Problem 3.

This problem uses the Weekly.csv dataset (uploaded on Bruinlearn) containing 1089 weekly stock returns for 21 years.

(a) Use the full dataset to fit a logistic regression of today's stock movement (up or down) on the five lags of returns and the trading volume.

(b) Calculate the confusion matrix, accuracy, precision, recall, and F1 score for the in-sample predictions. Does the model uniformly beat random guessing in terms of these performance metrics?

(c) On the same graph, plot precision and recall against the threshold (varying over $[0, 1]$) used to generate predicted labels from predicted probabilities. Explain the pattern you see.

(d) Now fit the logistic regression using only data up to (and including) the year 2008, with Lag2 as the only predictor.

(e) Repeat (b) using the remaining observations as a test sample.

(f) Which of the two fitted models would you use for real-time stock return prediction?

ML HW 2

January 24, 2024

0.1 Problem 3

0.1.1 a)

[41]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
```

[42]:
```
df = pd.read_csv("/Users/sandinatatu/Desktop/Weekly.csv")
```

Below I create another column in the dataset ("uod") which has the value 1 if it is on a row where the Direction is Up and 0 if it is on a row where the Direction is Down.

[43]:
```
df['uod'] = df['Direction'].replace({'Up':1, 'Down':0})
y= df['uod']
X=df.loc[:,'Lag1': 'Volume']
```

I then fit a Logistic Regression on the enitre dataset using the uod columns as the dependent variable and Lags 1 through 5, as well as the Volume, as the independent variables. I also compute the accuracy of this Logistic Regression.

[44]:
```
logreg = LogisticRegression()
logreg.fit(X,y)
y_pred = logreg.predict(X)
```

[45]:
```
print("Accuracy:",logreg.score(X,y))
```

```
Accuracy: 0.5610651974288338
```

0.1.2 b)

Below I compute the confusion matrix. We have 49 True Negaties, 40 False Negatives, 565 True Pozitives and 435 False Pozitives.

[46]:
```
cm=confusion_matrix(y, y_pred)
tn, fp, fn, tp =cm.ravel()
print(cm)
print(tn, fp, fn, tp)
```

```
[[ 54 430]
 [ 48 557]]
54 430 48 557
```

The accuracy of the model is ~56.4%. This, combined with a precision of ~0.57, a Recall of ~0.93 and an F1 Score of ~0.7, means that our model is performing better than random guessing.

[47]:
```
print('Accurracy:', (tp+tn)/1089)
print('Precision:', tp/(tp+fp))
print('Recall:', tp/(tp+fn))
precision = tp/(tp+fp)
recall = tp/(tp+fn)
print('F1 Score:', 2*precision*recall/(precision+recall))
```

```
Accurracy: 0.5610651974288338
Precision: 0.5643363728470111
Recall: 0.9206611570247933
F1 Score: 0.6997487437185929
```

0.1.3 c)

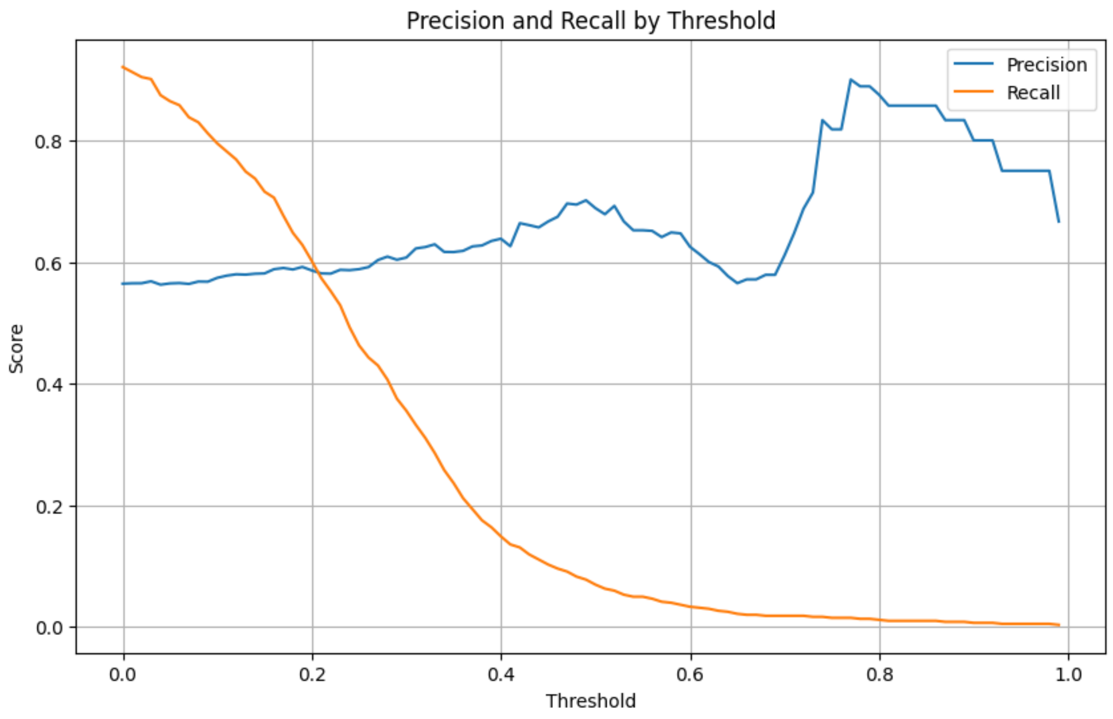
Below I compute the precision and recall scores for 100 different thesholds between 0 and 1.

[48]:
```
precisions=[]
recalls=[]
thresholds=[]
for threshold in np.arange(0,1,.01):
    df_temp=df.copy()
    df_temp['pred'] = np.where(logreg.decision_function(X)>threshold,1,0)
    FN=len(df_temp[(df_temp['pred'] == 0) & (df_temp['uod']==1)])
    TN=len(df_temp[(df_temp['pred'] == 0) & (df_temp['uod']==0)])
    TP=len(df_temp[(df_temp['pred'] == 1) & (df_temp['uod']==1)])
    FP=len(df_temp[(df_temp['pred'] == 1) & (df_temp['uod']==0)])
    precisions.append(TP/(TP+FP))
    recalls.append(TP/(TP+FN))
    thresholds.append(threshold)
```

I then plot the precision and recall scores for each of this thresholds. Generally, we notice that the higher the threshold, the higher the precision, until the threshold becomes very large. This makes sense, as a higher threshold generally decreases the risk of false pozitives, resulting in an increase in the precision. However, a very high threshold increases the risk of not correctly identifying true observations.

However, a very high threshold also increases the risk of false negatives. Thus, the recall decreases as the threshold increases, which can also be observed from the graph.

[49]:
```
plt.figure(figsize=(10, 6))
plt.plot(thresholds, precisions, label="Precision")
plt.plot(thresholds, recalls, label="Recall")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Precision and Recall by Threshold")
plt.legend()
plt.grid(True)
plt.show()
```



0.1.4 d)

Below I create training and testing datasets. I will be fitting the model by having the “uod” column up to and including 2008 as the dependent variable, and the “Lag2” column up to and including 2008 as the independent variable.

[50]:
```
ytrain = df[df['Year']<=2008][['uod']]
Xtrain=df[df['Year']<=2008][['Lag2']]
ytest = df[df['Year']>2008][['uod']]
Xtest=df[df['Year']>2008][['Lag2']]
```

[53]:
```
import warnings
warnings.filterwarnings("ignore")

logreg.fit(Xtrain,ytrain)
y_pred = logreg.predict(Xtest)
```

Below I calculate the confusion matrix for the test sample. I notice that I have 9 true negatives, 34 false pozitives, 5 false negatives, and 56 true pozitives.

[52]:
```
cm=confusion_matrix(ytest, y_pred)
tn, fp, fn, tp =cm.ravel()
print(cm)
```

```
[[ 9 34]
 [ 5 56]]
```

0.1.5 e)

This model also beats random geassing. This is due to its accuracy of ~0.63, its precision of ~0.62, the recall of ~0.92 and tthe F1 Score of ~0.74 .

[35]:
```
print('Accurracy:', (tp+tn)/104)
print('Precision:', tp/(tp+fp))
print('Recall:', tp/(tp+fn))
precision = tp/(tp+fp)
recall = tp/(tp+fn)
print('F1 Score:', 2*precision*recall/(precision+recall))
```

```
Accurracy: 0.625
Precision: 0.6222222222222222
Recall: 0.9180327868852459
F1 Score: 0.7417218543046358
```

0.1.6 f)

The second model has better scores than the first one (with the exception of recall), so overall I would prefer the seconf model for real-time stock return prediction.