strategy underperforms against our strategies on an annualized Sharpe ratio basis, recording a Sharpe of c. 1.15, compared to c. 1.71 and c. 1.92 for our long-short and proportional trading strategies, respectively. Although our proportional strategy provided us with the lowest cumulative returns it is the best on a risk-adjusted basis.

As a result of this analysis, we recommend that if an investor seeks to maximize cumulative returns, they should follow our long-short strategy but if they want to achieve the highest risk-adjusted returns they would be best suited following our proportional investing strategy.

6. Appendix

6.1 Julia Code

In [1]:
```
using GLMNet
using RDatasets
using MLBase
using DecisionTree
using Distances
using NearestNeighbors
using Random
using LinearAlgebra
using DataStructures
using LIBSVM
using CSV
using DataFrames
using Statistics
using GLM
using Distributions
using Plots
using DataFrames
using MarketData
using Random
```

In [2]:
```
using Flux
```

In [3]:
```
using XGBoost
```

Step 1

Download Data

Our equity of choice was visa which has the ticker "V".

In [4]:
```
# Download visa data
visa=DataFrame(yahoo("V"));
```

Step 2

Prepare Data

We prepared input and output data matrices of lagged daily (‘trimmed’) logarithmic return, removing non-trading (exactly zero-return output) days as shown in class. (In addition to lagged return information, you should include several lags of log(1+Volume) as Inputs)

In [5]:
```
# Calculate log returns
lrt=log.((visa.AdjClose[2:end])./(visa.AdjClose[1:end-1]))
lrt=tanh.(lrt/0.03)*0.03;
```

In [6]:
```
# Calculate log(1+volume)
log_vol = log.(visa.Volume[2:end] .+= 1);
```

In [7]:
```
# Convert to dataframe
data=convert(DataFrame, hcat(lrt,log_vol));
```

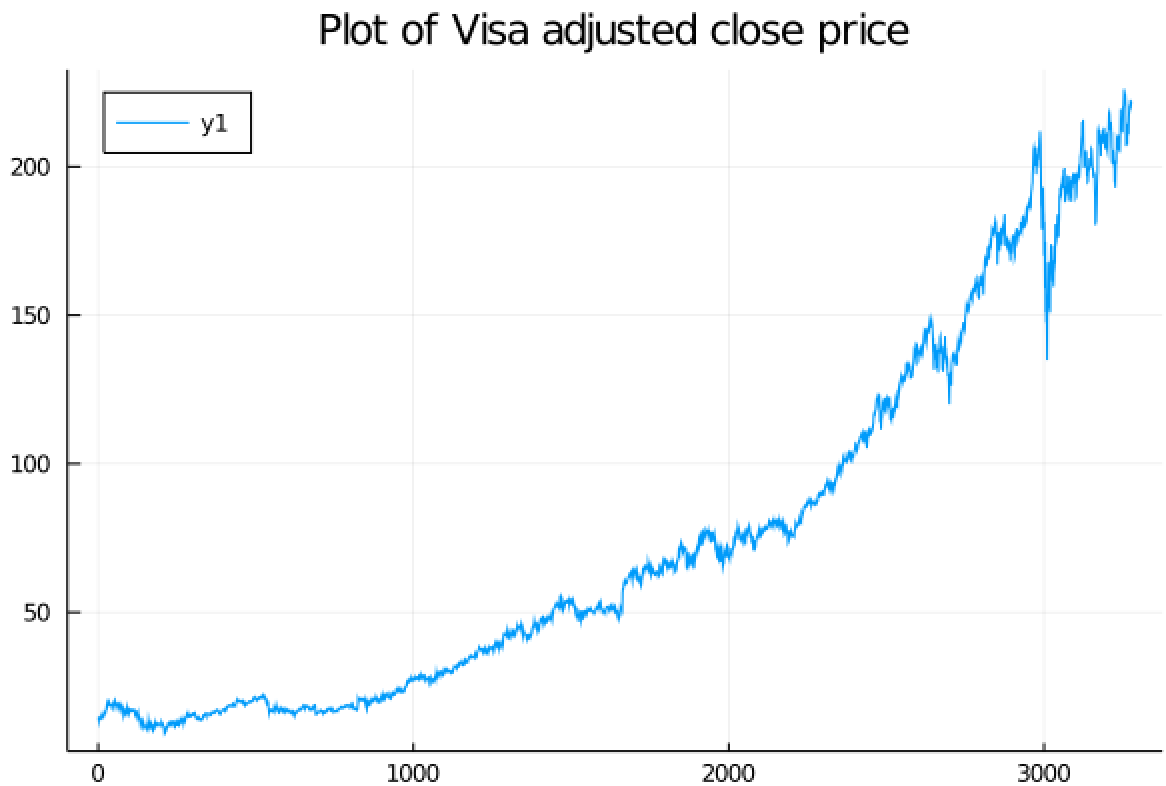
In [8]:
```
#Remove zero values for returns
remove = data[:,1].!=0
data=data[remove,:];

#Remove the same rows in visa dataframe
visa = (visa[2:end,:])[remove,:];
```

In [9]:
```
#Remove zero values for volume
remove = data[:,2].!=0
data=data[remove,:];

#Remove the same rows in visa dataframe
visa = visa[remove,:];
```

In [10]:
```
# Make Lag matrix of log returns
global lag_rets=zeros(length(data[:,1])-10,0)
for i=1:10
global lag_rets
    lag_rets=[lag_rets data[i:(end-11+i),1]];
end
rets=data[11:end,1];
```

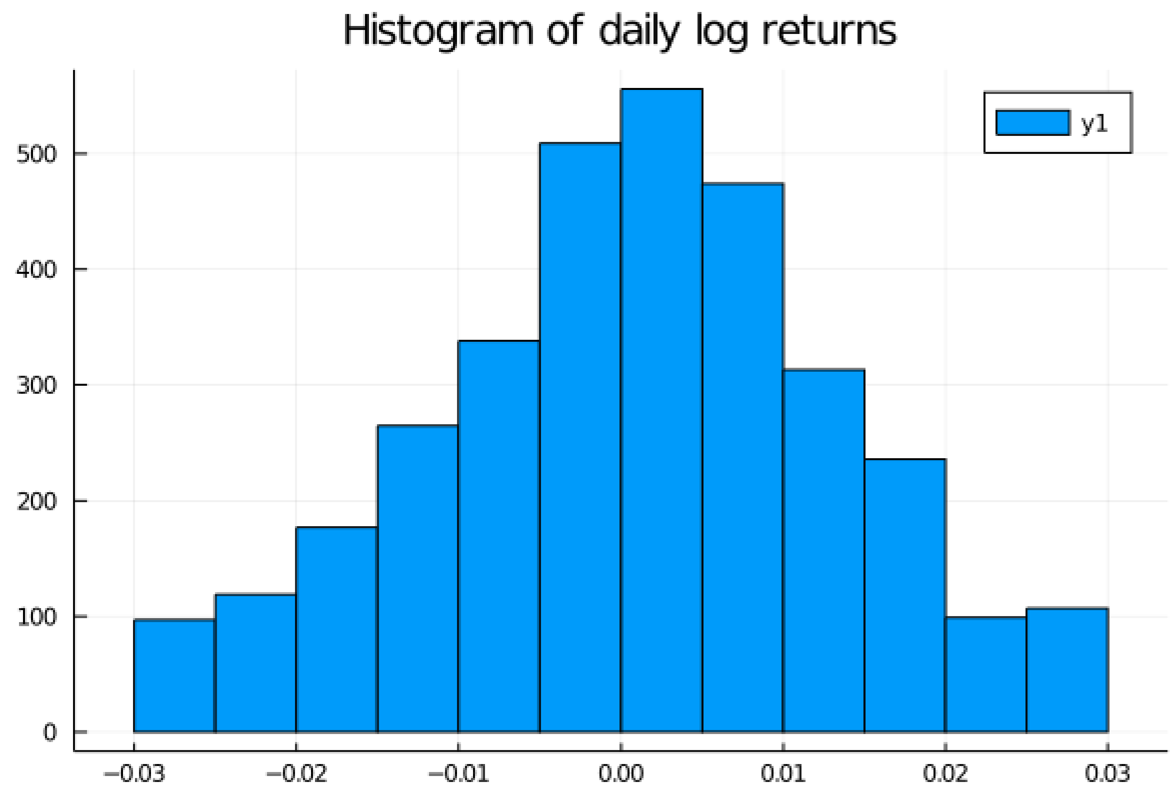
In [11]:
```
# Make Lag matrix of log volumes
global lag_vol=zeros(length(data[:,1])-10,0)
for i=1:10
global lag_vol
    lag_vol=[lag_vol data[i:(end-11+i),2]];
end
vol=data[11:end,2];
```

In [12]:
```
plot(visa.AdjClose,legend=:topleft,show=true,fmt=png, title = "Plot of Visa adjus
```
Out[12]:



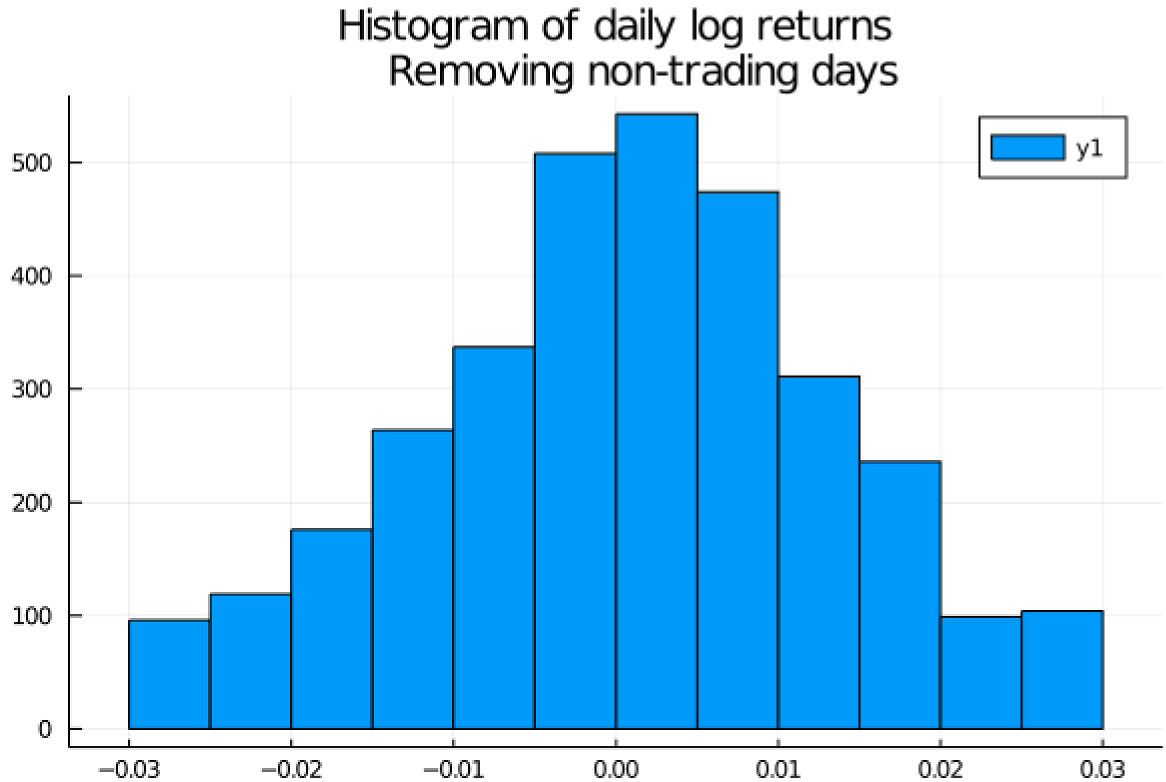
In [13]:
```
histogram(lrt,fmt=png, title = "Histogram of daily log returns")
```

Out[13]:



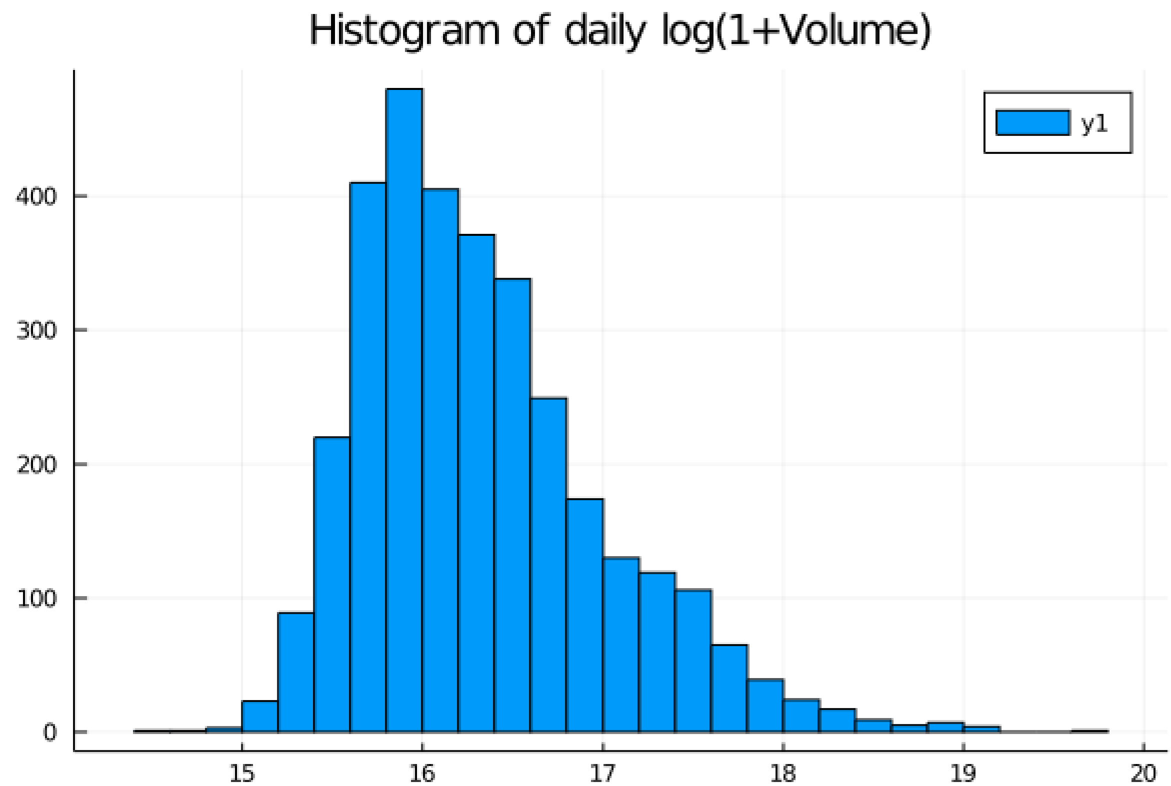
In [14]:
```
histogram(rets,fmt=png, title = "Histogram of daily log returns
    Removing non-trading days")
```

Out[14]:



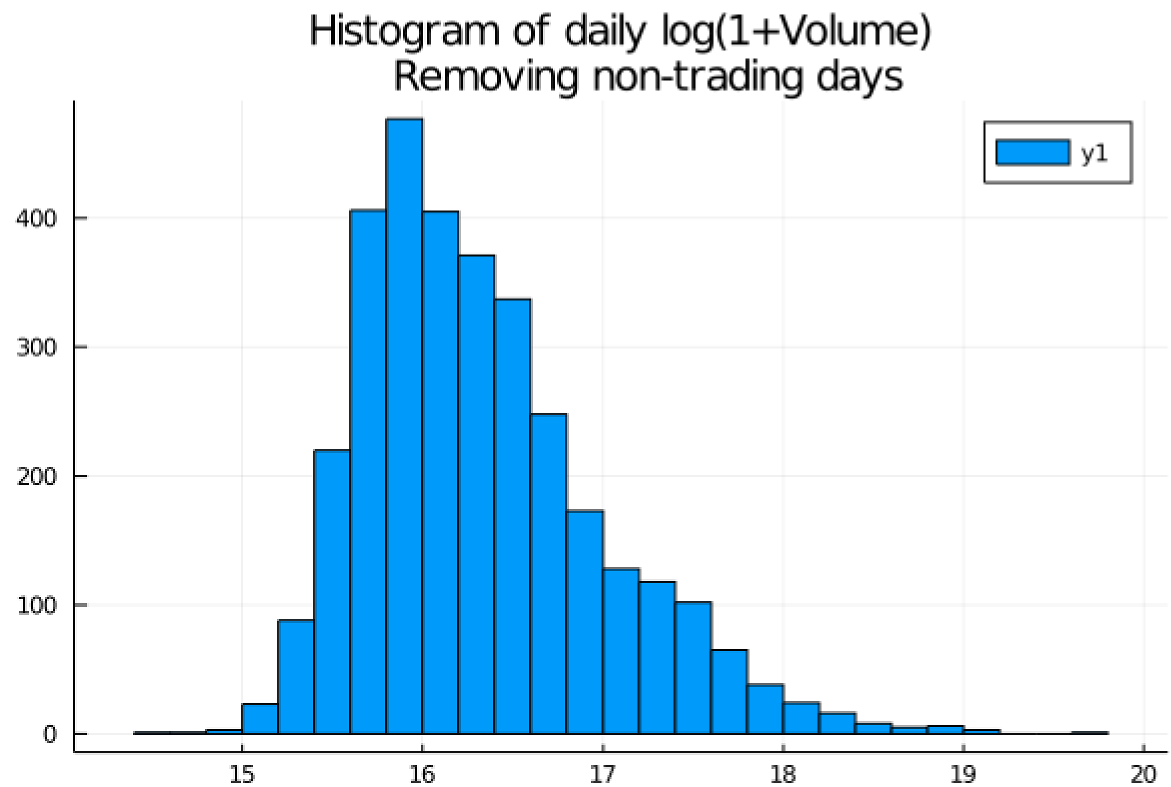
In [15]:
```
histogram(log_vol,fmt=png, title = "Histogram of daily log(1+Volume)")
```

Out[15]:



In [16]:
```
histogram(vol,fmt=png, title = "Histogram of daily log(1+Volume)
    Removing non-trading days")
```

Out[16]:



In [17]:
```
# 3621x20 lagged returns and volumes
x=hcat(lag_rets,lag_vol);
```

In [18]:
```
# Convert up day and down day to +/- 1's
adjclose_up = 2*((rets).>0).-1;
adjclose_up=adjclose_up[:,1];
```

In [19]:
```
all_data = hcat(rets, vol, adjclose_up, lag_rets, lag_vol);
```

Step 3

Testing and 5 Fold Training Dataset Preparation

Procedure

1. Remove last year as testing final stacked ensemble
2. Generate 5 random folds, and get id's/indexes for each in returns dataset
3. Make them equal size by reducing them to the length of the minimum length of the arrays

In [20]:
```
#Split the data into testing (training and validation) and testing (for testing e
data_train = all_data[1:end-252,:]
data_test = all_data[end-251:end,:];
```

In [21]:
```
# Split up data into folds
function perclass_splits(y,at)
    uids = unique(y)
    keepids = []
    for ui in uids
        curids = findall(y.==ui)
        rowids = randsubseq(curids, at)
        push!(keepids,rowids...)
    end
    return keepids
end
```

Out[21]: perclass_splits (generic function with 1 method)

In [22]:
```
Random.seed!(1)
fold1ids = perclass_splits(data_train[:,1],0.2)
fold2ids = perclass_splits(data_train[:,1],0.2)
fold3ids = perclass_splits(data_train[:,1],0.2)
fold4ids = perclass_splits(data_train[:,1],0.2)
fold5ids = perclass_splits(data_train[:,1],0.2);
```

In [23]:
```
lids=min(length(fold1ids), length(fold2ids),length(fold3ids),length(fold4ids),len
```

Out[23]: 568

In [24]:
```
fold1ids=fold1ids[1:lids]
fold2ids=fold2ids[1:lids]
fold3ids=fold3ids[1:lids]
fold4ids=fold4ids[1:lids]
fold5ids=fold5ids[1:lids];
```

In [25]:
```
foldids = vcat(fold1ids,fold2ids,fold3ids,fold4ids,fold5ids);
```

Model 1 - Lasso Regression

In [26]:
```
using GLMNet
using RDatasets
using MLBase
using DecisionTree
using Distances
using NearestNeighbors
using Random
using LinearAlgebra
using DataStructures
using LIBSVM
using CSV
using DataFrames
using Statistics
using GLM
using Distributions
```

In [27]:
```
findaccuracy(predictedvals,groundtruthvals) = sum(predictedvals.==groundtruthvals
```

Out[27]:
```
findaccuracy (generic function with 1 method)
```

In [28]:
```
# Fold1 testing data predictions
test1=vcat(fold2ids,fold3ids,fold4ids,fold5ids);
path1 = glmnet(data_train[test1,4:end], data_train[test1,1])
cv1 = glmnetcv(data_train[test1,4:end], data_train[test1,1],nfolds = 5)
# choose the best lambda to predict with.
path1 = glmnet(data_train[test1,4:end], data_train[test1,1])
mylambda1 = path1.lambda[argmin(cv1.meanloss)]
path1 = glmnet(data_train[test1,4:end], data_train[test1,1],lambda=[mylambda1])
predictions_lasso1 = GLMNet.predict(path1,data_train[fold1ids,4:end]);
```

In [29]:
```
# Calculate the MSE on the first fold
mean((predictions_lasso1-data_train[fold1ids,1]).^2)
```

Out[29]:
```
0.00015209858641858164
```

In [30]:
```
# Fold2 testing data predictions
test2=vcat(fold1ids,fold3ids,fold4ids,fold5ids);
path2 = glmnet(data_train[test2,4:end], data_train[test2,1])
cv2 = glmnetcv(data_train[test2,4:end], data_train[test2,1],nfolds = 5)
# choose the best lambda to predict with.
path2 = glmnet(data_train[test2,4:end], data_train[test2,1])
mylambda2 = path2.lambda[argmin(cv2.meanloss)]
path2 = glmnet(data_train[test2,4:end], data_train[test2,1],lambda=[mylambda2])
predictions_lasso2 = GLMNet.predict(path2,data_train[fold2ids,4:end]);
```

In [31]:
```
mean((predictions_lasso2-data_train[fold2ids,1]).^2)
```

Out[31]:
```
0.00015451032090438532
```

In [32]:
```
# Fold3 testing data predictions
test3=vcat(fold1ids,fold2ids,fold4ids,fold5ids);
path3 = glmnet(data_train[test3,4:end], data_train[test3,1])
cv3 = glmnetcv(data_train[test3,4:end], data_train[test3,1],nfolds = 5)
# choose the best lambda to predict with.
path3 = glmnet(data_train[test3,4:end], data_train[test3,1])
mylambda3 = path3.lambda[argmin(cv3.meanloss)]
path3 = glmnet(data_train[test3,4:end], data_train[test3,1],lambda=[mylambda3])
predictions_lasso3 = GLMNet.predict(path3,data_train[fold3ids,4:end]);
```

In [33]:
```
mean((predictions_lasso3-data_train[fold3ids,1]).^2)
```

Out[33]: 0.00015929870998535622

In [34]:
```
# Fold4 testing data predictions
test4=vcat(fold1ids,fold2ids,fold3ids,fold5ids);
path4 = glmnet(data_train[test4,4:end], data_train[test4,1])
cv4 = glmnetcv(data_train[test4,4:end], data_train[test4,1],nfolds = 5)
# choose the best lambda to predict with.
path4 = glmnet(data_train[test4,4:end], data_train[test4,1])
mylambda4 = path4.lambda[argmin(cv4.meanloss)]
path4 = glmnet(data_train[test4,4:end], data_train[test4,1],lambda=[mylambda4])
predictions_lasso4 = GLMNet.predict(path4,data_train[fold4ids,4:end]);
```

In [35]:
```
mean((predictions_lasso4-data_train[fold4ids,1]).^2)
```

Out[35]: 0.00016856151319722927

In [36]:
```
# Fold5 testing data predictions
test5=vcat(fold1ids,fold2ids,fold3ids,fold4ids);
path5 = glmnet(data_train[test5,4:end], data_train[test5,1])
cv5 = glmnetcv(data_train[test5,4:end], data_train[test5,1],nfolds = 5)
# choose the best lambda to predict with.
path5 = glmnet(data_train[test5,4:end], data_train[test5,1])
mylambda5 = path5.lambda[argmin(cv5.meanloss)]
path5 = glmnet(data_train[test5,4:end], data_train[test5,1],lambda=[mylambda5])
predictions_lasso5 = GLMNet.predict(path5,data_train[fold5ids,4:end]);
```

In [37]:
```
mean((predictions_lasso5-data_train[fold5ids,1]).^2)
```

Out[37]: 0.00016305012621257225

In [38]:
```
# Collate forecasts
lasso_forecasts = vcat(predictions_lasso1,predictions_lasso2,predictions_lasso3,p
```

In [39]:
```
#Convert to binary output for stack
lasso_ensemble=ones(length(lasso_forecasts))
# Get directions for our random dataset
for i=1:length(lasso_ensemble)
    if lasso_forecasts[i]>0
        lasso_ensemble[i] = 1
    else
        lasso_ensemble[i] = -1
    end
end
```

In [40]:
```
# Calculate the prediction accuracy cross validation
mean(lasso_ensemble.==data_train[foldids,3])
```

Out[40]: 0.5433098591549296

Fit Model on Whole Training Sample - Lasso Regression

In [41]:
```
# Fit training data model
pathx = glmnet(data_train[:,4:end], data_train[:,1])
cv = glmnetcv(data_train[:,4:end], data_train[:,1],nfolds = 5)
# choose the best lambda to predict with.
pathx = glmnet(data_train[:,4:end], data_train[:,1])
mylambda = pathx.lambda[argmin(cv.meanloss)]
pathx = glmnet(data_train[:,4:end], data_train[:,1],lambda=[mylambda])
```

Out[41]: Least Squares GLMNet Solution Path (1 solutions for 20 predictors in 9 passes):

	df	pct_dev	λ
[1]	7	0.0155001	0.000162982

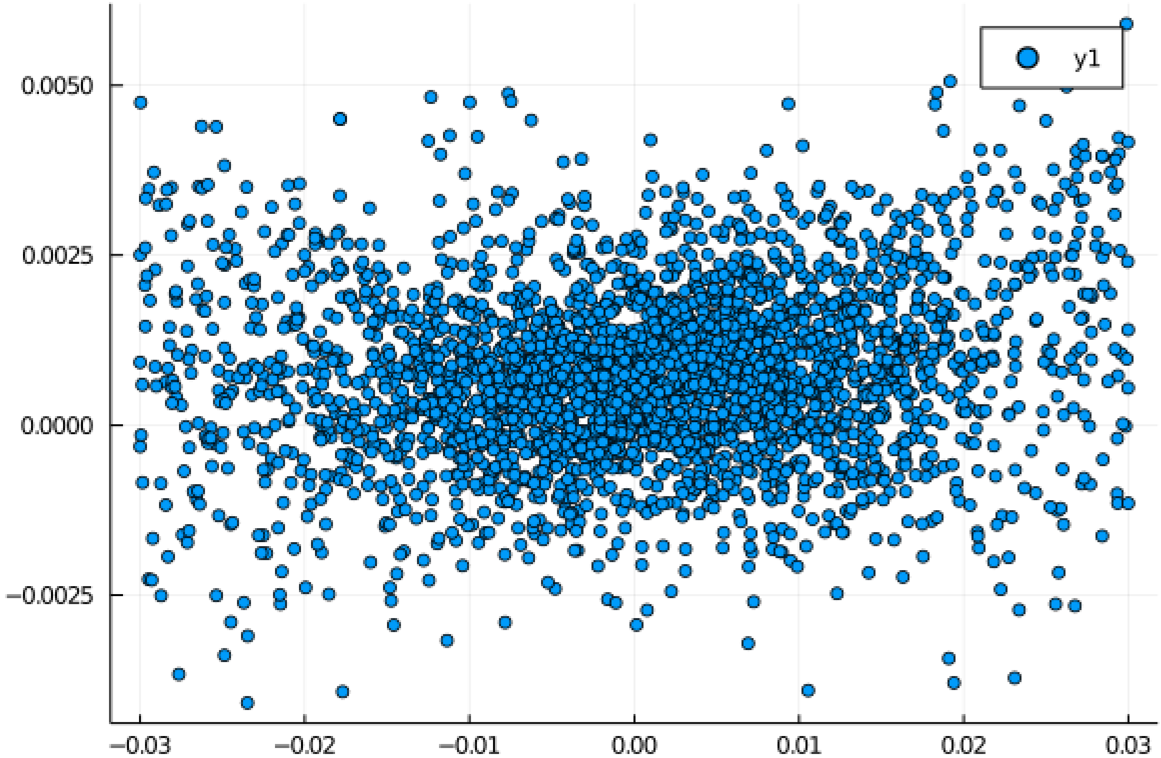
In [42]:
```
predictions_lasso_whole = GLMNet.predict(pathx,data_train[:,4:end]);
```

In [43]:
```
# Calculate the MSE on the whole train set
mean((predictions_lasso_whole-data_train[:,1]).^2)
```

Out[43]: 0.00015800539914409575

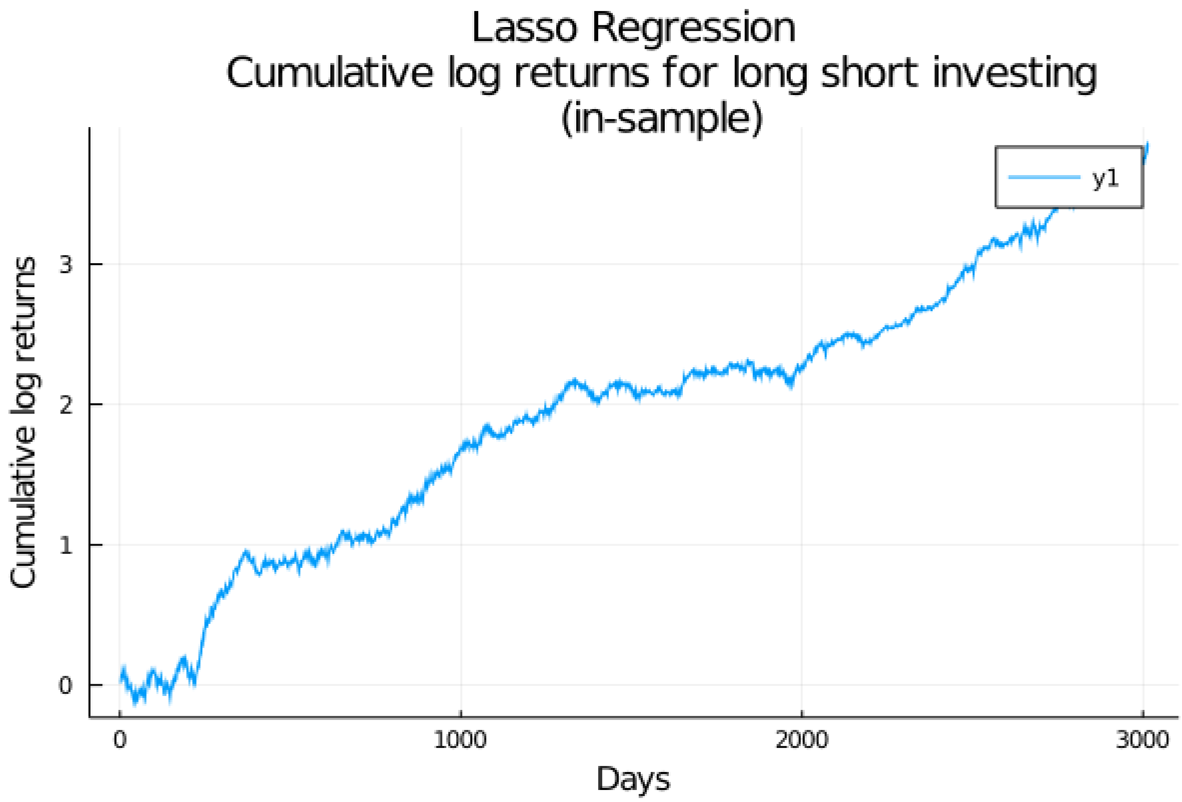
In [44]:
```
# Scatter plot of actual vs predicted
Plots.scatter(data_train[:,1],predictions_lasso_whole,fmt=png)
```

Out[44]:



In [45]:
```
plot(cumsum(data_train[:,1].*sign.(predictions_lasso_whole.-0),dims=1),fmt=png, t
    Cumulative log returns for long short investing
    (in-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[45]:



In [46]:
```
# calculate in-sample sharpe
sum_lasso_in_sample = sum(sign.(predictions_lasso_whole.-0).*data_train[:,1])/(30
sd_lasso_in_sample = std(sign.(predictions_lasso_whole.-0).*data_train[:,1])
sum_lasso_in_sample/(sd_lasso_in_sample*(252^0.5))
```

Out[46]:
```
1.6085700452357552
```

In [47]:
```
binary_predictions_lasso_whole = ones(length(data_train[:,1]));
```

In [48]:
```
# Convert to binary classifier output
for i=1:length(data_train[:,1])
    if predictions_lasso_whole[i,1]>0
        binary_predictions_lasso_whole[i] = 1
    else
        binary_predictions_lasso_whole[i] = -1
    end
end
```

In [49]:
```
# GET ACCURACY FOR MODEL WITH BINARY OUTPUT
findaccuracy(binary_predictions_lasso_whole,data_train[:,3])
```

Out[49]:
```
0.560530679933665
```

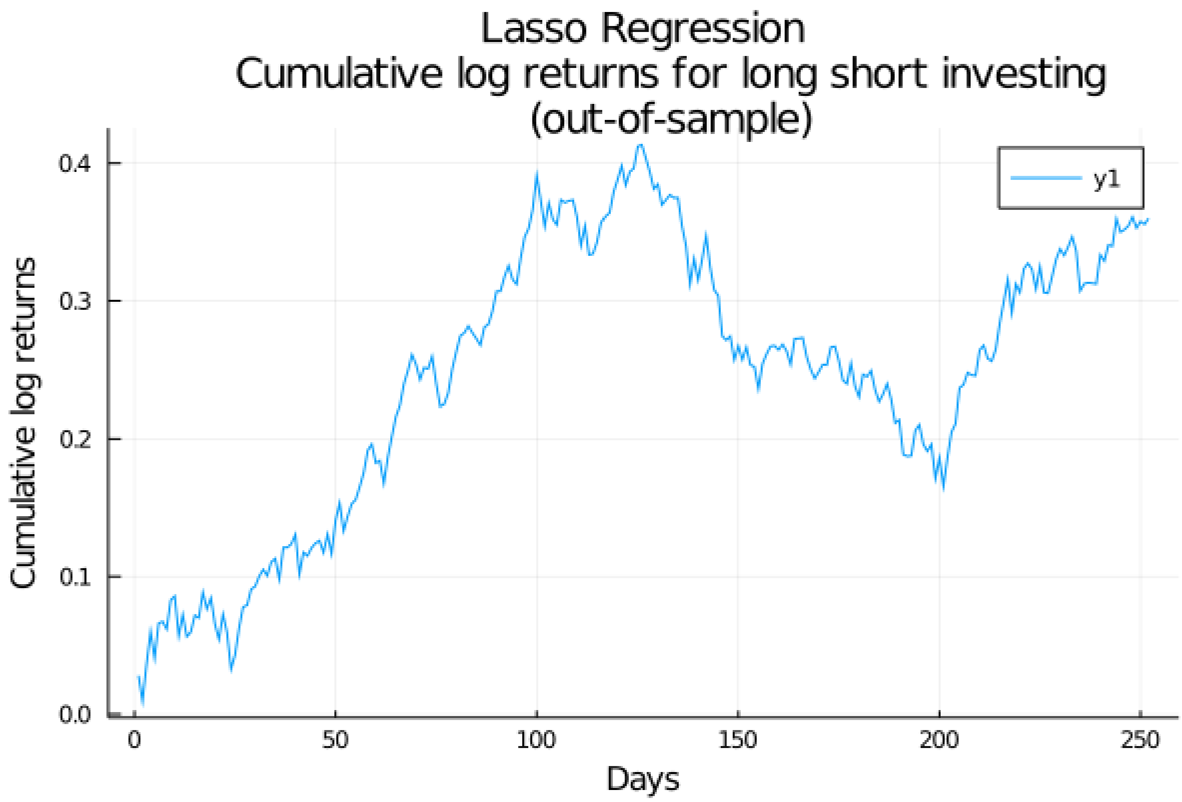
In [593]:
```
# Calculate Returns
sum(data_train[:,1].*sign.(predictions_lasso_whole.-0),dims=1) * (252/3015)
```

Out[593]:
```
1×1 Array{Float64,2}:
 0.3224650426611486
```

Test sample - Lasso regression

In [50]:
```
# Make predictions on test data
predictions_lasso_test = GLMNet.predict(pathx,data_test[:,4:end]);

plot(cumsum(data_test[:,1].*sign.(predictions_lasso_test.-0),dims=1),fmt=png, tit
    Cumulative log returns for long short investing
    (out-of-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[50]:



In [51]:
```
# Calculate out of sample sharpe
sum_lasso_out_of_sample = sum(sign.(predictions_lasso_test.-0).*data_test[:,1])
sd_lasso_out_of_sample = std(sign.(predictions_lasso_test.-0).*data_test[:,1])
sum_lasso_out_of_sample/(sd_lasso_out_of_sample*(252^0.5))
```

Out[51]:
```
1.7572526599096443
```

In [577]:
```
# Calculate Out-of-sample returns
sum(data_test[:,1].*sign.(predictions_lasso_test.-0),dims=1)
```

Out[577]:
```
1×1 Array{Float64,2}:
 0.3599508706498156
```

Model 2 - Logistic Regression

In [52]:
```
data1=convert(DataFrame, data_train[test1,:])
data2=convert(DataFrame, data_train[test2,:])
data3=convert(DataFrame, data_train[test3,:])
data4=convert(DataFrame, data_train[test4,:])
data5=convert(DataFrame, data_train[test5,:]);
```

In [53]:
```
positive1 = data1[:,1].>0
data1 = hcat(data1,positive1, makeunique=true)

positive2 = data2[:,1].>0
data2 = hcat(data2,positive2, makeunique=true)

positive3 = data3[:,1].>0
data3 = hcat(data3,positive3, makeunique=true)

positive4 = data4[:,1].>0
data4 = hcat(data4,positive4, makeunique=true)

positive5 = data5[:,1].>0
data5 = hcat(data5,positive5, makeunique=true);
```

In [54]:
```
glm_fit1=glm(@formula(x1_1 ~ x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x1
glm_fit2=glm(@formula(x1_1 ~ x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x1
glm_fit3=glm(@formula(x1_1 ~ x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x1
glm_fit4=glm(@formula(x1_1 ~ x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x1
glm_fit5=glm(@formula(x1_1 ~ x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x1
```

In [55]:
```
test1=convert(DataFrame, data_train[fold1ids,:]);
test2=convert(DataFrame, data_train[fold2ids,:]);
test3=convert(DataFrame, data_train[fold3ids,:]);
test4=convert(DataFrame, data_train[fold4ids,:]);
test5=convert(DataFrame, data_train[fold5ids,:]);
```

In [56]:
```
glm_probs1 =GLM.predict(glm_fit1,test1)
glm_probs2 =GLM.predict(glm_fit2,test2)
glm_probs3 =GLM.predict(glm_fit3,test3)
glm_probs4 =GLM.predict(glm_fit4,test4)
glm_probs5 =GLM.predict(glm_fit5,test5);
```

In [57]:
```
Up1 = glm_probs1.>0.5
Up2 = glm_probs2.>0.5
Up3= glm_probs3.>0.5
Up4 = glm_probs4.>0.5
Up5 = glm_probs5.>0.5;
```

In [58]:
```
un = ones(length(glm_probs1));
deux = ones(length(glm_probs1));
trois= ones(length(glm_probs1));
quatre = ones(length(glm_probs1));
cinq = ones(length(glm_probs1));
```

In [59]:
```
for i=1:length(glm_probs1)
    if glm_probs1[i] > 0.5
        un[i] = 1
    else
        un[i] = -1
    end
end
```

In [60]:
```
for i=1:length(glm_probs2)
    if glm_probs2[i] > 0.5
        deux[i] = 1
    else
        deux[i] = -1
    end
end
```

In [61]:
```
for i=1:length(glm_probs3)
    if glm_probs3[i] > 0.5
        trois[i] = 1
    else
        trois[i] = -1
    end
end
```

In [62]:
```
for i=1:length(glm_probs4)
    if glm_probs4[i] > 0.5
        quatre[i] = 1
    else
        quatre[i] = -1
    end
end
```

In [63]:
```
for i=1:length(glm_probs5)
    if glm_probs5[i] > 0.5
        cinq[i] = 1
    else
        cinq[i] = -1
    end
end
```

In [64]:
```
m1_accurary = findaccuracy(un,test1[:,3])
```

Out[64]:
```
0.5580985915492958
```

In [65]:
```
m2_accurary = findaccuracy(deux,test2[:,3])
```

Out[65]:
```
0.5545774647887324
```

In [66]:
```
m3_accurary = findaccuracy(trois,test3[:,3])
```

Out[66]:
```
0.5545774647887324
```

In [67]:
```
m4_accurary = findaccuracy(quatre,test4[:,3])
```

Out[67]:
```
0.545774647887324
```

In [68]:
```
m5_accurary = findaccuracy(cinq,test5[:,3])
```

Out[68]:
```
0.5316901408450704
```

In [69]:
```
logistic_ensemble=vcat(un, deux, trois, quatre, cinq);
```

In [70]:
```
# Calculate the prediction accuracy cross validation
mean(logistic_ensemble.==data_train[foldids,3])
```

Out[70]:
```
0.548943661971831
```

Fit Model on Whole Training Sample - Logistic Regression

In [71]:
```
# Logistic Regression Model
data_comp=convert(DataFrame, data_train);
positive = data_comp[:,1].>0
data_comp = hcat(data_comp,positive, makeunique=true)
glm_fitx=glm(@formula(x1_1 ~ x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x1
df_test=convert(DataFrame, data_test);
glm_probsx =GLM.predict(glm_fitx,df_test)
Upx = glm_probsx.>0.5;
fin = ones(length(glm_probsx));
for i=1:length(glm_probsx)
    if glm_probsx[i] > 0.5
        fin[i] = 1
    else
        fin[i] = -1
    end
end
```

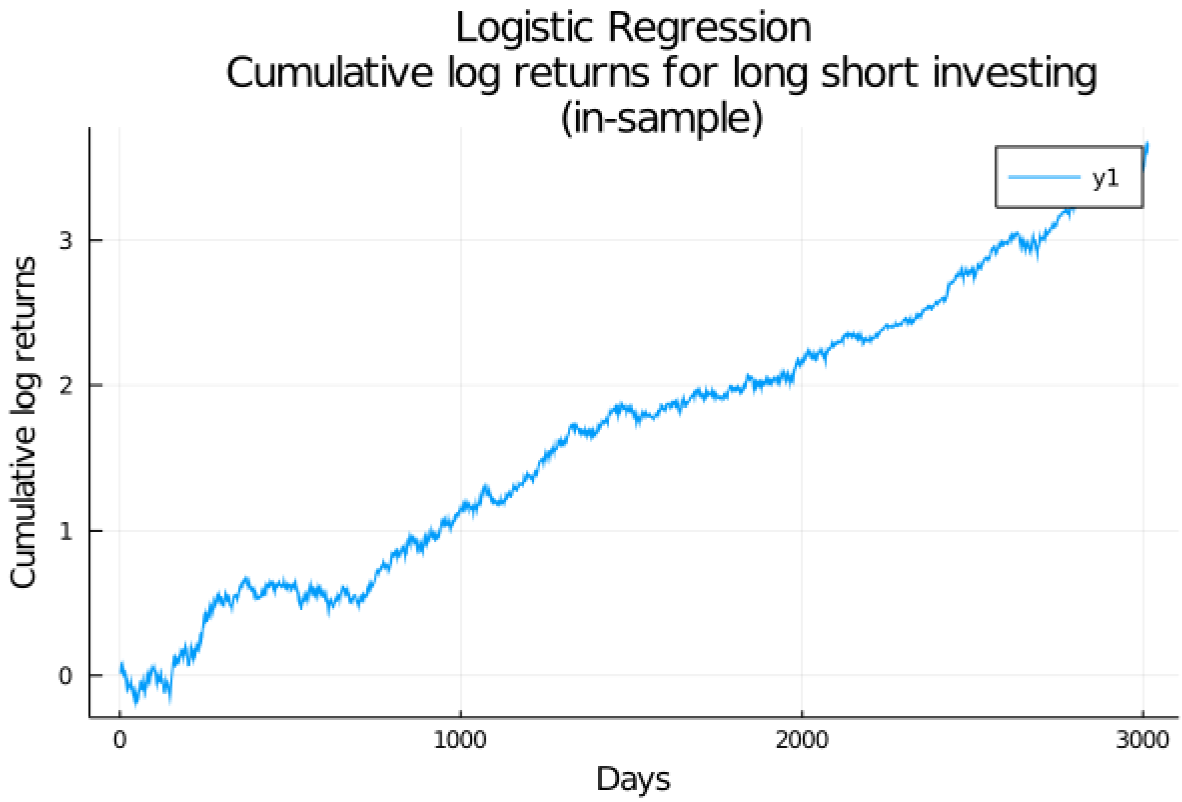
In [72]:
```
# make predictions on training data using model
df_train=convert(DataFrame, data_train);
glm_probsx_train =GLM.predict(glm_fitx,df_train)
Upx_train = glm_probsx_train.>0.5;
fin_train = ones(length(glm_probsx_train));
for i=1:length(glm_probsx_train)
    if glm_probsx_train[i] > 0.5
        fin_train[i] = 1
    else
        fin_train[i] = -1
    end
end
```

In [73]:
```
# Calculate the prediction accuracy on the whole train set
mean(fin_train.==data_train[:,3])
```

Out[73]:
```
0.560530679933665
```

In [74]:
```
plot(cumsum(data_train[:,1].*fin_train),fmt=png, title = "Logistic Regression
    Cumulative log returns for long short investing
    (in-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[74]:



In [75]:
```
# calculate in-sample sharpe
sum_logistic_in_sample = sum(data_train[:,1].*fin_train)/(3015/252)
sd_logistic_in_sample = std(data_train[:,1].*fin_train)
sum_logistic_in_sample/(sd_logistic_in_sample*(252^0.5))
```

Out[75]: 1.5279989121499595

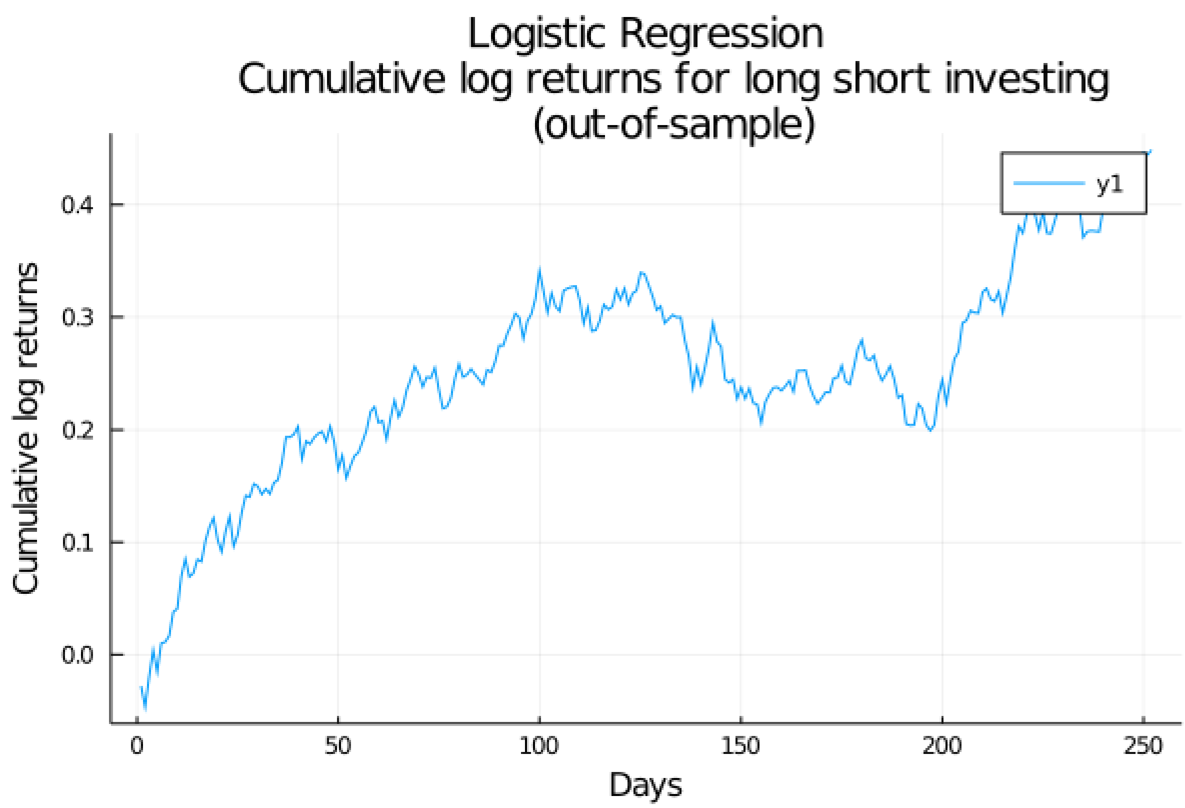
In [594]:
```
# Calculate in sample returns
sum(data_train[:,1].*fin_train) * (252/3015)
```

Out[594]: 0.30646539233248127

Test Sample - Logistic Regression

In [76]:
```
# Make predictions on test set
df_test=convert(DataFrame, data_test);
glm_probsx_test =GLM.predict(glm_fitx,df_test)
Upx_test = glm_probsx_test.>0.5;
fin_test = ones(length(glm_probsx_test));
for i=1:length(glm_probsx_test)
    if glm_probsx_test[i] > 0.5
        fin_test[i] = 1
    else
        fin_test[i] = -1
    end
end

# Plot
plot(cumsum(data_test[:,1].*fin_test),fmt=png, title = "Logistic Regression
    Cumulative log returns for long short investing
    (out-of-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[76]:



In [77]:
```
# Calculate out of sample sharpe
sum_logistic_out_of_sample = sum(data_test[:,1].*fin_test)
sd_logistic_out_of_sample = std(data_test[:,1].*fin_test)
sum_logistic_out_of_sample/(sd_logistic_out_of_sample*(252^0.5))
```

Out[77]: 2.1979868259430813

In [579]:
```
# Calculate out of sample returns
sum(data_test[:,1].*fin_test)
```

Out[579]: 0.448693129276095

Model 3 - Support Vector Machine

In [78]:
```
using DataFrames
using RDatasets
using GLM
using Plots
using Statistics
using LIBSVM
```

In [79]:
```
# Fold1 testing data predictions
test1=vcat(fold2ids,fold3ids,fold4ids,fold5ids);

# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test1,4:end])
up_vec_train = data_train[test1,3];

variables_test = Array(data_train[fold1ids,4:end])
up_vec_test = data_train[fold1ids,3];

# train  SV model remembering that input data need to be transposed
sv_model1 = svmtrain(variables_train', up_vec_train);
sv_preds1,sv_info1=sv_predict=svmpredict(sv_model1,variables_test');
sv_preds1=vec(sv_preds1); # Predictions
sv_scores1=vec(sv_info1[1,:]); # Scores (on which Predictions are based)

# Calculate predictive accuracy
pred_acc1 = mean(sv_preds1.==up_vec_test)
```

Out[79]: 0.5598591549295775

In [80]:
```
# Fold2 testing data predictions
test2=vcat(fold1ids,fold3ids,fold4ids,fold5ids);

# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test2,4:end])
up_vec_train = data_train[test2,3];

variables_test = Array(data_train[fold2ids,4:end])
up_vec_test = data_train[fold2ids,3];

# train  SV model remembering that input data need to be transposed
sv_model2 = svmtrain(variables_train', up_vec_train);
sv_preds2,sv_info2=sv_predict=svmpredict(sv_model2,variables_test');
sv_preds2=vec(sv_preds2); # Predictions
sv_scores2=vec(sv_info2[1,:]); # Scores (on which Predictions are based)

# Calculate predictive accuracy
pred_acc2 = mean(sv_preds2.==up_vec_test)
```

Out[80]:
```
0.5580985915492958
```

In [81]:
```
# Fold3 testing data predictions
test3=vcat(fold1ids,fold2ids,fold4ids,fold5ids);

# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test3,4:end])
up_vec_train = data_train[test3,3];

variables_test = Array(data_train[fold3ids,4:end])
up_vec_test = data_train[fold3ids,3];

# train  SV model remembering that input data need to be transposed
sv_model3 = svmtrain(variables_train', up_vec_train);
sv_preds3,sv_info3=sv_predict=svmpredict(sv_model3,variables_test');
sv_preds3=vec(sv_preds3); # Predictions
sv_scores3=vec(sv_info3[1,:]); # Scores (on which Predictions are based)

# Calculate predictive accuracy
pred_acc3 = mean(sv_preds3.==up_vec_test)
```

Out[81]:
```
0.528169014084507
```

In [82]:
```
# Fold4 testing data predictions
test4=vcat(fold1ids,fold2ids,fold3ids,fold5ids);

# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test4,4:end])
up_vec_train = data_train[test4,3];

variables_test = Array(data_train[fold4ids,4:end])
up_vec_test = data_train[fold4ids,3];

# train  SV model remembering that input data need to be transposed
sv_model4 = svmtrain(variables_train', up_vec_train);
sv_preds4,sv_info4=sv_predict=svmpredict(sv_model4,variables_test');
sv_preds4=vec(sv_preds4); # Predictions
sv_scores4=vec(sv_info4[1,:]); # Scores (on which Predictions are based)

# Calculate predictive accuracy
pred_acc4 = mean(sv_preds4.==up_vec_test)
```

Out[82]:
```
0.5528169014084507
```

In [83]:
```
# Fold5 testing data predictions
test5=vcat(fold1ids,fold2ids,fold3ids,fold4ids);

# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test5,4:end])
up_vec_train = data_train[test5,3];

variables_test = Array(data_train[fold5ids,4:end])
up_vec_test = data_train[fold5ids,3];

# train  SV model remembering that input data need to be transposed
sv_model5 = svmtrain(variables_train', up_vec_train);
sv_preds5,sv_info5=sv_predict=svmpredict(sv_model5,variables_test');
sv_preds5=vec(sv_preds5); # Predictions
sv_scores5=vec(sv_info5[1,:]); # Scores (on which Predictions are based)

# Calculate predictive accuracy
pred_acc5 = mean(sv_preds5.==up_vec_test)
```

Out[83]:
```
0.5404929577464789
```

In [84]:
```
svm_ensemble=vcat(sv_preds1,sv_preds2,sv_preds3,sv_preds4,sv_preds5);
```

In [85]:
```
# Calculate the prediction accuracy cross validation
mean(svm_ensemble.==data_train[foldids,3])
```

Out[85]:
```
0.547887323943662
```

Fit Model on Whole Training Sample - Support Vector Machine

In [86]:
```
# Support Vector Machine
# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[:,4:end])
up_vec_train = data_train[:,3];

variables_test = Array(data_test[:,4:end])
up_vec_test = data_test[:,3];

# train  SV model remembering that input data need to be transposed
sv_model = svmtrain(variables_train', up_vec_train);
sv_preds,sv_info=sv_predict=svmpredict(sv_model,variables_test');
sv_preds=vec(sv_preds); # Predictions
sv_scores=vec(sv_info[1,:]); # Scores (on which Predictions are based)

# Calculate predictive accuracy
svm_pred_acc = mean(sv_preds.==up_vec_test)
```

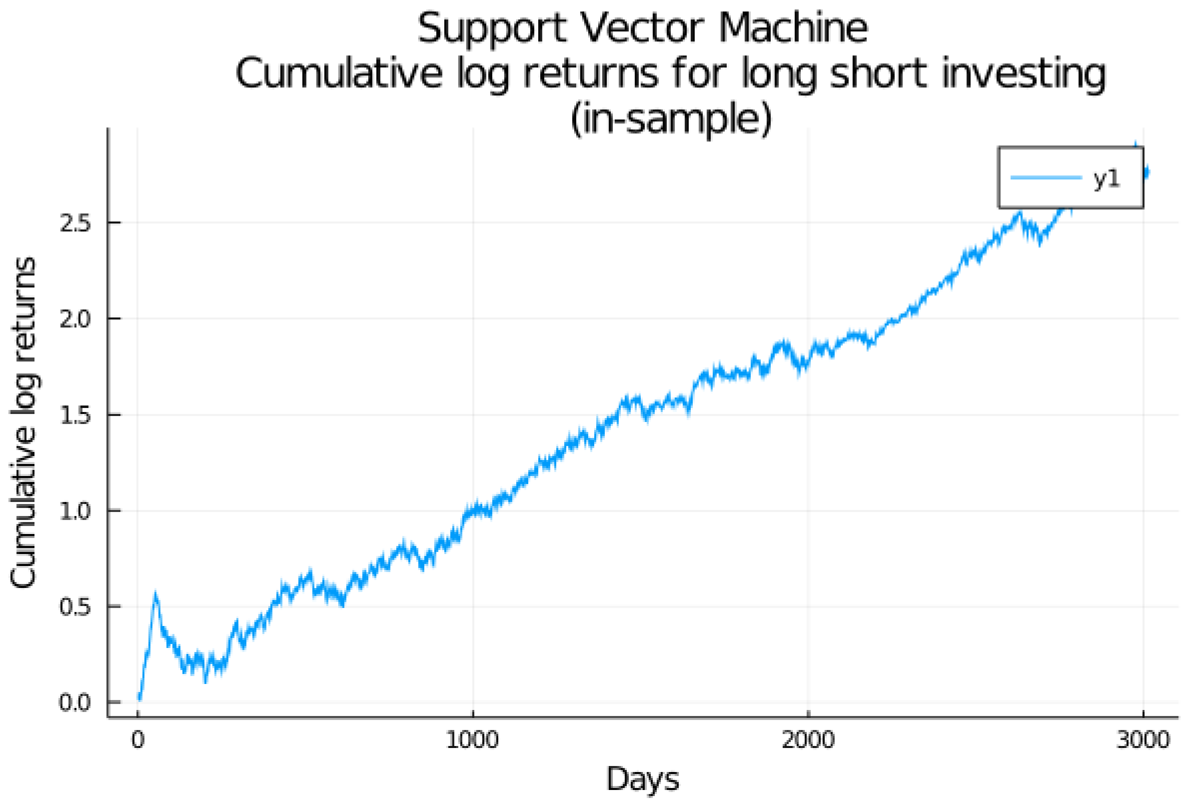
Out[86]: 0.5396825396825397

In [87]:
```
# make predictions on training data using model
sv_preds_train,sv_info_train=sv_predict_train=svmpredict(sv_model,variables_trair
sv_preds_train=vec(sv_preds_train); # Predictions
sv_scores_train=vec(sv_info_train[1,:]); # Scores (on which Predictions are basec

# Calculate predictive accuracy
svm_pred_acc_train = mean(sv_preds_train.==up_vec_train)
```

Out[87]: 0.5466003316749586

In [88]:
```
# Plot of Cumulative log returns of training data
plot(cumsum(data_train[:,1].*sv_preds_train),fmt=png, title = "Support Vector Mac
    Cumulative log returns for long short investing
    (in-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[88]:



In [89]:
```
# calculate in-sample sharpe
sum_svm_in_sample = sum(data_train[:,1].*sv_preds_train)/(3015/252)
sd_svm_in_sample = std(data_train[:,1].*sv_preds_train)
sum_svm_in_sample/(sd_svm_in_sample*(252^0.5))
```

Out[89]: 1.1541384196388123

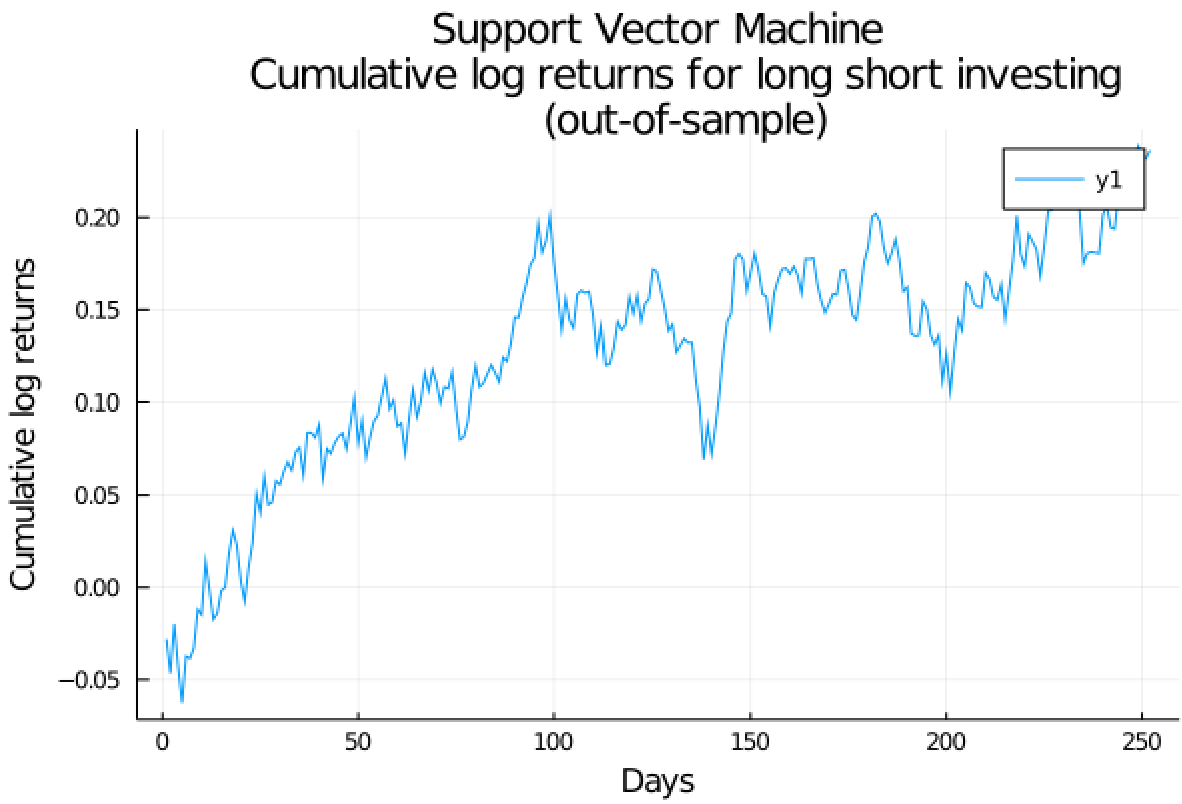
In [595]:
```
# calaculate in sample returns
sum(data_train[:,1].*sv_preds_train) * (252/3015)
```

Out[595]: 0.23193932860633235

Test Sample - Support Vector Machine

In [90]:
```
# make predictions on testing data using model
sv_preds_test,sv_info_test=sv_predict_test=svmpredict(sv_model,variables_test');
sv_preds_test=vec(sv_preds_test); # Predictions
sv_scores_test=vec(sv_info_test[1,:]); # Scores (on which Predictions are based)

# Plot
plot(cumsum(data_test[:,1].*sv_preds_test),fmt=png, title = "Support Vector Machi
    Cumulative log returns for long short investing
    (out-of-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[90]:



In [91]:
```
# calculate out-of-sample sharpe
sum_svm_out_of_sample = sum(data_test[:,1].*sv_preds_test)
sd_svm_out_of_sample = std(data_test[:,1].*sv_preds_test)
sum_svm_out_of_sample/(sd_svm_out_of_sample*(252^0.5))
```

Out[91]: 1.149858485499543

In [581]:
```
# calculate out of sample returns
sum(data_test[:,1].*sv_preds_test)
```

Out[581]: 0.23635659649735793

Model 4 - Gradient Boosting

```
In [92]: # Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test1,4:end])
ret_train = data_train[test1,1];
up_vec_train = data_train[test1,3];

variables_test = Array(data_train[fold1ids,4:end])
ret_test = data_train[fold1ids,1];
up_vec_test = data_train[fold1ids,3];

#  This saves memory and makes computation much faster!!!
variables_train=convert.(Float32,variables_train);
ret_train=convert.(Float32,ret_train);
variables_test=convert.(Float32,variables_test);
ret_test=convert.(Float32,ret_test);

# Create an xgboost on the training dataset
dtrain1 = DMatrix(variables_train, label = ret_train);
boost1 = xgboost(dtrain1, 100, eta = 0.1,max_depth=2);

# make predictions using model
yp1=XGBoost.predict(boost1,variables_test);

# Calculate MSE
mse1 = mean((yp1-ret_test).^2);

# Calculate predictive accuracy
bets1=sign.(yp1);
pred_acc1 = mean(bets1.==up_vec_test);
```

```
[1]     train-rmse:0.449667
[2]     train-rmse:0.404759
[3]     train-rmse:0.364343
[4]     train-rmse:0.327972
[5]     train-rmse:0.295242
[6]     train-rmse:0.265789
[7]     train-rmse:0.239286
[8]     train-rmse:0.215440
[9]     train-rmse:0.193985
[10]    train-rmse:0.174684
[11]    train-rmse:0.157322
[12]    train-rmse:0.141706
[13]    train-rmse:0.127663
[14]    train-rmse:0.115037
[15]    train-rmse:0.103687
[16]    train-rmse:0.093489
[17]    train-rmse:0.084328
[18]    train-rmse:0.076103
[19]    train-rmse:0.068721
```

```
In [93]: print(pred_acc1, mse1)
```

```
0.61443661971830990.00013685274
```

In [94]:
```
# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test2,4:end])
ret_train = data_train[test2,1];
up_vec_train = data_train[test2,3];

variables_test = Array(data_train[fold2ids,4:end])
ret_test = data_train[fold2ids,1];
up_vec_test = data_train[fold2ids,3];

#  This saves memory and makes computation much faster!!!
variables_train=convert.(Float32,variables_train);
ret_train=convert.(Float32,ret_train);
variables_test=convert.(Float32,variables_test);
ret_test=convert.(Float32,ret_test);

# Create an xgboost on the training dataset
dtrain2 = DMatrix(variables_train, label = ret_train)
boost2 = xgboost(dtrain2, 100, eta = 0.1,max_depth=2)

# make predictions using model
yp2=XGBoost.predict(boost2,variables_test);

# Calculate MSE
mse2 = mean((yp2-ret_test).^2);

# Calculate predictive accuracy
bets2=sign.(yp2);
pred_acc2 = mean(bets2.==up_vec_test);
```

```
[1]	train-rmse:0.449576
[2]	train-rmse:0.404677
[3]	train-rmse:0.364269
[4]	train-rmse:0.327906
[5]	train-rmse:0.295182
[6]	train-rmse:0.265735
[7]	train-rmse:0.239238
[8]	train-rmse:0.215397
[9]	train-rmse:0.193946
[10]	train-rmse:0.174649
[11]	train-rmse:0.157290
[12]	train-rmse:0.141677
[13]	train-rmse:0.127637
[14]	train-rmse:0.115014
[15]	train-rmse:0.103667
[16]	train-rmse:0.093470
[17]	train-rmse:0.084311
[18]	train-rmse:0.076087
[19]	train-rmse:0.068707
```

In [95]:
```
print(pred_acc2, mse2)
```

```
0.61267605633802810.00013839868
```

In [96]:
```
# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test3,4:end])
ret_train = data_train[test3,1];
up_vec_train = data_train[test3,3];

variables_test = Array(data_train[fold3ids,4:end])
ret_test = data_train[fold3ids,1];
up_vec_test = data_train[fold3ids,3];

#  This saves memory and makes computation much faster!!!
variables_train=convert.(Float32,variables_train);
ret_train=convert.(Float32,ret_train);
variables_test=convert.(Float32,variables_test);
ret_test=convert.(Float32,ret_test);

# Create an xgboost on the training dataset
dtrain3 = DMatrix(variables_train, label = ret_train)
boost3 = xgboost(dtrain3, 100, eta = 0.1,max_depth=2)

# make predictions using model
yp3=XGBoost.predict(boost3,variables_test);

# Calculate MSE
mse3 = mean((yp3-ret_test).^2);

# Calculate predictive accuracy
bets3=sign.(yp3);
pred_acc3 = mean(bets3.==up_vec_test);
```

```
[1]	train-rmse:0.449686
[2]	train-rmse:0.404775
[3]	train-rmse:0.364357
[4]	train-rmse:0.327984
[5]	train-rmse:0.295252
[6]	train-rmse:0.265798
[7]	train-rmse:0.239294
[8]	train-rmse:0.215446
[9]	train-rmse:0.193990
[10]	train-rmse:0.174688
[11]	train-rmse:0.157324
[12]	train-rmse:0.141707
[13]	train-rmse:0.127663
[14]	train-rmse:0.115036
[15]	train-rmse:0.103685
[16]	train-rmse:0.093485
[17]	train-rmse:0.084323
[18]	train-rmse:0.076097
[19]	train-rmse:0.068714
```

In [97]:
```
print(pred_acc3, mse3)
```

```
0.57570422535211260.00015031695
```

In [98]:
```
# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test4,4:end])
ret_train = data_train[test4,1];
up_vec_train = data_train[test4,3];

variables_test = Array(data_train[fold4ids,4:end])
ret_test = data_train[fold4ids,1];
up_vec_test = data_train[fold4ids,3];

#  This saves memory and makes computation much faster!!!
variables_train=convert.(Float32,variables_train);
ret_train=convert.(Float32,ret_train);
variables_test=convert.(Float32,variables_test);
ret_test=convert.(Float32,ret_test);

# Create an xgboost on the training dataset
dtrain4 = DMatrix(variables_train, label = ret_train)
boost4 = xgboost(dtrain4, 100, eta = 0.1,max_depth=2)

# make predictions using model
yp4=XGBoost.predict(boost4,variables_test);

# Calculate MSE
mse4 = mean((yp4-ret_test).^2);

# Calculate predictive accuracy
bets4=sign.(yp4);
pred_acc4 = mean(bets4.==up_vec_test);
```

```
[1]     train-rmse:0.449852
[2]     train-rmse:0.404924
[3]     train-rmse:0.364491
[4]     train-rmse:0.328104
[5]     train-rmse:0.295359
[6]     train-rmse:0.265893
[7]     train-rmse:0.239378
[8]     train-rmse:0.215521
[9]     train-rmse:0.194056
[10]    train-rmse:0.174746
[11]    train-rmse:0.157375
[12]    train-rmse:0.141751
[13]    train-rmse:0.127700
[14]    train-rmse:0.115067
[15]    train-rmse:0.103711
[16]    train-rmse:0.093506
[17]    train-rmse:0.084339
[18]    train-rmse:0.076108
[19]    train-rmse:0.068720
```

In [99]:
```
print(pred_acc4, mse4)
```

```
0.58274647887323940.00015346221
```

In [100]:

```
# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[test5,4:end])
ret_train = data_train[test5,1];
up_vec_train = data_train[test5,3];

variables_test = Array(data_train[fold4ids,4:end])
ret_test = data_train[fold4ids,1];
up_vec_test = data_train[fold4ids,3];

#  This saves memory and makes computation much faster!!!
variables_train=convert.(Float32,variables_train);
ret_train=convert.(Float32,ret_train);
variables_test=convert.(Float32,variables_test);
ret_test=convert.(Float32,ret_test);

# Create an xgboost on the training dataset
dtrain5 = DMatrix(variables_train, label = ret_train)
boost5 = xgboost(dtrain5, 100, eta = 0.1,max_depth=2)

# make predictions using model
yp5=XGBoost.predict(boost5,variables_test);

# Calculate MSE
mse5 = mean((yp5-ret_test).^2);

# Calculate predictive accuracy
bets5=sign.(yp5);
pred_acc5 = mean(bets5.==up_vec_test);
```

```
[1]     train-rmse:0.449620
[2]     train-rmse:0.404716
[3]     train-rmse:0.364304
[4]     train-rmse:0.327936
[5]     train-rmse:0.295209
[6]     train-rmse:0.265758
[7]     train-rmse:0.239257
[8]     train-rmse:0.215413
[9]     train-rmse:0.193960
[10]    train-rmse:0.174659
[11]    train-rmse:0.157298
[12]    train-rmse:0.141683
[13]    train-rmse:0.127640
[14]    train-rmse:0.115014
[15]    train-rmse:0.103665
[16]    train-rmse:0.093466
[17]    train-rmse:0.084304
[18]    train-rmse:0.076078
[19]    train-rmse:0.068695
```

In [101]:

```
print(pred_acc5, mse5)
```

```
0.66373239436619710.0001345513
```

In [102]:

```
gboost_ensemble = vcat(bets1,bets2,bets3,bets4,bets5);
```

In [103]:
```
# Calculate the prediction accuracy cross validation
mean(gboost_ensemble.==data_train[foldids,3])
```

Out[103]:
```
0.5767605633802817
```

Fit Model on Whole Training Sample - Gradient Boosting

In [104]:

```julia
# Gradient boosting Model
# Make input Array of lagged returns and lagged volume
variables_train = Array(data_train[:,4:end])
ret_train = data_train[:,1];
up_vec_train = data_train[:,3];

variables_test = Array(data_test[:,4:end])
ret_test = data_test[:,1];
up_vec_test = data_test[:,3];

#  This saves memory and makes computation much faster!!!
variables_train=convert.(Float32,variables_train);
ret_train=convert.(Float32,ret_train);
variables_test=convert.(Float32,variables_test);
ret_test=convert.(Float32,ret_test);

# Create an xgboost on the training dataset
dtrain = DMatrix(variables_train, label = ret_train)
boost = xgboost(dtrain, 100, eta = 0.1,max_depth=2)

# make predictions using model
yp=XGBoost.predict(boost,variables_test);

# Calculate MSE
mse = mean((yp-ret_test).^2);

# Calculate predictive accuracy
bets=sign.(yp);
pred_acc = mean(bets.==up_vec_test)
```

```
[1]	train-rmse:0.449519
[2]	train-rmse:0.404620
[3]	train-rmse:0.364214
[4]	train-rmse:0.327850
[5]	train-rmse:0.295128
[6]	train-rmse:0.265682
[7]	train-rmse:0.239187
[8]	train-rmse:0.215347
[9]	train-rmse:0.193898
[10]	train-rmse:0.174602
[11]	train-rmse:0.157244
[12]	train-rmse:0.141633
[13]	train-rmse:0.127594
[14]	train-rmse:0.114971
[15]	train-rmse:0.103625
[16]	train-rmse:0.093429
[17]	train-rmse:0.084271
[18]	train-rmse:0.076047
[19]	train-rmse:0.068667
```

In [105]:

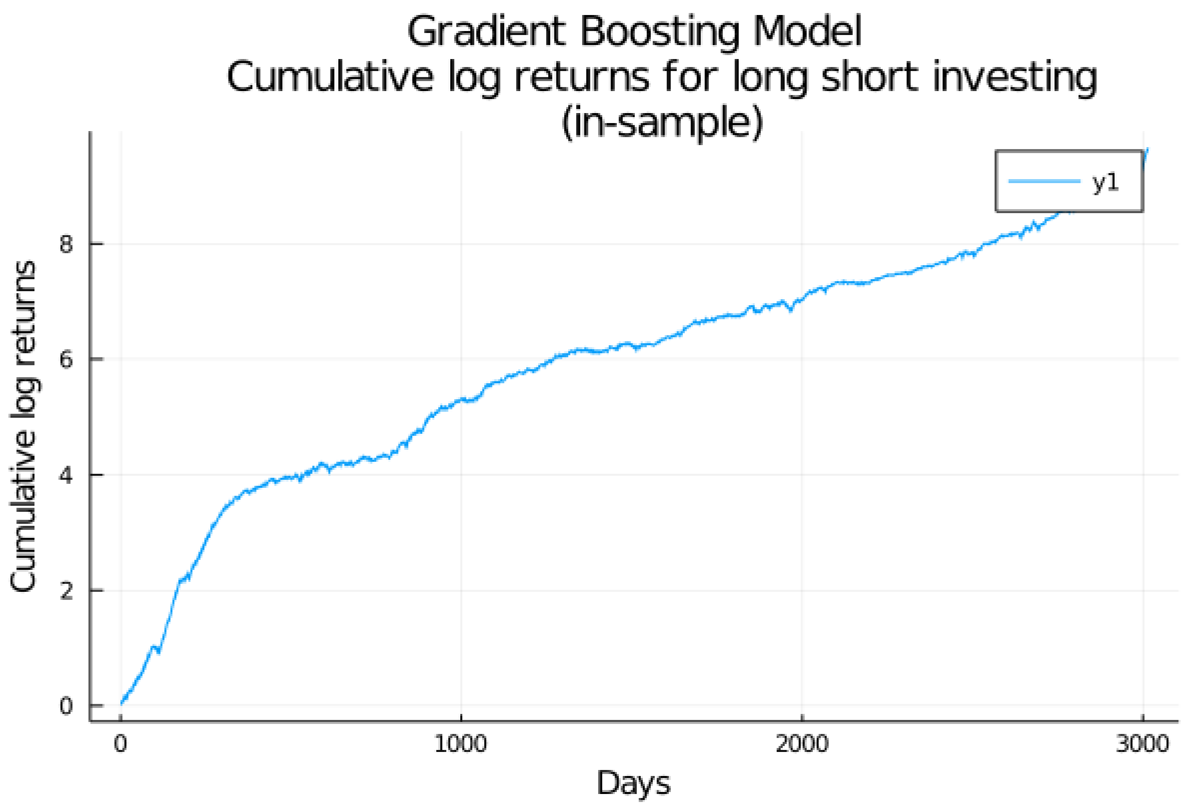
```
# make predictions on training data using model
pred_train=XGBoost.predict(boost,variables_train);

# Calculate MSE
mse_train = mean((pred_train-ret_train).^2);

# Calculate predictive accuracy
up_train=sign.(pred_train);
pred_acc_train = mean(up_train.==up_vec_train)
```

Out[105]: 0.6169154228855721

In [106]:

```
# Plot of Cumulative log returns of training data
plot(cumsum(data_train[:,1].*up_train),fmt=png, title = "Gradient Boosting Model
    Cumulative log returns for long short investing
    (in-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[106]:



In [107]:

```
# calculate in-sample sharpe
sum_boost_in_sample = sum(data_train[:,1].*up_train)/(3015/252)
sd_boost_in_sample = std(data_train[:,1].*up_train)
sum_boost_in_sample/(sd_boost_in_sample*(252^0.5))
```

Out[107]: 4.14151208387201

In [596]:

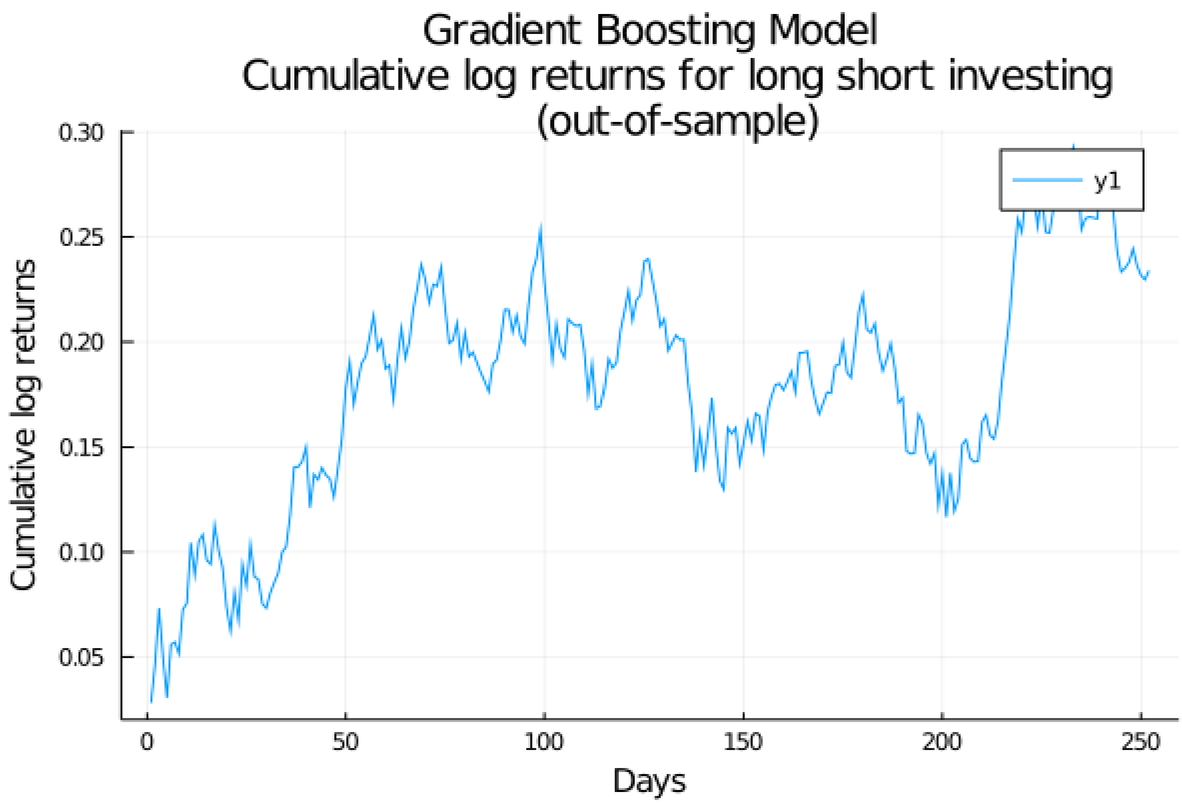
```
# calculate in sample returns
sum(data_train[:,1].*up_train) * (252/3015)
```

Out[596]: 0.8074532181993385

Test Sample - Gradient Boosting

In [108]:
```
# make predictions on testing data using model
pred_test=XGBoost.predict(boost,variables_test);
up_test=sign.(pred_test);

# Plot of Cumulative log returns of testing data
plot(cumsum(data_test[:,1].*up_test),fmt=png, title = "Gradient Boosting Model
    Cumulative log returns for long short investing
    (out-of-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[108]:



In [109]:
```
# calculate out-of-sample sharpe
sum_boost_out_of_sample = sum(data_test[:,1].*up_test)
sd_boost_out_of_sample = std(data_test[:,1].*up_test)
sum_boost_out_of_sample/(sd_boost_out_of_sample*(252^0.5))
```

Out[109]: 1.1381686111686875

In [583]:
```
# calculate out of sample returns
sum(data_test[:,1].*up_test)
```

Out[583]: 0.23396611126091874

Model 5 - Linear Discriminant Analysis

In [110]:
```
using DiscriminantAnalysis
using DiscriminantAnalysis: lda
using DiscriminantAnalysis: classify
using DiscriminantAnalysis: posteriors
```

In [111]:
```
# Convert to 1/-1 as in all_data
convert_lda_output(datas)=
    for i=1:length(datas)
        if datas[i]== 2.0
            datas[i] = 1
        else
            datas[i] = -1
        end
    end
```

Out[111]:
```
convert_lda_output (generic function with 1 method)
```

In [112]:
```
# Fit LDA excluding fold1 and get fold1 "out-of-sample" predictions
ld_fit1=lda(data_train[test1,4:end],1 .+(data_train[test1,1].>=0))
ld_preds1=classify(ld_fit1,data_train[fold1ids,4:end])
convert_lda_output(ld_preds1)
```

In [113]:
```
findaccuracy(ld_preds1,data_train[fold1ids,3])
```

Out[113]:
```
0.5563380281690141
```

In [114]:
```
ld_fit2=lda(data_train[test2,4:end],1 .+(data_train[test2,1].>=0))
ld_preds2=classify(ld_fit2,data_train[fold2ids,4:end])
convert_lda_output(ld_preds2)
```

In [115]:
```
findaccuracy(ld_preds2,data_train[fold2ids,3])
```

Out[115]:
```
0.5528169014084507
```

In [116]:
```
ld_fit3=lda(data_train[test3,4:end],1 .+(data_train[test3,1].>=0))
ld_preds3=classify(ld_fit3,data_train[fold3ids,4:end])
convert_lda_output(ld_preds3)
```

In [117]:
```
findaccuracy(ld_preds3,data_train[fold3ids,3])
```

Out[117]:
```
0.5545774647887324
```

In [118]:
```
ld_fit4=lda(data_train[test4,4:end],1 .+(data_train[test4,1].>=0))
ld_preds4=classify(ld_fit4,data_train[fold4ids,4:end])
convert_lda_output(ld_preds4)
```

In [119]:
```
findaccuracy(ld_preds4,data_train[fold4ids,3])
```

Out[119]:
```
0.5475352112676056
```

In [120]:
```
ld_fit5=lda(data_train[test5,4:end],1 .+(data_train[test5,1].>=0))
ld_preds5=classify(ld_fit5,data_train[fold5ids,4:end])
convert_lda_output(ld_preds5)
```

In [121]:
```
findaccuracy(ld_preds5,data_train[fold5ids,3])
```

Out[121]:
```
0.5334507042253521
```

In [122]:
```
# Collate forecasts
ld_ensemble = vcat(ld_preds1,ld_preds2,ld_preds3,ld_preds4,ld_preds5);
```

In [123]:
```
# Calculate the prediction accuracy cross validation
mean(ld_ensemble.==data_train[foldids,3])
```

Out[123]:
```
0.548943661971831
```

Fit Model on Whole Training Sample - Linear Discriminant Analysis

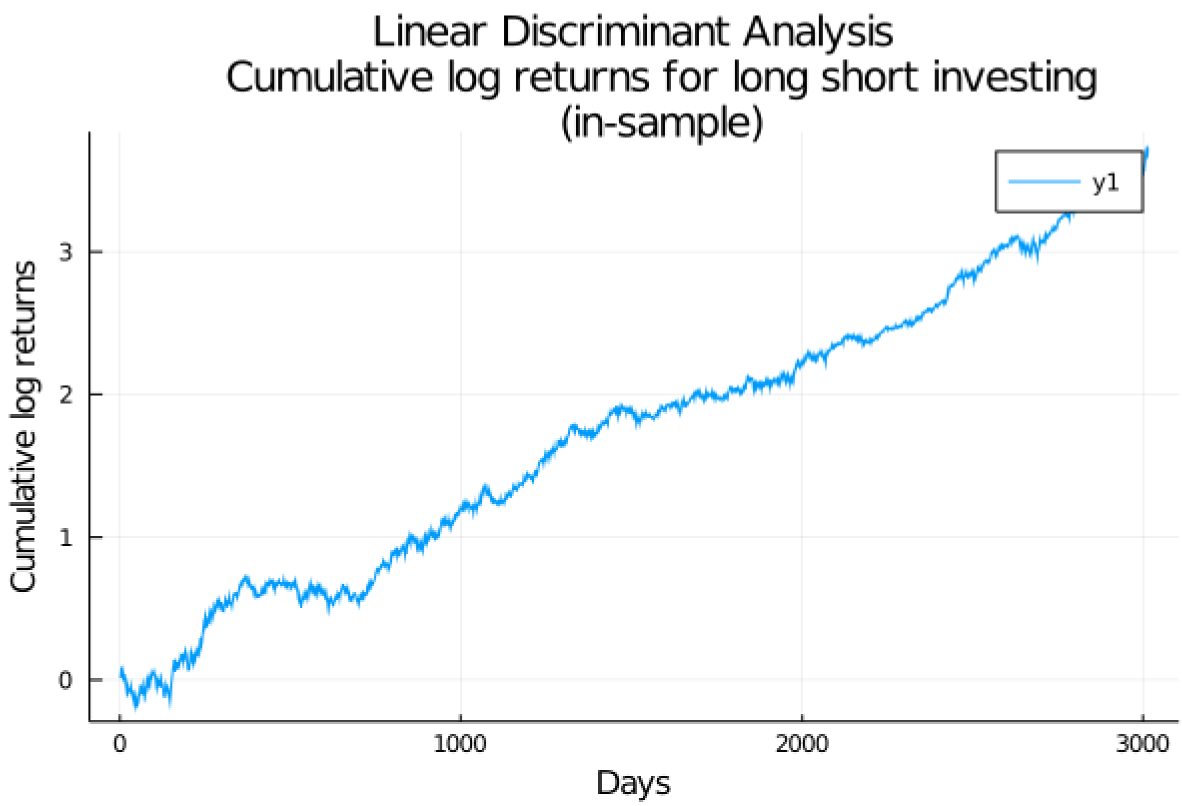
In [124]:
```
ld_fit=lda(data_train[:,4:end],1 .+(data_train[:,1].>=0));
```

In [125]:
```
ld_preds=classify(ld_fit,data_train[:,4:end]);
```

In [126]:
```
ld_post=posteriors(ld_fit,data_train[:,4:end]);
```

In [127]:
```
plot(cumsum(data_train[:,1].*sign.(ld_post[:,2].-.5)),fmt=png, title = "Linear Di
    Cumulative log returns for long short investing
    (in-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[127]:



In [128]:
```
# calculate in-sample sharpe
sum_lda_in_sample = sum(data_train[:,1].*sign.(ld_post[:,2].-.5))/(3015/252)
sd_lda_in_sample = std(data_train[:,1].*sign.(ld_post[:,2].-.5))
sum_lda_in_sample/(sd_lda_in_sample*(252^0.5))
```

Out[128]: 1.5526966513774125

In [597]:
```
# calculate in sample returns
sum(data_train[:,1].*sign.(ld_post[:,2].-.5)) * (252/3015)
```

Out[597]: 0.31137234487552534

In [129]:
```
convert_lda_output(ld_preds)
```

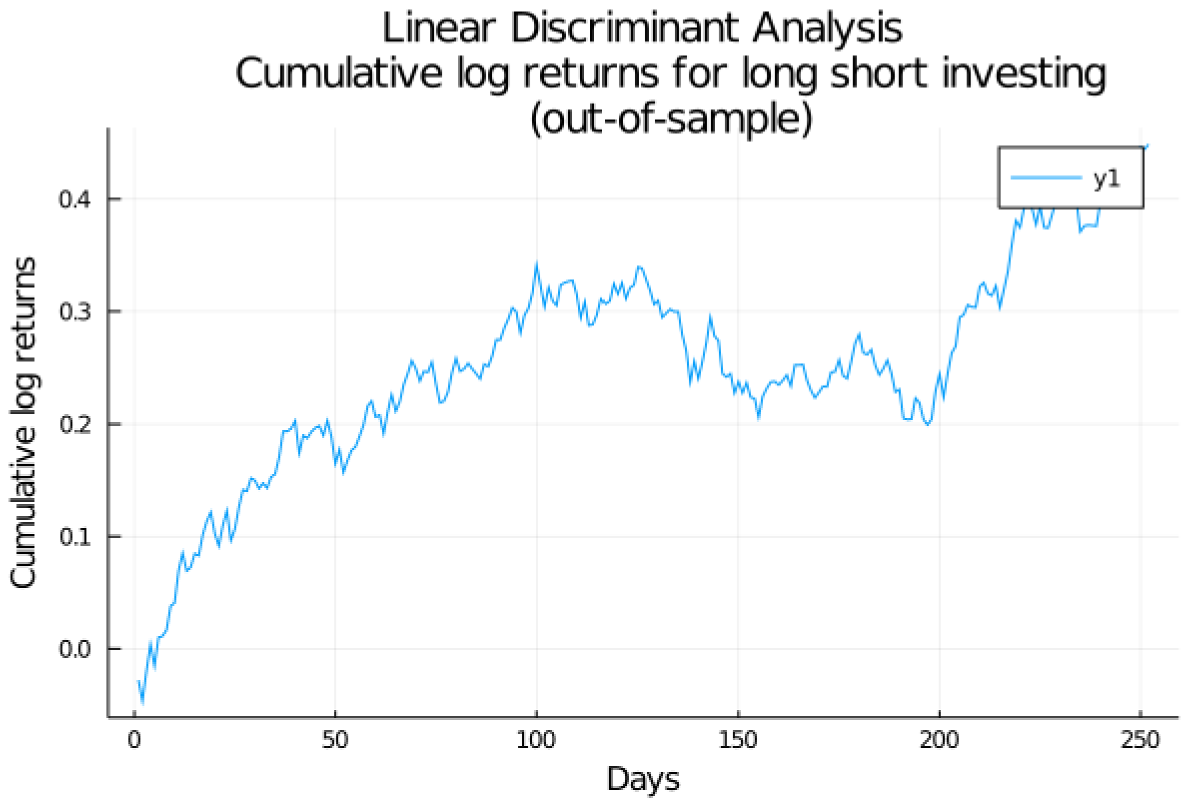
In [130]:
```
findaccuracy(ld_preds,data_train[:,3])
```

Out[130]: 0.5611940298507463

Test Sample - Linear Discriminant Analysis

In [131]:
```
# Make out-of-sample predictions
ld_preds_test=classify(ld_fit,data_test[:,4:end]);
ld_post_test=posteriors(ld_fit,data_test[:,4:end]);

# Plot out-of-sample
plot(cumsum(data_test[:,1].*sign.(ld_post_test[:,2].-.5)),fmt=png, title = "Linea
    Cumulative log returns for long short investing
    (out-of-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[131]:



In [132]:
```
# calculate out-of-sample sharpe
sum_lda_out_of_sample = sum(data_test[:,1].*sign.(ld_post_test[:,2].-.5))
sd_lda_out_of_sample = std(data_test[:,1].*sign.(ld_post_test[:,2].-.5))
sum_lda_out_of_sample/(sd_lda_out_of_sample*(252^0.5))
```

Out[132]: 2.1979868259430813

In [585]:
```
# calaculate out of sample returns
sum(data_test[:,1].*sign.(ld_post_test[:,2].-.5))
```

Out[585]: 0.448693129276095

Step 4

Deep Neural Network

We ran a deep neural network on the expert forecasts above alongside the original input variables.

In [133]:
```
using Flux
```

In [134]:
```
stacking_forecasts = hcat(data_train[foldids,3:end],lasso_ensemble,logistic_ensem
```

In [135]:
```
# Dropout used to prevent overfitting
m=Chain(Dense(25,20,relu),Dense(20,10,relu),Dropout(.1),Dense(10,1,tanh))
```

Out[135]:
```
Chain(Dense(25, 20, relu), Dense(20, 10, relu), Dropout(0.1), Dense(10, 1, tan
h))
```

In [136]:
```
m2=m # for use later
m[1]
```

Out[136]:
```
Dense(25, 20, relu)
```

In [137]:
```
loss2(x,y)=mean((m(x)'.*y'.-1).^2)
```

Out[137]:
```
loss2 (generic function with 1 method)
```

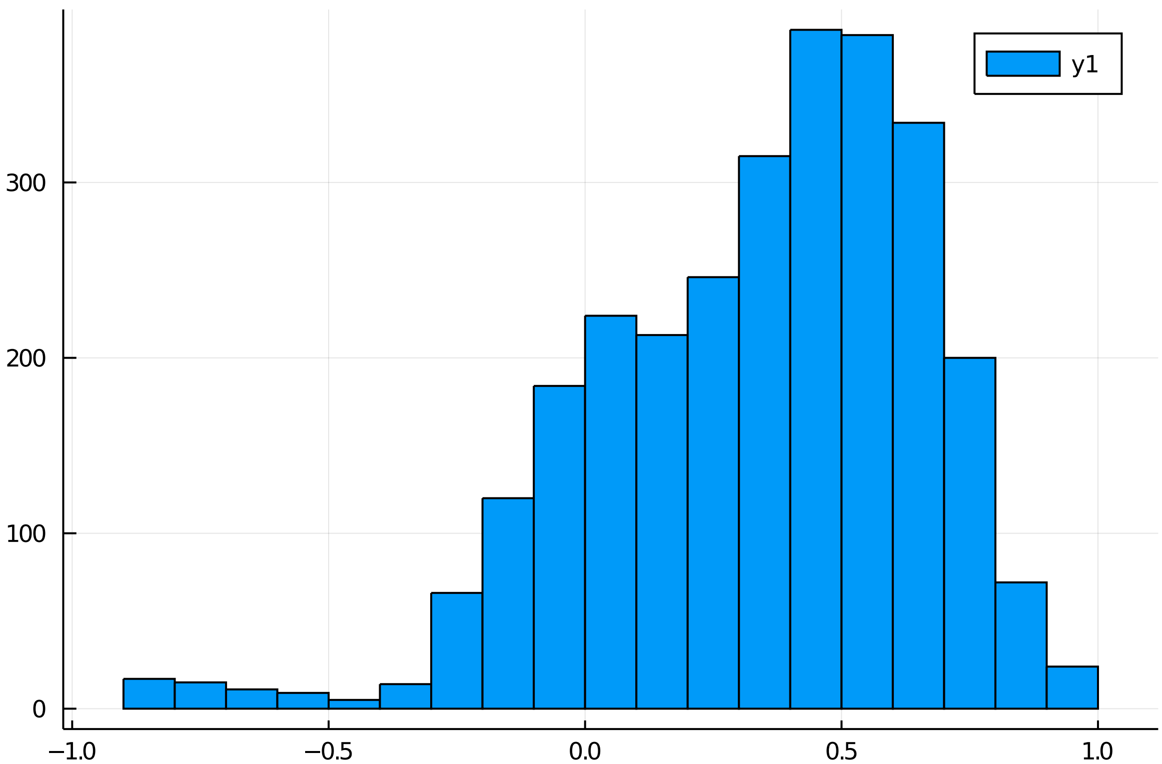
In [142]:
```
X=Array(stacking_forecasts[:,2:end])  # Make input Array
y=stacking_forecasts[:,1];

X = convert.(Float32,X)
y = convert.(Float32,y);
```

In [533]:
```
Flux.@epochs 10000 Flux.train!(loss2, Flux.params(m), [(X', y')], ADAM());
```

```
┌ Info: Epoch 1
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 2
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 3
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 4
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 5
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 6
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 7
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 8
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 9
└ @ Main C:\Users\tomas\.julia\packages\Flux\05b38\src\optimise\train.jl:114
┌ Info: Epoch 10
```

In [534]:
```
nn_scores2=vec(m2(X'))
nn_preds2=sign.(nn_scores2);
mean(nn_preds2.==y)
```
Out[534]: 0.581338028169014

In [613]:
```
# Histogram of scores
histogram(nn_scores2)
```
Out[613]:



Step 5

Stacked Ensemble

1. Run each model separately on the final 252 days
2. Get the -1/1 output values and get the predictions based on our already calculated neural network using nn_scores2=vec(m2(X')); nn_preds2=sign.(nn_scores2);
3. Make a copy of the matrix to get a 0/1 output for the proportional trading strategy

In [535]:
```
# Lasso Predictions for out of sample
lasso_preds_outofsample = GLMNet.predict(pathx,data_test[:,4:end])

#Convert to binary output for ensemble
lasso_binary_preds_outofsample = ones(length(lasso_preds_outofsample))
# Get directions
for i=1:length(lasso_preds_outofsample)
    if lasso_preds_outofsample[i]>0
        lasso_binary_preds_outofsample[i] = 1
    else
        lasso_binary_preds_outofsample[i] = -1
    end
end
```

In [536]:
```
# Logistic Regression Predictions for out of sample
fin;
```

In [537]:
```
# LD Predictions for out of sample
ld_preds_outofsample=classify(ld_fit,data_test[:,4:end])
convert_lda_output(ld_preds_outofsample)
ld_preds_outofsample;
```

In [538]:
```
# Support Vector Machine Predictions for out of sample
sv_preds;
```

In [539]:
```
# Gradient Boosting Predictions for out of sample
bets;
```

In [540]:
```
stacking_test = hcat(data_test[:,4:end], lasso_binary_preds_outofsample, fin,sv_p
```

In [541]:
```
nn_scores3=vec(m2(stacking_test'));
nn_preds3=sign.(nn_scores3);
mean(nn_preds3.==data_test[:,3])
```

Out[541]:
```
0.5476190476190477
```

In [614]:
```
maximum(nn_scores3)
```

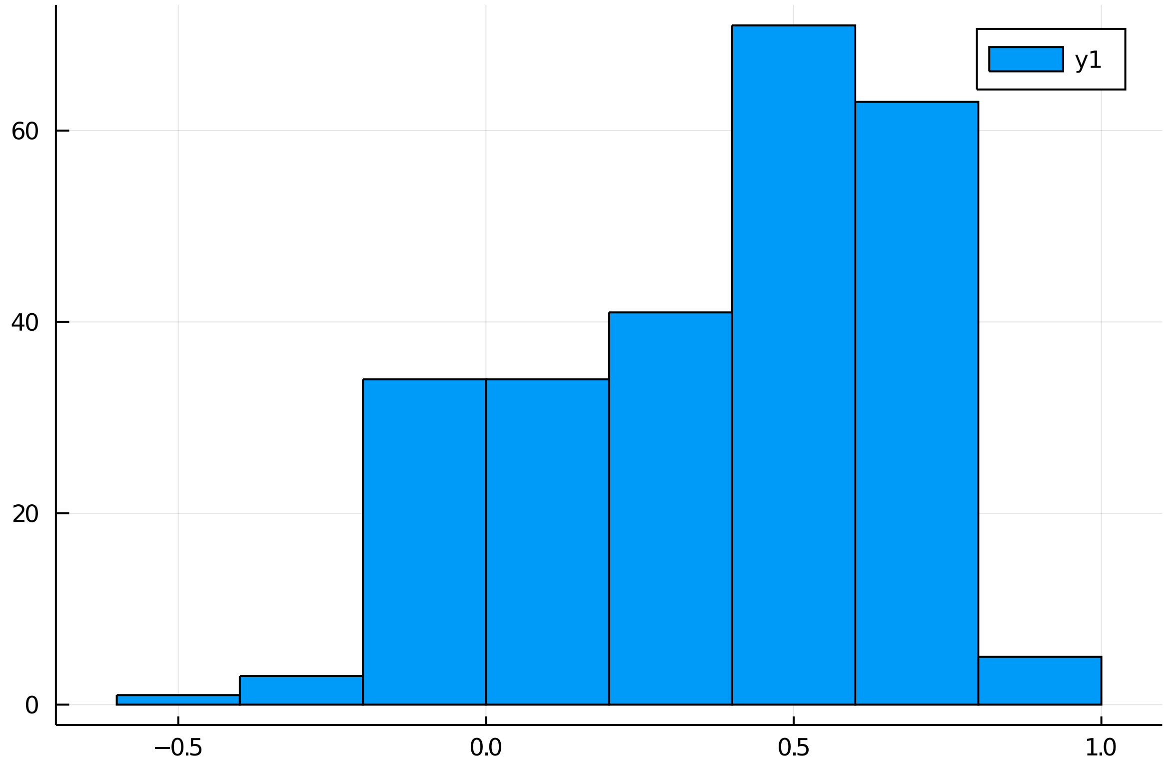
Out[614]:
```
0.85524917f0
```

In [615]:
```
minimum(nn_scores3)
```

Out[615]:
```
-0.5570618f0
```

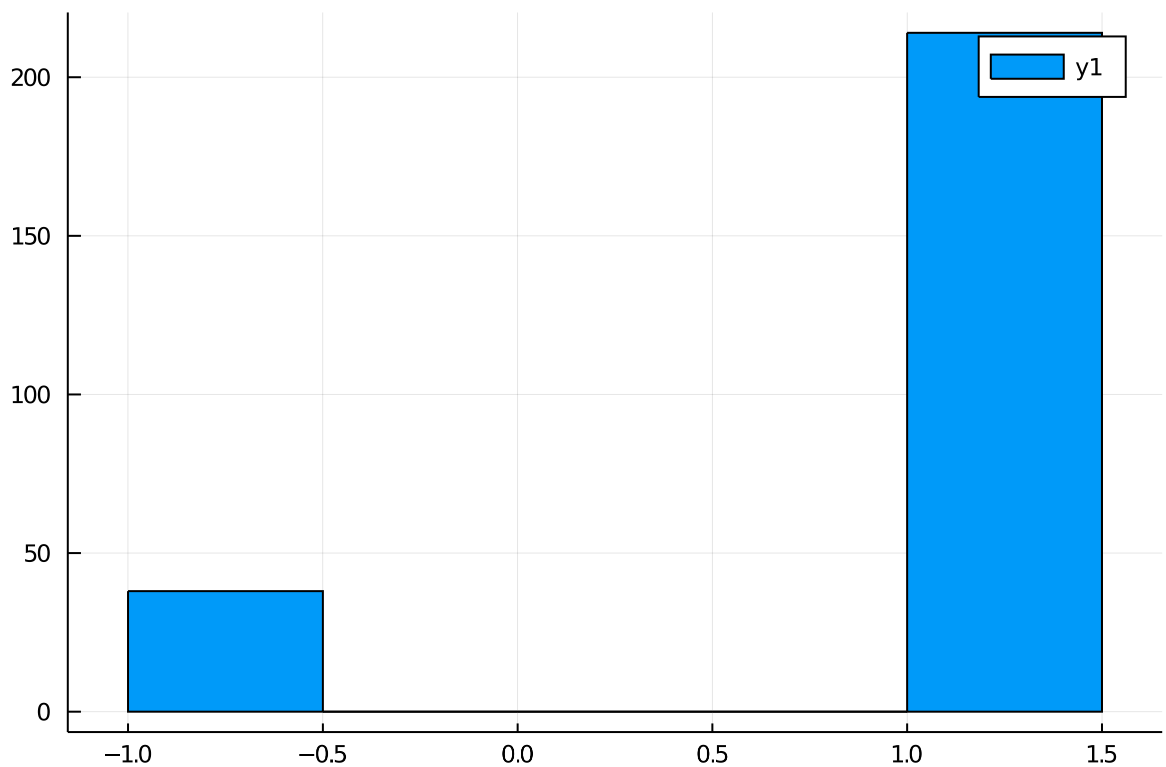
In [542]:
```
histogram(nn_scores3)
```

Out[542]:



In [543]:
```
histogram(nn_preds3)
```

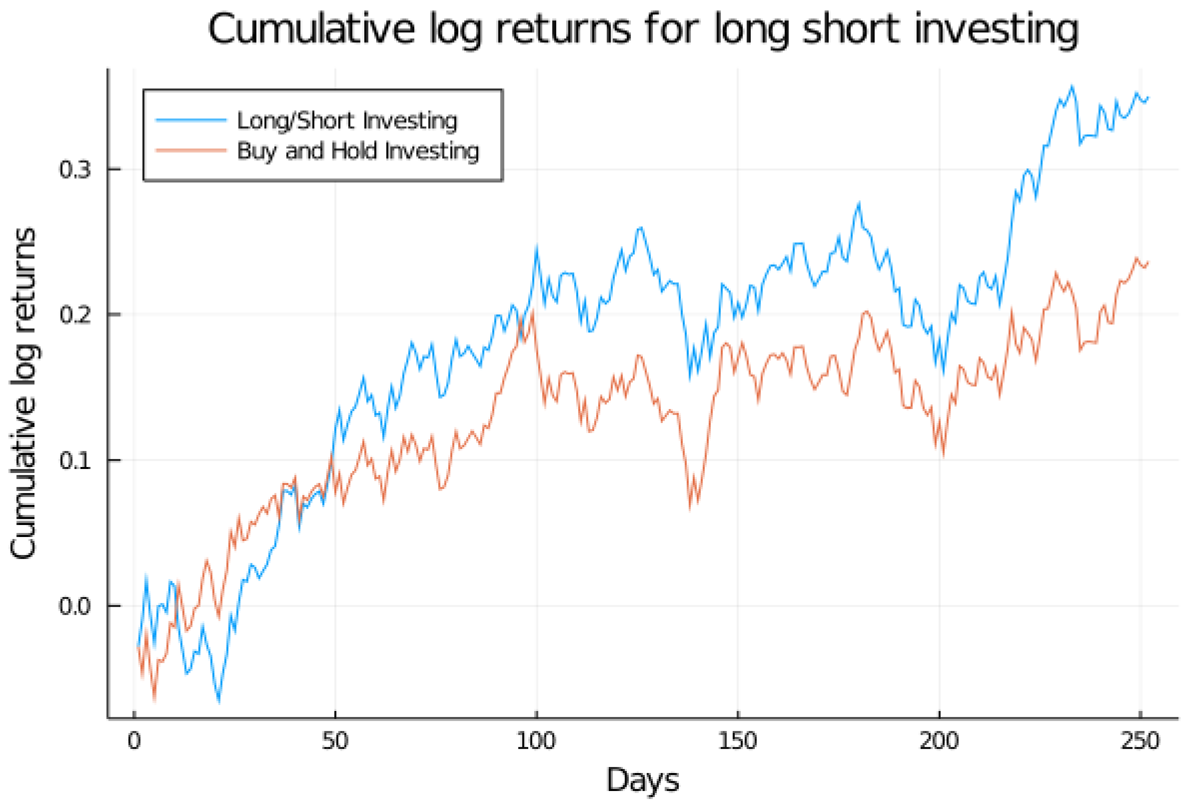
Out[543]:



In [574]:

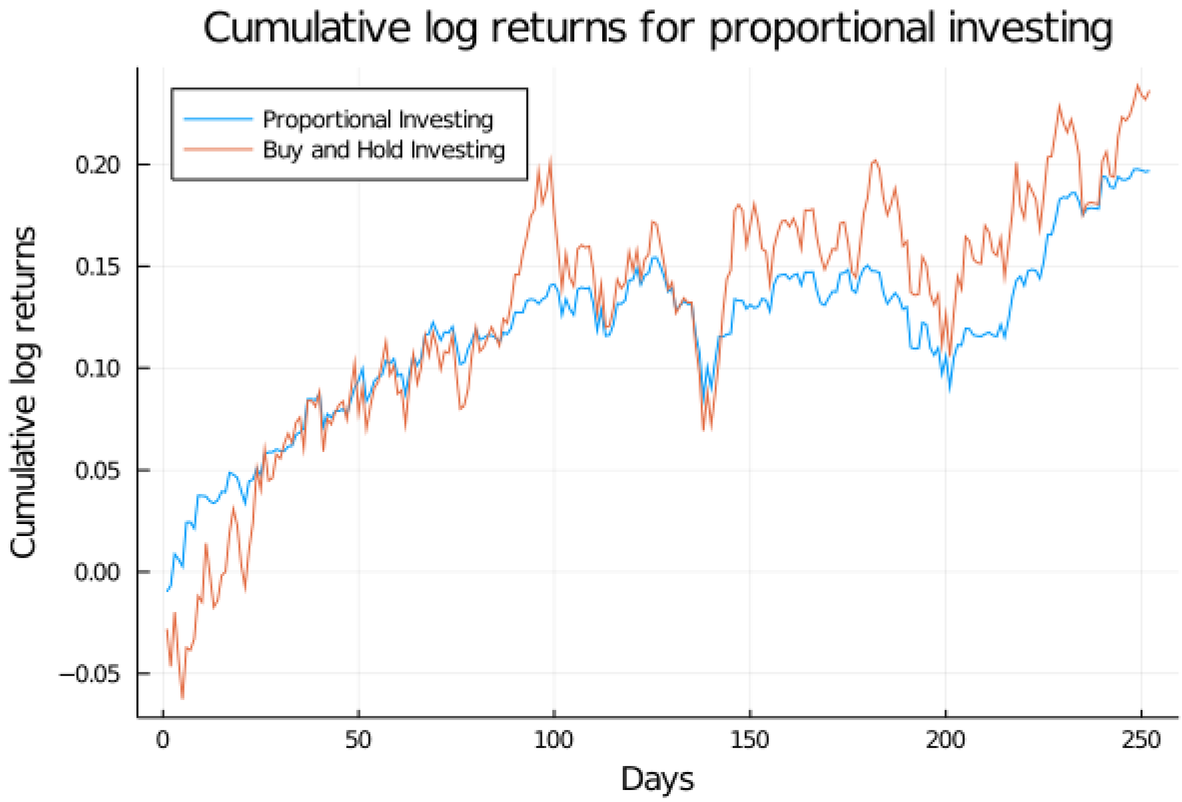
```
# Long Short cumsum
plot(cumsum(nn_preds3.*data_test[:,1]), label = "Long/Short Investing",
    legend=:topleft,fmt="png", title = "Cumulative log returns for Long/Short inv
plot!(cumsum(data_test[:,1]), label = "Buy and Hold Investing")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[574]:



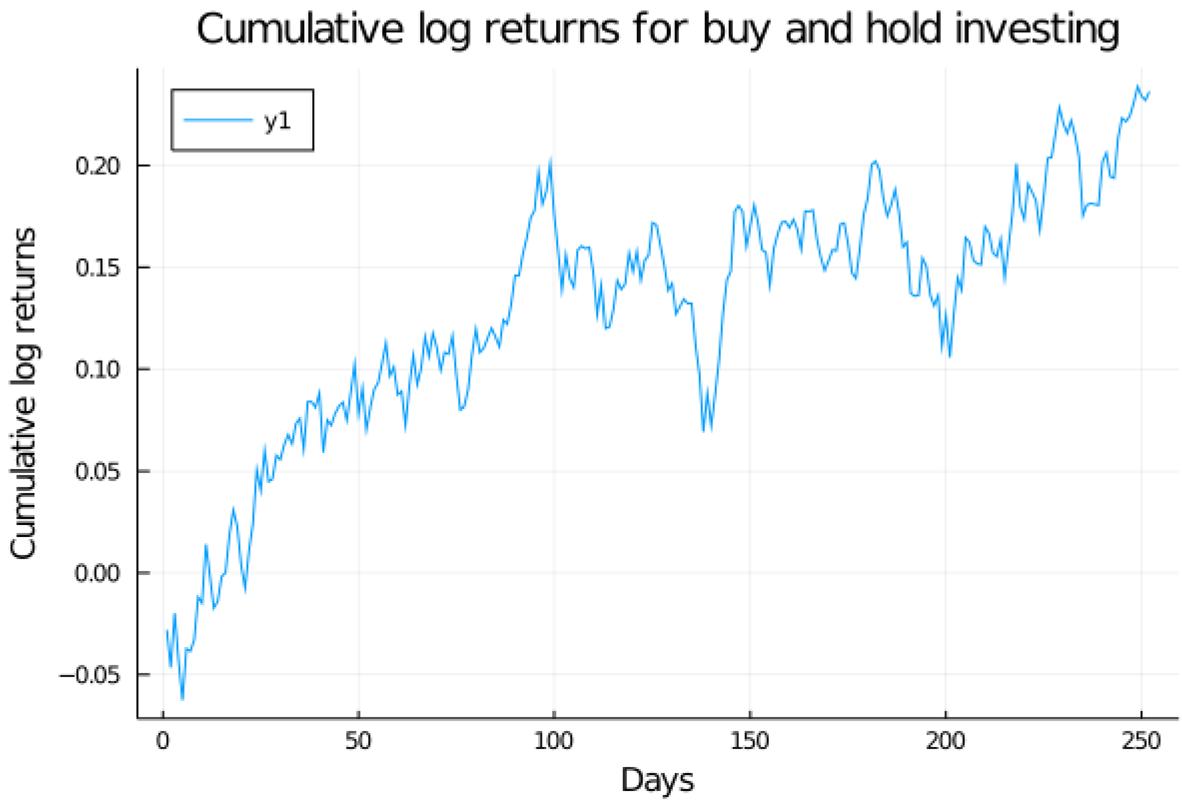
In [573]:
```
# Proportional cumsum
plot(cumsum(nn_scores3.*data_test[:,1]), label = "Proportional Investing",
    legend=:topleft,fmt="png", title = "Cumulative log returns for proportional i
plot!(cumsum(data_test[:,1]), label = "Buy and Hold Investing")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[573]:



In [546]:
```
# Buy and hold investing cumsum
plot(cumsum(data_test[:,1]),legend=:topleft,fmt="png", title = "Cumulative log re
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[546]:



In [589]:
```
# Long Short Investing - out-of-sample  - returns
sum(nn_preds3.*data_test[:,1])
```

Out[589]: 0.34989474070506893

In [591]:
```
# Proportional Investing - out-of-sample - returns
sum(nn_scores3.*data_test[:,1])
```

Out[591]: 0.19691574402055304

In [590]:
```
# Buy and Hold - out-of-sample - returns
sum(data_test[:,1])
```

Out[590]: 0.23635659649735793

Calculate Sharpe Ratio

In [547]:
```
using Statistics
```

In [548]:
```
# Long Short Investing
sum_long_short = sum(nn_preds3.*data_test[:,1])
sd_long_short = std(nn_preds3.*data_test[:,1])
sum_long_short/(sd_long_short*(252^0.5))
```

Out[548]: 1.7075807984958395

In [549]:
```
# Proportional Investing
cumsum_proportional = sum(nn_scores3.*data_test[:,1])
proportional_std = std(nn_scores3.*data_test[:,1])
cumsum_proportional/(proportional_std*(252^0.5))
```

Out[549]: 1.9200443648575534

In [550]:
```
# Buy and Hold Investing
sum_buy_hold = sum(data_test[:,1])
sd_buy_hold = std(data_test[:,1])
sum_buy_hold/(sd_buy_hold*(252^0.5))
```

Out[550]: 1.149858485499543

In [551]:
```
sum_buy_hold
```

Out[551]: 0.23635659649735793

In [552]:
```
mean(bets.==data_test[:,3])
```

Out[552]: 0.5436507936507936

In [553]:
```
mean(fin.==data_test[:,3])
```

Out[553]: 0.5912698412698413

In [554]:
```
mean(ld_preds_outofsample.==data_test[:,3])
```

Out[554]: 0.5912698412698413

In [555]:
```
mean(sv_preds.==data_test[:,3])
```

Out[555]: 0.5396825396825397

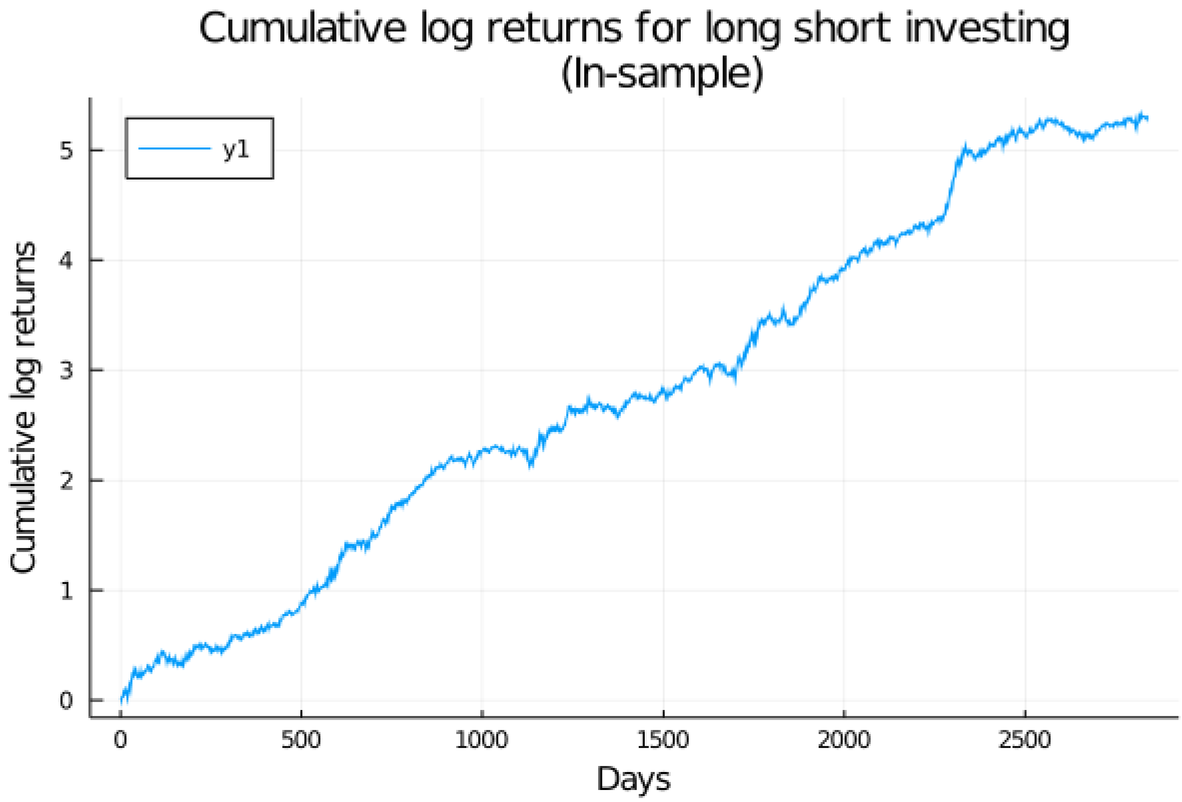
In [556]:
```
mean(lasso_binary_preds_outofsample.==data_test[:,3])
```

Out[556]: 0.5992063492063492

In [557]:

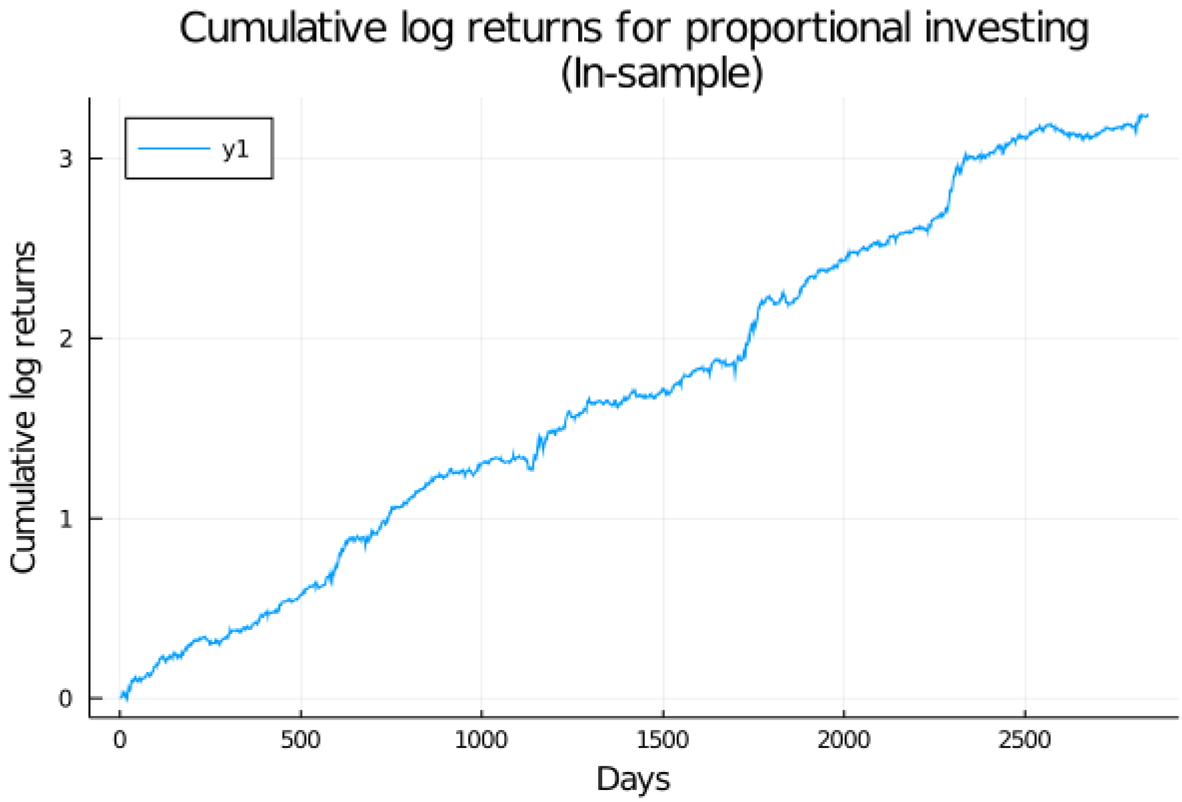
```
# Long Short cumsum - in-sample
plot(cumsum(nn_preds2.*data_train[foldids,1]),legend=:topleft,fmt="png", title =
    (In-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[557]:



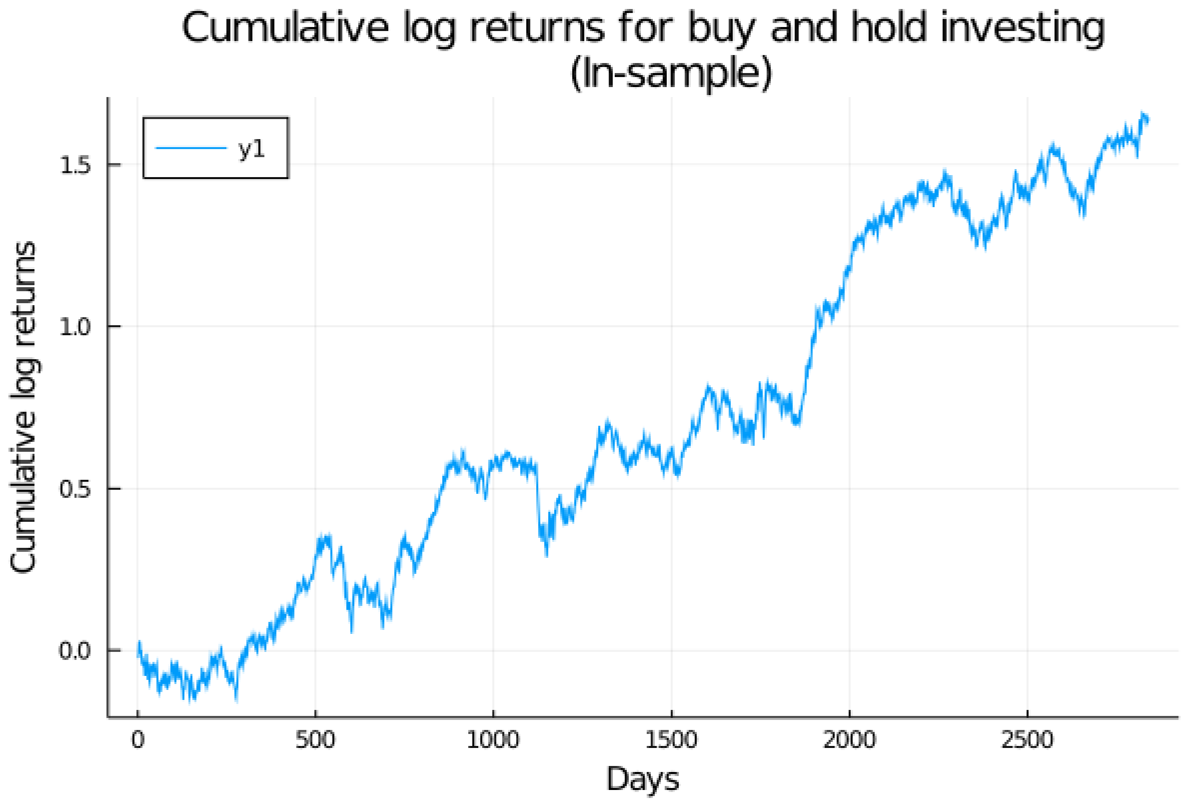
In [558]:
```
# Proportional cumsum - in-sample
plot(cumsum(nn_scores2.*data_train[foldids,1]),legend=:topleft,fmt="png", title =
    (In-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[558]:



In [559]:
```
# Buy and hold investing cumsum
plot(cumsum(data_train[foldids,1]),legend=:topleft,fmt="png", title = "Cumulative
    (In-sample)")
xlabel!("Days")
ylabel!("Cumulative log returns")
```

Out[559]:



In [599]:
```
# Long Short Investing - in-sample  - returns
sum(nn_preds2.*data_train[foldids,1]) * (252/2840)
```

Out[599]: 0.4704518976444124

In [600]:
```
# Proportional Investing - in-sample - returns
sum(nn_scores2.*data_train[foldids,1]) * (252/2840)
```

Out[600]: 0.28767996521282924

In [601]:
```
# Buy and Hold Investing - in-sample - returns
sum(data_train[foldids,1]) * (252/2840)
```

Out[601]: 0.14589928348978617

In [560]:
```
# Long Short Investing - in-sample  - sharpe
sum_long_short_in_sample = sum(nn_preds2.*data_train[foldids,1])/(2840/252)
sd_long_short_in_sample = std(nn_preds2.*data_train[foldids,1])
sum_long_short_in_sample/(sd_long_short_in_sample*(252^0.5))
```

Out[560]: 2.3553236369156716

In [561]:
```
# Proportional Investing - in-sample - sharpe
sum_proportional_in_smaple = sum(nn_scores2.*data_train[foldids,1])/(2840/252)
proportional_std_in_smaple = std(nn_scores2.*data_train[foldids,1])
sum_proportional_in_smaple/(proportional_std_in_smaple*(252^0.5))
```

Out[561]: 2.8530618245526935

In [562]:
```
# Buy and Hold Investing - in-sample - sharpe
sum_buy_hold_in_smaple = sum(data_train[foldids,1])/(2840/252)
sd_buy_hold_in_sample = std(data_train[foldids,1])*(252^0.5)
sum_buy_hold_in_smaple/(sd_buy_hold_in_sample)
```

Out[562]: 0.7232840952402656

In [ ]: