Machine Learning Assignment II

Ze Yang (zey@andrew.cmu.edu)

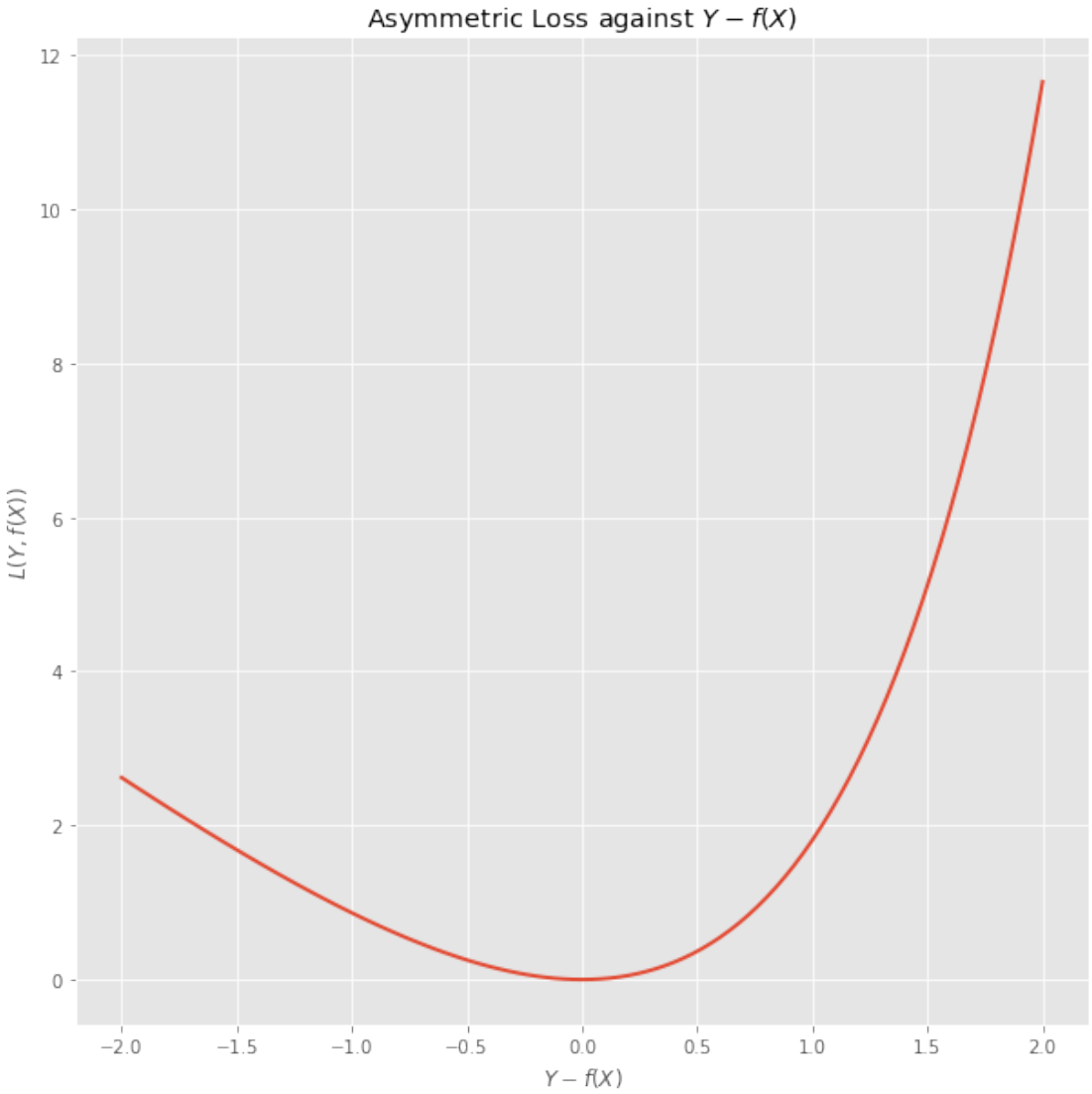
January 29, 2018

1 Asymmetric Loss and the Regression Function

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline

from scipy.stats import norm
from sklearn.linear_model import Lasso, LassoCV, lasso_path
from sklearn.model_selection import validation_curve, KFold
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

from progressbar import ProgressBar
```

In [3]:
```
def asymmetric_loss(a, b):
    return lambda true, pred: np.array(
    [b*(np.exp(a*(y-y_hat))-a*(y-y_hat)-1)
     for (y, y_hat) in zip(true, pred)])
```

In [4]:
```
y = np.linspace(-2, 2, 101)
y_hat = np.array([0]*101)
loss = asymmetric_loss(1.1, 2)

fig, ax = plt.subplots(1, 1, figsize=(10,10))
ax.plot(y,loss(y,y_hat), linewidth=2)
ax.set_title(r"Asymmetric Loss against $Y-f(X)$")
ax.set_xlabel(r"$Y-f(X)$")
_ = ax.set_ylabel(r"$L(Y,f(X))$")
```



Question (a) The loss function is asymmetric with respect to $z = Y - f(X) = 0$. Namely, $\mathcal{L}(z)$ is greater for $z > 0$ than $z < 0$ with the same absolute value.
Such behavior of $\mathcal{L}(Y, f(X))$ implies that it "prefers" positive prediction error than negative error with same absolute value. We may want to adopt such a loss function in a real problem if we know that *underestimating* y is going to hurt us more than *overestimating* it on a same extent, and thus one of our main concerns is to reduce the chance in which our model underestimates y.

Question (b) The expected loss is calculated as

$$\begin{aligned} \mathbb{E}\left[\mathcal{L}(Y, f(X))\right] &= \mathbb{E}\left[\mathbb{E}\left[b\left(e^{a(Y-f(X))} - a(Y - f(X)) - 1\right)\middle| X\right]\right] \\ &= \mathbb{E}\left[be^{-af(X)}\mathbb{E}\left[e^{aY}\middle| X\right] + abf(X) - ab\mathbb{E}\left[Y|X\right] - b\right] \end{aligned} \tag{1}$$

The first & second order derivatives with respect to $f(X)$:

$$\begin{aligned}\frac{\partial \mathcal{L}(Y, f(X))}{\partial f} &= \mathbb{E}\left[-abe^{-af(X)}\mathbb{E}\left[e^{aY}\middle|X\right] + ab\right] \\ \frac{\partial^2 \mathcal{L}(Y, f(X))}{\partial f^2} &= \mathbb{E}\left[a^2be^{-af(X)}\mathbb{E}\left[e^{aY}\middle|X\right]\right] \geq 0, \quad \forall f(X) \in \mathbb{R}; \quad b \geq 0\end{aligned} \tag{2}$$

The second order condition implies that $\mathcal{L}$ is convex in f, so its global minimum is reached at:

$$\frac{\partial \mathcal{L}(Y, f(X))}{\partial f} = 0 \quad \Rightarrow \quad f(X) = \frac{1}{a}\log \mathbb{E}\left[e^{aY}\middle|X\right] \quad (\dagger) \tag{3}$$

Question (c) Given that $Y|(X = x) \sim \mathcal{N}(\beta x, \sigma^2)$, $(\dagger)$ can be explicitly evaluated using the moment generating function of normal distribution, namely $\mathbb{E}\left[e^{aY}\middle|X = x\right] = e^{\beta x a + \frac{1}{2}\sigma^2 a^2}$. We have:

$$f(x) = \beta x + \frac{1}{2}\sigma^2 a \tag{4}$$

That is, our estimate of y will be greater than its conditional mean by a constant to accommodate the fact that we dislike underestimating y.

Question (d)

```
In [5]: # Set some parameters
        beta = 0.5
        b = 2
        sigma = 2
        a = 1

        #Define the loss function, where z = y - yhat
        def loss(z):
            return b*(np.exp(a*z)-a*z-1)

        # Estimation functions
        # Estimation using the conditional expectation of Y|X
        def f_condexp(x):
            return beta*x

        # TODO: Put your function in here.
        # You can reference a,b,sigma, and it will just pull them from
        # the outside namespace
        def f_yours(x):
            return beta*x+0.5*a*(sigma)**2
        #Simulation to see how you do
        reps = 1000

        # Just generate the X variables normally.  We don't really care
        x = norm.rvs(size=reps, loc=0, scale=1)

        # Generate the Y variables from our normal model
```

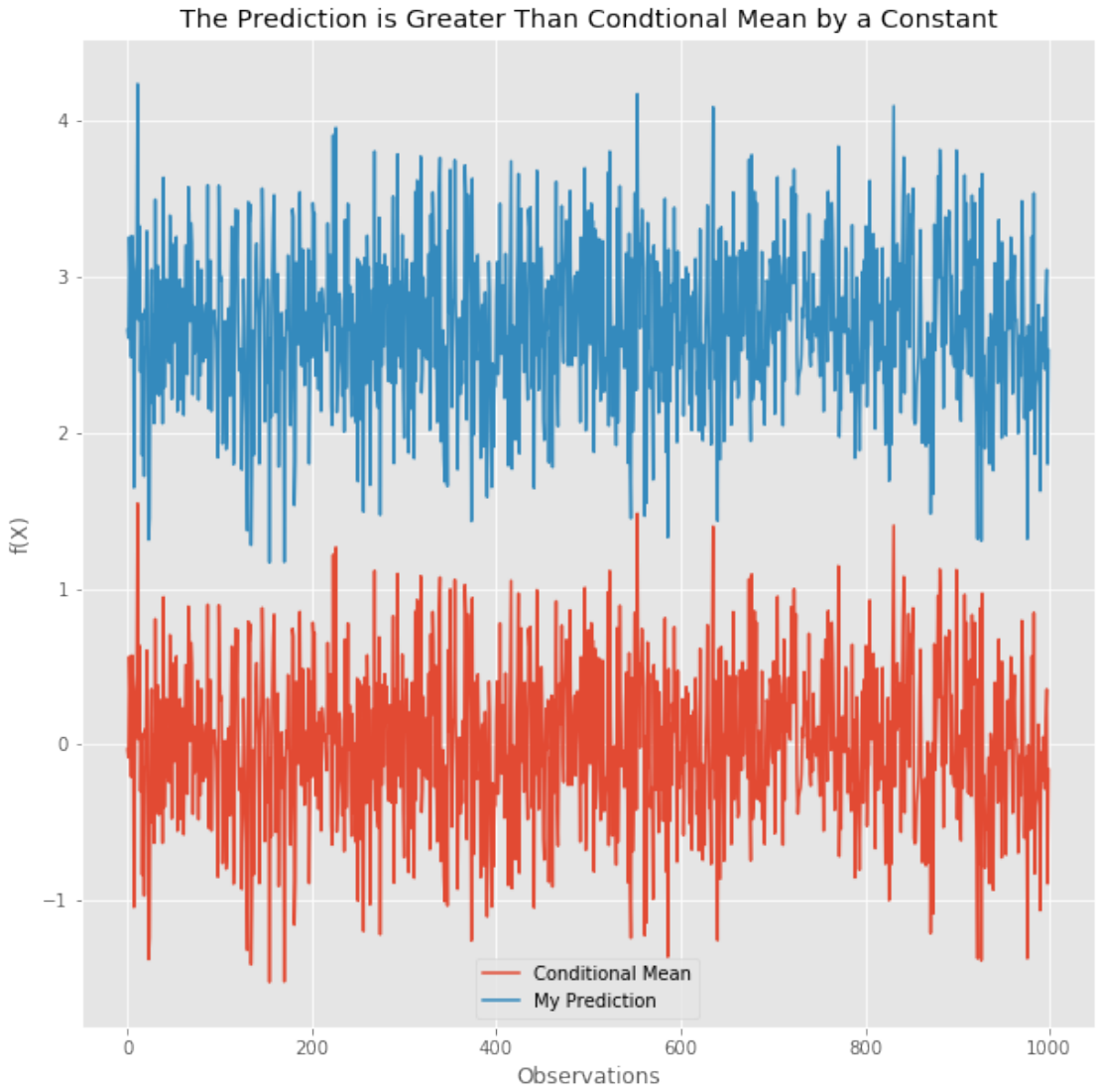
```
        y = norm.rvs(size=reps, loc=x*beta, scale=sigma)

        # Calculate the fitted values for each method
        yhat_condexp = np.apply_along_axis(f_condexp, 0, x)
        yhat_yours = np.apply_along_axis(f_yours, 0, x)

        # Compute the losses
        condexp_losses = np.apply_along_axis(loss, 0, y-yhat_condexp)
        your_losses = np.apply_along_axis(loss, 0, y-yhat_yours)

        print("Average loss of the conditional expectation:",
              round(np.mean(condexp_losses),2))

        print("Average loss of your method:",
              round(np.mean(your_losses),2))
```

```
Average loss of the conditional expectation: 9.85
Average loss of your method: 3.62
```

The average loss of my function and the conditional mean in the simulation are shown above. Indeed, I have a lower average loss than the conditional mean. Like we've already discussed in question (c), our estimate of y is greater than its conditional mean by a constant to accommodate the fact that we dislike underestimating y (see the plot in the section below).

Go back to our first plot of $\mathcal{L}(Y, f(X))$ V.S. $Y - f(X)$; If we use the conditional mean estimator, $f(X) = \beta X$, then the distribution of the quantity $Y - f(X)$ on the random sample is centered at 0. But with our estimator, $f(X)$ will be always greater than βX by a constant, which implies that we *translate* the distribution of $Y - f(X)$ to *left*, where the loss is lower than its corresponding part in the right.

```
In [6]: fig, ax = plt.subplots(1, 1, figsize=(10,10))

        ax.plot(yhat_condexp, label="Conditional Mean")
        ax.plot(yhat_yours, label="My Prediction")
        ax.set_title("The Prediction is Greater Than Condtional Mean by a Constant")
        ax.set_xlabel("Observations")
        ax.set_ylabel("f(X)")
        _ = ax.legend()
```



---

2 Derivation of Ridge Estimator

Proof. **Question (a)** The ridge regression estimate can be written as the solution to the following optimization problem:

$$\begin{aligned}\widehat{\boldsymbol{\beta}}^{\text{ridge}} &= \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2 \\ &= \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \|\mathbf{0}_p - \sqrt{\lambda}\boldsymbol{I}_p\boldsymbol{\beta}\|_2^2\end{aligned} \tag{5}$$

where we translate the ℓ^2 norm term into an expression that looks alike the firt least square term, where $\mathbf{0}_p$ is $p\times 1$ column vector of zeros, $\boldsymbol{I}_p$ is $p\times p$ identity matrix. We can now augment the first

term with the second one. Let

$$\overline{\boldsymbol{y}} = \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{0}_p \end{pmatrix} \in \mathbb{R}^{n+p}, \quad \overline{\boldsymbol{X}} = \begin{pmatrix} \boldsymbol{X} \\ \sqrt{\lambda}\boldsymbol{I}_p \end{pmatrix} \in \mathbb{R}^{(n+p)\times p}$$

Then the ridge regression problem becomes a least square problem with augmented design matrix and response:

$$\widehat{\boldsymbol{\beta}}^{\text{ridge}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \left\| \overline{\boldsymbol{y}} - \overline{\boldsymbol{X}}\boldsymbol{\beta} \right\|_2^2 \tag{6}$$

Question (b) $\overline{\boldsymbol{X}}$ has full column rank becasue of the $\sqrt{\lambda}\boldsymbol{I}_p$ part. Consider linear combinations of columns of $\overline{\boldsymbol{X}}$:

$$\overline{\boldsymbol{X}}\boldsymbol{c} = \begin{pmatrix} \boldsymbol{X}\boldsymbol{c} \\ \sqrt{\lambda}\boldsymbol{c} \end{pmatrix} \in \text{span}\{\text{Col}(\overline{\boldsymbol{X}})\} \tag{7}$$

It's obvious that $\overline{\boldsymbol{X}}\boldsymbol{c} = \boldsymbol{0} \iff \boldsymbol{c} = \boldsymbol{0}$ given $\lambda > 0$. Hence the columns of $\overline{\boldsymbol{X}}$ are linearly independent, which implies $\overline{\boldsymbol{X}}$ has full column rank.

Question (c) Take advantage of (a), and the closed form formula for ordinary least square estimator:

$$\begin{aligned} \widehat{\boldsymbol{\beta}}^{\text{ridge}} &= (\overline{\boldsymbol{X}}^\top \overline{\boldsymbol{X}})^{-1} \overline{\boldsymbol{X}}^\top \overline{\boldsymbol{y}} = \left(\begin{pmatrix} \boldsymbol{X}^\top & \sqrt{\lambda}\boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{X} \\ \sqrt{\lambda}\boldsymbol{I}_p \end{pmatrix} \right)^{-1} \begin{pmatrix} \boldsymbol{X}^\top & \sqrt{\lambda}\boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{0} \end{pmatrix} \\ &= (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_p)^{-1} \boldsymbol{X}^\top \boldsymbol{y} \end{aligned} \tag{8}$$

And it follows that

$$\widehat{\boldsymbol{y}}^{\text{ridge}} = \boldsymbol{X}\widehat{\boldsymbol{\beta}}^{\text{ridge}} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_p)^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{9}$$

which can be regarded as $\boldsymbol{A}\boldsymbol{y}$, $\boldsymbol{A} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_p)^{-1} \boldsymbol{X}^\top$ is a non-random matrix given $\boldsymbol{X}$. This is clearly a linear function of $\boldsymbol{y}$. □

---

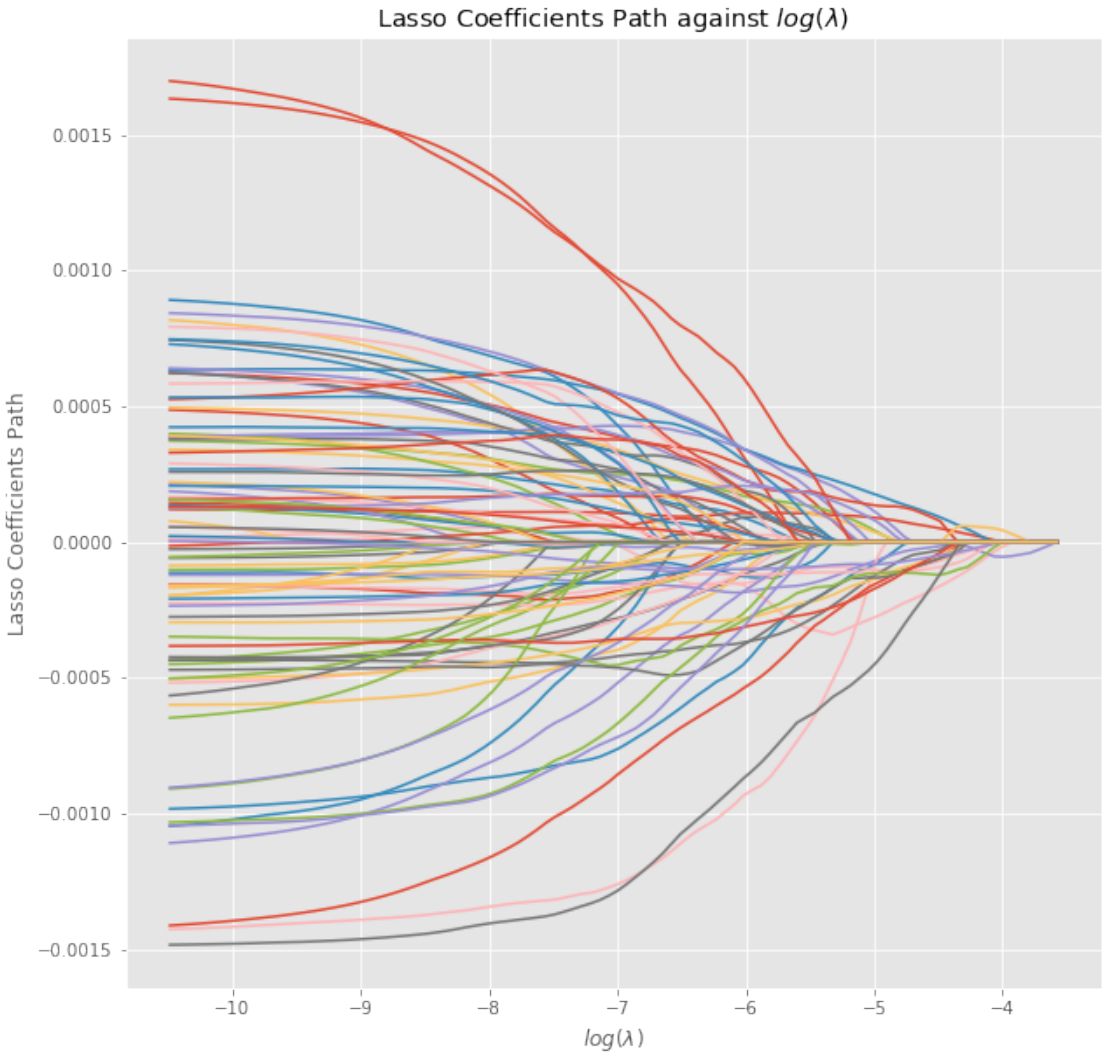
3 Google Trend Signal with LASSO

Question (a): The LASSO Coefficients Paths

```
In [7]: # load datasets
        train = pd.read_csv("trends_train.csv")
        test = pd.read_csv("trends_test.csv")
        dates_train = train.values[:,0]
        dates_test = test.values[:,0]
        X_train = np.asfarray(train.values[:,1:-1])
        y_train = np.asfarray(train.values[:,-1])
        X_test = np.asfarray(test.values[:,1:-1])
        y_test = np.asfarray(test.values[:,-1])
        words = train.columns[1:-1]
```

```
In [8]: # make the lasso coefficients path
        lambdas, coef_path, _ = lasso_path(X_train,y_train)
        fig, ax = plt.subplots(1, 1, figsize=(10,10))
```

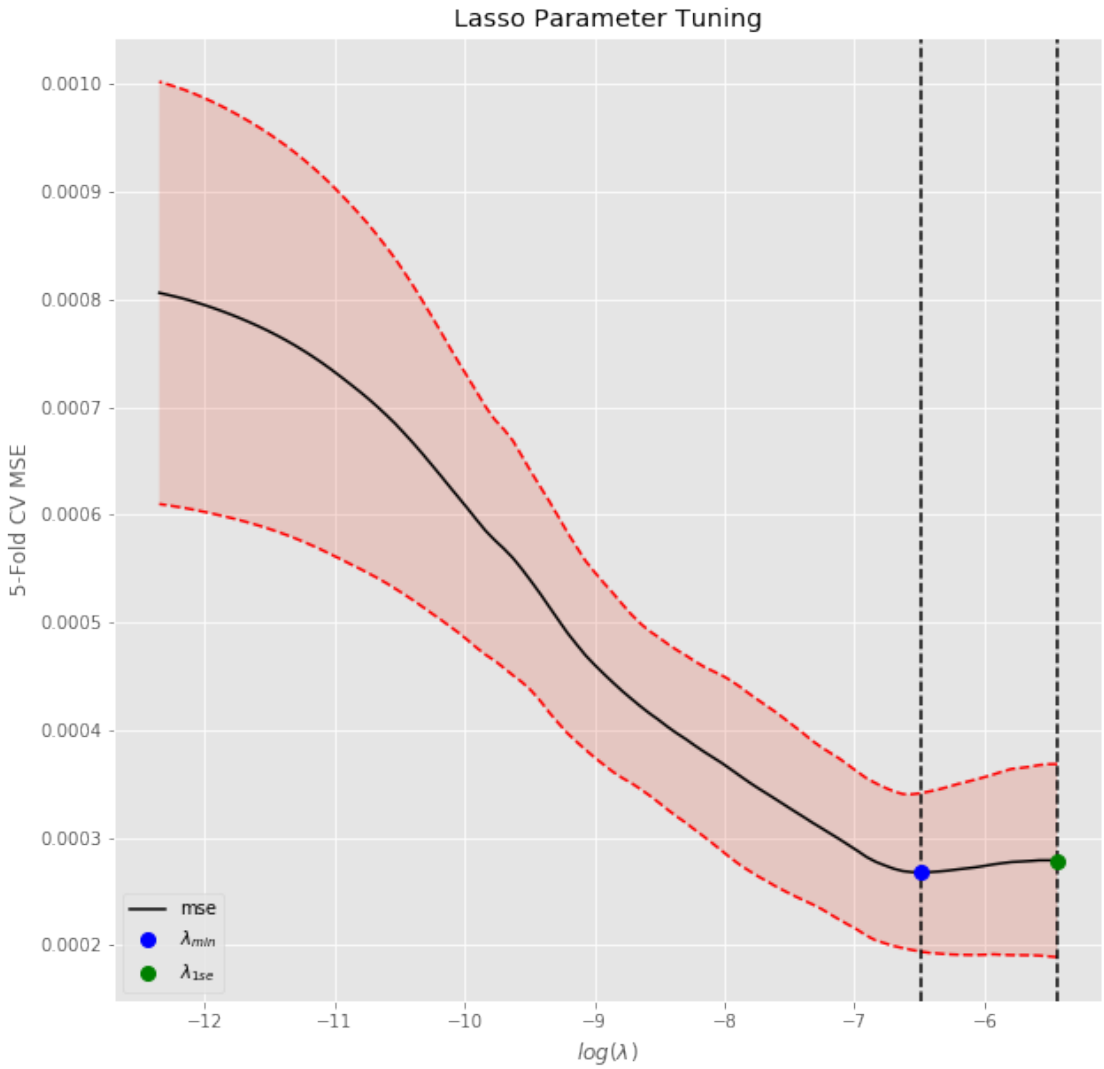
```
_ = ax.plot(np.log(lambdas), coef_path.T)
ax.set_title(r"Lasso Coefficients Path against $log(\lambda)$")
ax.set_xlabel(r"$log(\lambda)$")
_ = ax.set_ylabel("Lasso Coefficients Path")
```



Question (b): Parameter Tuning with Cross-Validation

```
In [9]: # preprocess data and fit model
        cv_lasso = LassoCV(cv=5, normalize=0, max_iter=3000)
        cv_clf = Pipeline([
            ('scl', StandardScaler()),
            ('lasso', cv_lasso),
        ])

        # sklearn.linear_model.LassoCV has a wired normalizing method,
        # it devides the columns of X by their l2 norm,
```

```
# instead of standard deviation. It's not a natural thing to do,
# so I used sklearn.preprocessing.StandardScaler() instead.

cv_clf.fit(X_train, y_train)

# find lambda_min and mse path
lambda_range = cv_lasso.alphas_
mse_values = cv_lasso.mse_path_.mean(axis = 1)
sd_values = cv_lasso.mse_path_.std(axis = 1)
lambda_min = cv_lasso.alpha_
lambda_min_idx = np.where(lambda_range == lambda_min)[0][0]
one_se = sd_values[lambda_min_idx]
mse_min = mse_values[lambda_min_idx]

# find lambda_1se
i = lambda_min_idx
while i >= 0:
    if mse_values[i] >= mse_min + one_se:
        break
    i -= 1
if i < 0: i = 0
lambda_1se = lambda_range[i]
mse_1se = mse_values[i]

# make the plot
fig, ax = plt.subplots(1, 1, figsize=(10,10))
ax.plot(np.log(lambda_range), mse_values ,
        "-", color='black',label='mse')
ax.plot(np.log(lambda_range), mse_values + sd_values,"r--")
ax.plot(np.log(lambda_range), mse_values - sd_values,"r--")
ax.fill_between(np.log(lambda_range),
                mse_values + sd_values,
                mse_values - sd_values, alpha = .2)
ax.axvline(np.log(lambda_min), ls='--', color='black')
ax.axvline(np.log(lambda_1se), ls='--', color='black')
ax.plot(np.log(lambda_min), mse_min, 'o', color='blue',
        markersize=8, label=r'$\lambda_{min}$')
ax.plot(np.log(lambda_1se), mse_1se, 'o', color='green',
        markersize=8, label=r'$\lambda_{1se}$')
ax.legend()
ax.set_title(r"Lasso Parameter Tuning")
ax.set_xlabel(r"$log(\lambda)$")
_ = ax.set_ylabel("5-Fold CV MSE")
```



Question (c,d): Fit Model on the Training Set

```
In [10]: # refit the model using all the training set
         clf_lambda_min = Pipeline([
             ('scl', StandardScaler()),
             ('lasso', Lasso(alpha=lambda_min, max_iter=2000))
         ])
         clf_lambda_min.fit(X_train,y_train)

         clf_lambda_1se = Pipeline([
             ('scl', StandardScaler()),
             ('lasso', Lasso(alpha=lambda_1se, max_iter=2000))
         ])
         clf_lambda_1se.fit(X_train,y_train)

         coef_min = clf_lambda_min.named_steps['lasso'].coef_
```

```
coef_1se = clf_lambda_1se.named_steps['lasso'].coef_
y_hat_train = clf_lambda_min.predict(X_train)

a, b = np.polyfit(y_hat_train, y_train, 1)
reg_line = np.linspace(-0.02, 0.02, 10)

fig, ax = plt.subplots(1, 1, figsize=(10,10))
ax.plot(y_hat_train, y_train, 'o')
ax.plot([-0.02,0.02], [-0.02,0.02], '--',
        color='black', label=r'Perfect Prediction, $y=\hat{y}$')
ax.plot(reg_line, reg_line*a + b, 'red', label='Regression Line')
ax.set_ylabel(r"true $y$")
ax.set_xlabel(r"predicted $\hat{y}$")
ax.set_title('Prediction v.s. Ground Truth (Training Set)')
_ = ax.legend()
```

```
In [11]: print("Variables in lambda_min model: ",
                [_ for _ in words[np.where(coef_min != 0)]], '\n')
         print("Variables in lambda_1se model: ",
                [_ for _ in words[np.where(coef_1se != 0)]])
```

```
Variables in lambda_min model:  ['growth', 'dow', 'invest', 'leverage', 'cash',
'nyse', 'sell', 'returns', 'present', 'rich', 'house', 'tourism', 'holiday',
'health', 'fine', 'marriage', 'restaurant']

Variables in lambda_1se model:  []
```

- **Question (c):** As suggested above, λ_{min} leads us to a model with 17 variables, while λ_{1se} leads us to an empty model. It's worth to notice that in the 5-folds cross validation MSE plot, there is a flat basin around large λ's. The difference between the cross-validation MSE at λ_{min} and λ_{1se} is so small that the $MSE_{CV}(\lambda_{1se})$ does not really exceed the 1-standard-error threshold, we just truncate it off at λ_{1se} because the MSE_{CV} won't reduce any more beyond that for greater λ's, and the model is empty from that point.

  These observation suggests there is very few useful signal (17 out of 98), and those signals picked by λ_{min} is likely to be too weak to survive the variance on an data set that the model has never seen.

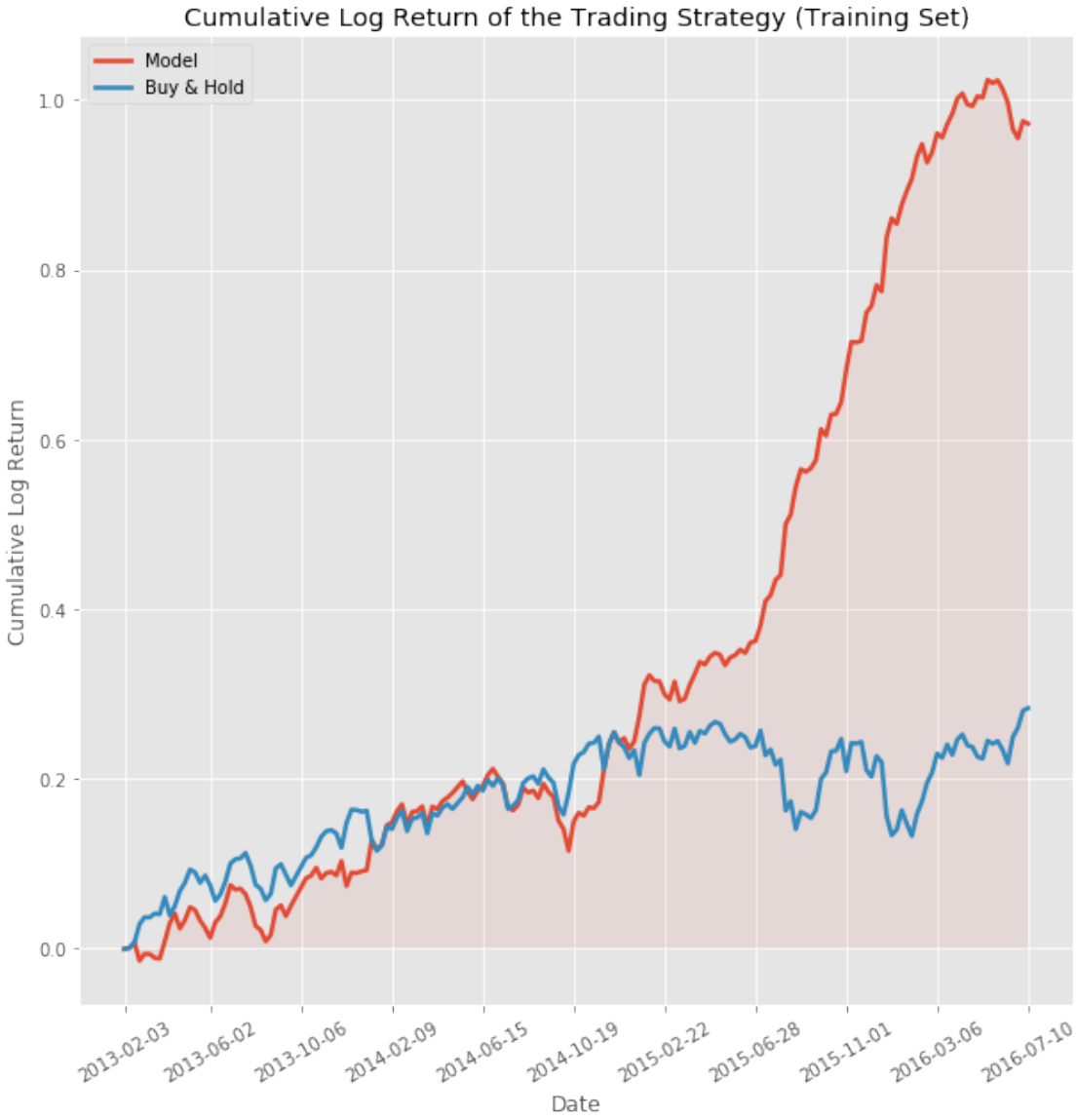
- **Question (d):** The scatter plot is overlaid with two lines: an imagined perfect prediction line $y = \hat{y}$ (black dashed), and the regression line for all the $(y, \hat{y})$ we obtained. We can see that there is a somewhat of positive linear correlation between y and $\hat{y}$ on the training set.

Question (e): We run the simple trading strategy on the training set as below.

```
In [12]: # The simple trading stategy on training set
         cum_return = np.cumsum(np.sign(y_hat_train) * y_train)
         buy_and_hold = np.cumsum(y_train)

         fig, ax = plt.subplots(1, 1, figsize=(10,10))
         ax.plot(cum_return, linewidth=2.5, label='Model')
         ax.plot(buy_and_hold, linewidth=2.5, label='Buy & Hold')
         ax.fill_between(range(len(y_train)), cum_return,
                         [0]*len(y_train), alpha = .08)
         indices = np.arange(-1, len(y_train), 18); indices[0] = 0
         ax.set_xticks(indices)
         ax.set_xticklabels(dates_train[indices], rotation=30)
         ax.set_title("Cumulative Log Return of the Trading Strategy (Training Set)")
         ax.set_xlabel("Date")
         ax.set_ylabel("Cumulative Log Return")
         _ = ax.legend()

         print("Cumulative log return using the model prediction:", cum_return[-1])
         print("Cumulative log return of buy and hold:", buy_and_hold[-1])
```

```
Cumulative log return using the model prediction: 0.972016182024
Cumulative log return of buy and hold: 0.283038084882
```

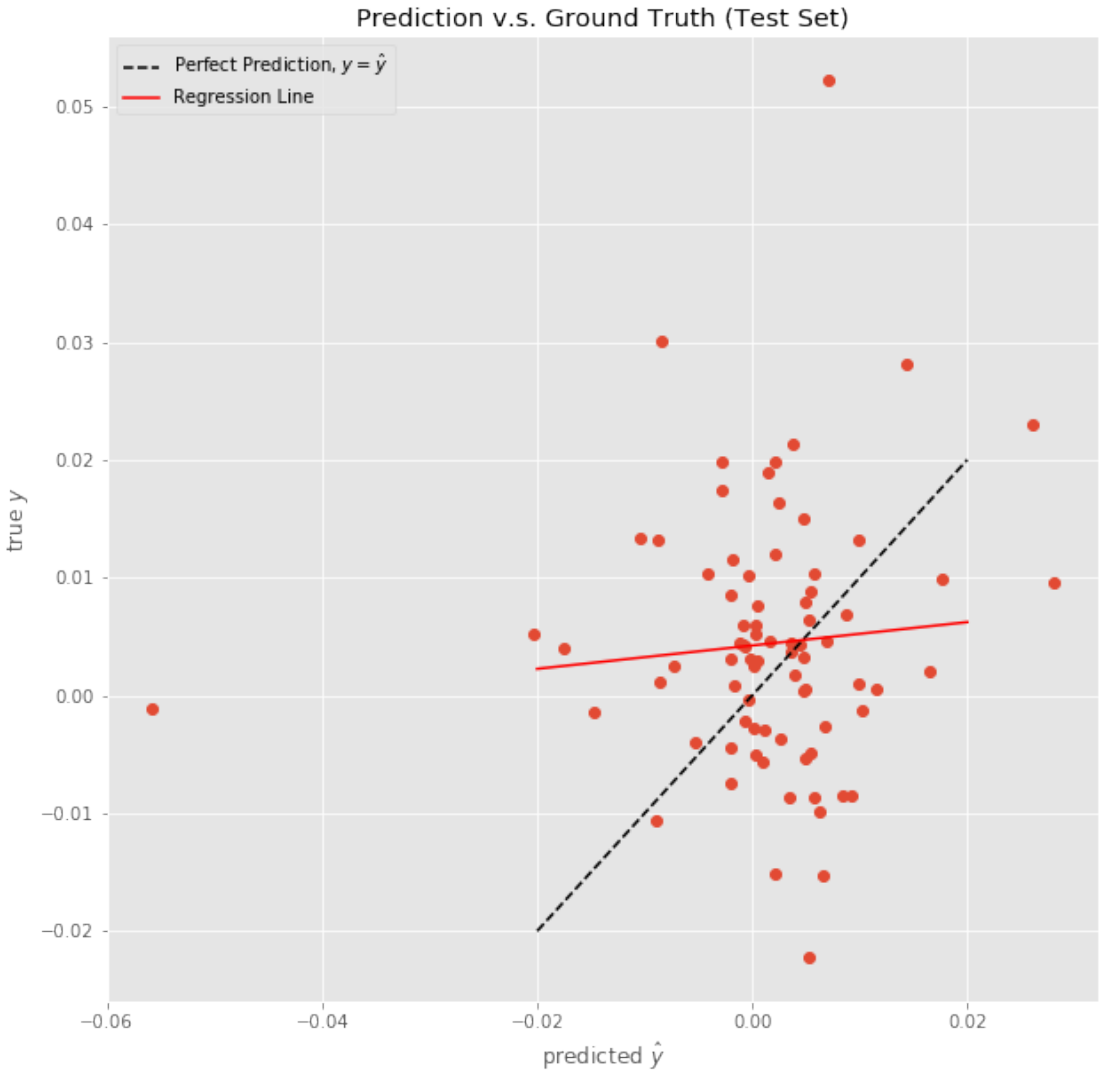
The cumulative log return using the model is 0.9720, which is about 264.3% cumulative return from Febueary 2013 to July 2016. This is indeed much higher than the "buy and hold" cumulative return, which is 0.2830 on log scale, and about 132.7% on 1 scale.



Question (f): Fit the model and carry out simple trading strategy on the test set.

```
In [13]: # make prediction on test set
         y_hat_test = clf_lambda_min.predict(X_test)
         a, b = np.polyfit(y_hat_test, y_test, 1)
         reg_line = np.linspace(-0.02, 0.02, 10)
```

```
fig, ax = plt.subplots(1, 1, figsize=(10,10))
ax.plot(y_hat_test, y_test, 'o')
ax.plot([-0.02,0.02], [-0.02,0.02], '--',
        color='black', label=r'Perfect Prediction, $y=\hat{y}$')
ax.plot(reg_line, reg_line*a + b, 'red', label='Regression Line')
ax.set_ylabel(r"true $y$")
ax.set_xlabel(r"predicted $\hat{y}$")
ax.set_title('Prediction v.s. Ground Truth (Test Set)')
_ = ax.legend()
```

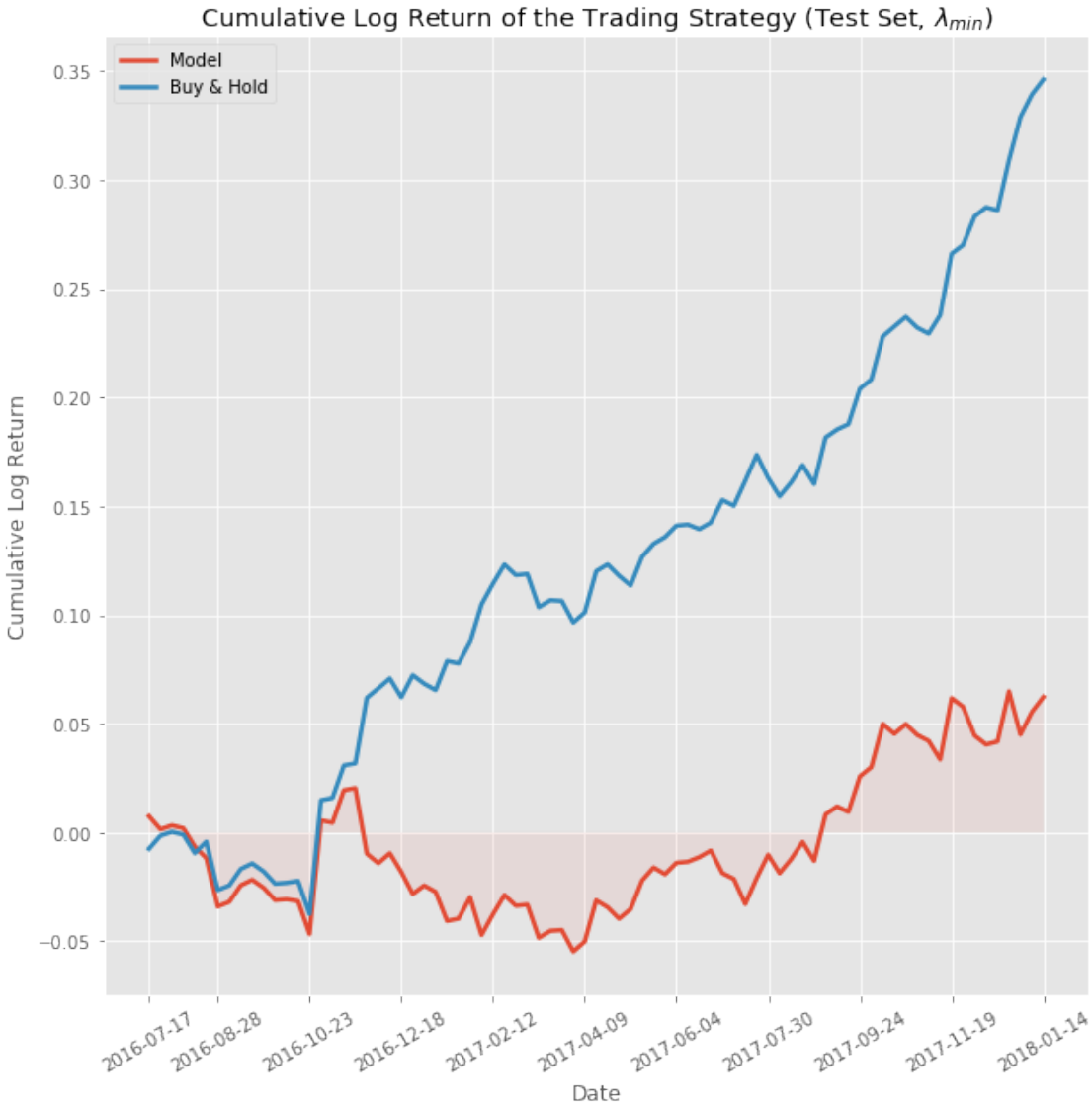


In [14]:
```
# The simple trading stategy on test set
cum_return = np.cumsum(np.sign(y_hat_test) * y_test)
buy_and_hold = np.cumsum(y_test)
```

```
fig, ax = plt.subplots(1, 1, figsize=(10,10))
ax.plot(cum_return, linewidth=2.5, label='Model')
ax.plot(buy_and_hold, linewidth=2.5, label='Buy & Hold')
ax.fill_between(range(len(y_test)), cum_return,
                [0]*len(y_test), alpha = .08)
indices = np.arange(-2, 79, 8); indices[0] = 0
ax.set_xticks(indices)
ax.set_xticklabels(dates_test[indices], rotation=30)
ax.set_title(r"Cumulative Log Return of the Trading Strategy (Test Set, $\lambda_{min}$
ax.set_xlabel("Date")
ax.set_ylabel("Cumulative Log Return")
_ = ax.legend()

print("Cumulative log return using the model prediction:", cum_return[-1])
print("Cumulative log return of buy and hold:", buy_and_hold[-1])
```

```
Cumulative log return using the model prediction: 0.0623050794901
Cumulative log return of buy and hold: 0.346153588946
```

- The model has poor performance when extrapolated to the test set. First off, we can see from the scatter plot that the correlation between y and $\hat{y}$ is minimal, the points are scattered in randomly, and the regression line has nearly no slope.
- The trading strategy also behaved poorly, it generates only 0.0623 cumulative log return, beaten by the buy-and-hold return: about 0.3462.
- When we use the λ_{1se} to fit model, it again selects an empty model. Indeed, we are better off since the trading strategy with an empty model is equivalent to buy and hold, but this is not a discovery at all.



In [15]:
```
y_hat_test_1se = clf_lambda_1se.predict(X_test)

# The simple trading stategy on test set (using lambda_1se)
fig, ax = plt.subplots(1, 1, figsize=(10,10))
ax.plot(np.cumsum(np.sign(y_hat_test_1se) * y_test),
        linewidth=2.5, label='Model')
ax.plot(np.cumsum(y_test), linewidth=2.5, label='Buy & Hold')
ax.fill_between(range(len(y_test)),
                np.cumsum(np.sign(y_hat_test_1se) * y_test),
                [0]*len(y_test), alpha = .08)
indices = np.arange(-2, 79, 8); indices[0] = 0
ax.set_xticks(indices)
ax.set_xticklabels(dates_test[indices], rotation=30)
```

```
ax.set_title(r"Cumulative Log Return of the Trading Strategy (Test Set, $\lambda_{1se}$
ax.set_xlabel("Date")
ax.set_ylabel("Cumulative Log Return")
_ = ax.legend()

print("Cumulative log return using the model prediction:",
      np.cumsum(np.sign(y_hat_test_1se) * y_test)[-1])
print("Cumulative log return of buy and hold:", buy_and_hold[-1])
```

```
Cumulative log return using the model prediction: 0.346153588946
Cumulative log return of buy and hold: 0.346153588946
```