Machine Learning Homework

Problem Set 3

Submitted by Gaurav Shah (gns274)

Install the package rugarch in an R session and import the dataset dji30ret (showing the Dow Jones 30 Constituents daily log returns from March 16 1987 to February 3 2009) into a data frame. Split your sample into a Training Set: from March 16, 1987 to December 31, 2002 and a Test Set from January 1st, 2003 to February 3, 2009. Install Keras and Tensorflow to your computer and make sure you can use it to estimate Neural Networks.

1. Using the Training Set, fit an a 30-20 Neural Network (i.e. 2 hidden layers, the first with 30 units and the second with 20 units) to predict the t + 1 individual stock return. In this part of the exercise do not use Dropout, instead use a validation set and choose an early stopping in terms of the number of epochs. Choose the optimization algorithm you think is appropriate and a mini-batch size of 20. Report the mean absolute error for the test set.

Solution: Mean absolute error is **0.128566**

```
print("The mean absoulte error is :",mean_absolute_error(y_test,y_pred_model1))
```

```
The mean absoulte error is : 0.012856670606135746
```

2. Using the Training Set, fit a 15-10-5 Neural Network to predict the t + 1 individual stock return. This time use a Dropout rate of 0.3 and no validation. Report the mean absolute error for the test set.

Solution: Mean absolute error is **0.126311**

```
print("The mean absoulte error is :",mean_absolute_error(y_test,y_pred_model2))
```

```
The mean absoulte error is : 0.012631173459715548
```

3. Using the Training Set, fit 15-10-5 Neural Network to predict the t+ 1 individual stock return. This time do not use Dropout but instead use L2 regularization. Use cross-validation to choose the appropriate regularization parameter. Explain how you performed cross validation to avoid leakage. Report the mean absolute error for the test set.

Solution: Mean absolute error is **0.127423**

```
print("The mean absoulte error is :",mean_absolute_error(y_test,y_pred_model3))
```

```
The mean absoulte error is : 0.012742328417695684
```

Regularization has been implement through **keras.regularizers**

Cross validation: lambda used are 0.001, 0.01, 0.1, 1

Cross validation has been done keeping in mind to avoid leakage of data. Walk forward times series split has been performed. Training data is always chronologically behind the test data, i.e. the latest training

data is behind the earliest test data in a time series point of view. Also it has been made sure that no cross sectional leakage takes place in cross validating.

4. Using the Training Set, fit 15-10-5 Neural Network to predict the t+ 1 individual stock up/down move (1 for positive return and 0 for negative return). Again, use a Dropout rate of 0.3 and no validation. Report the classification accuracy for the test set. (Note that this is a classification problem.)

Solution: Classification accuracy is 49.6096%

```
In [56]: print('The accuracy is:', accuracy_score(y_pred_convert,y_test_convert)*100)
         The accuracy is: 49.60966954978931
```

5. Compare the mean absolute error of the different estimated models on the Test Set. What are your conclusions?

   Mean Absolute Error and Accuracy for the Models are summarized below:

   Model 1 Mean Absolute Error: 0.12856
   Model 2 Mean Absolute Error: 0.12631
   Model 3 Mean Absolute Error: 0.12742
   Model 4 Accuracy: 49.6096%

   We can see a drop in Mean Absolute Error in '2' and '3' as compared to '1' as we implement ways to train our model and avoid overfitting. Our model learns better by using Dropout and L2 regularization. But one would also note the extra time taken by '2' as compared to '1'

   In '1', we use a 20-80 validation-train set and mini-batching to help us train a better model. The model trains on 80% of the data and validation on 20% of data helps to optimize performance. In '2', we use the concept of dropout at 30%. Adding an extra hidden layer improves the performance and we can see that error goes down despite reducing the number of nodes. Dropout helps to avoid overfitting and co-adaptation which is very important in financial datasets which are noisy in nature. In '3', we use L2 regularization through which we add a penalty term. Thus, we achieve a lower error than '1' but cannot beat the error from model '1'.

APPENDIX

```
import pandas as pd
import numpy as np
import keras
import tensorflow
from sklearn.metrics import mean_absolute_error
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras import layers
from keras import regularizers
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score

data = pd.read_excel('C:\\Users\\shahg_000\\Downloads\\data.xlsx')
df=data.set_index('Date')

TrainingSet=df.loc[:'2002-12-31',:]
TestSet=df.loc['2003-01-01':,:]

def Stacking(toBeStacked,p):
  n=p+1
  stackedF=pd.DataFrame()
  temp=np.zeros(shape=(1,n))
  for cnum in range(toBeStacked.shape[1]):
    temp1=np.zeros(shape=(1,n))
    for rnum in range(n,len(toBeStacked)+1):
      temp1=np.append(temp1,[np.flip(np.array(toBeStacked.iloc[rnum-n:rnum,cnum]))],axis=0)
    temp1=np.delete(temp1,(0),axis=0)
    temp=np.append(temp,temp1,axis=0)
  return(pd.DataFrame(np.delete(temp,(0),axis=0)))

def DummyVariables(toBeDummy):
  p=int(toBeDummy.shape[0]/30)
  finalDataframe = pd.DataFrame()
  for i in range(0,29):
    dummyframes = pd.DataFrame(data = 0, index=range(p),columns=range(28))
    dummyframes.insert(i, "DV"+str(i), 1)
    finalDataframe = finalDataframe.append(dummyframes)
```

```
    dvAllZero = pd.DataFrame(data = 0, index=range(p),columns=range(28))
    finalDataframe = finalDataframe.append(dvAllZero, ignore_index=True)
    finalDataframe.drop(finalDataframe.iloc[:, 0:28], inplace = True, axis = 1)
    finalDataframe = finalDataframe.fillna(0)
    return(pd.concat([toBeDummy, finalDataframe], axis=1))

TrainingDummyStacked=DummyVariables(TrainingStacked)
TestDummyStacked=DummyVariables(TestStacked)

TrainingDummyStacked.head()

TrainingDummyStacked.describe()

y=TrainingDummyStacked.loc[:,0]
X=TrainingDummyStacked.drop([0],axis=1)
y_test=TestDummyStacked.loc[:,0]
X_test=TestDummyStacked.drop([0],axis=1)


# Q.1

model = Sequential()
model.add(Dense(30, activation='relu', input_shape=(59,)))
model.add(Dense(20, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

early_stopping_monitor = EarlyStopping(patience=5)
model.fit(X,y, validation_split=0.2, epochs=30,
        callbacks=[early_stopping_monitor],batch_size=20)

y_pred_model1=model.predict(X_test)
print("The mean absoulte error is :",mean_absolute_error(y_test,y_pred_model1))


# Q.2

model2 = Sequential()
model2.add(Dense(15, activation='relu', input_shape=(59,)))
model2.add(layers.Dropout(0.3, input_shape=(59,)))
model2.add(Dense(10, activation='relu'))
model2.add(layers.Dropout(0.3, input_shape=(59,)))
model2.add(Dense(5, activation='relu'))
model2.add(layers.Dropout(0.3, input_shape=(59,)))
```

```
model2.add(Dense(1))

model2.compile(optimizer='adam', loss='mean_squared_error')

model2.fit(X,y, epochs=30, callbacks=[early_stopping_monitor],batch_size=20)

y_pred_model2=model2.predict(X_test)
print("The mean absoulte error is :",mean_absolute_error(y_test,y_pred_model2))


# Q.3

model3 = Sequential()
#Credits: https://gist.github.com/mustafa-qamaruddin
class BlockingTimeSeriesSplit():
   def __init__(self, n_splits):
      self.n_splits = n_splits

   def get_n_splits(self, X, y, groups):
      return self.n_splits

   def split(self, X, y=None, groups=None):
      n_samples = len(X)
      k_fold_size = n_samples // self.n_splits
      indices = np.arange(n_samples)

      margin = 0
      for i in range(self.n_splits):
         start = i * k_fold_size
         stop = start + k_fold_size
         mid = int(0.8 * (stop - start)) + start
         yield indices[start: mid], indices[mid + margin: stop]
def l2_regularization_param(X, y):
   btscv = BlockingTimeSeriesSplit(n_splits=5)
   gammas = [0.001, 0.01, 0.1, 1]
   param_grid = {keras.regularizers.l2() : gammas}
   grid_search = GridSearchCV(model3.add(Dense(15, activation='relu', input_shape=(59,)),
param_grid, cv=btscv)
   grid_search.model3.add(X,y)
   grid_search.best_param_
   return grid_search.best_param_
l2_regularization_param(X,y)
```

```
model3 = Sequential()
model3.add(Dense(15,                                                    activation='relu',
input_shape=(59,),kernel_regularizer=keras.regularizers.l2(0.001)))
model3.add(Dense(10, activation='relu',kernel_regularizer=keras.regularizers.l2(0.001)))
model3.add(Dense(5, activation='relu',kernel_regularizer=keras.regularizers.l2(0.001)))
model3.add(Dense(1))

model3.compile(optimizer='adam', loss='mean_squared_error')

model3.fit(X,y,epochs=30, callbacks=[early_stopping_monitor],batch_size=20)

y_pred_model3=model3.predict(X_test)
print("The mean absoulte error is :",mean_absolute_error(y_test,y_pred_model3))


# Q.4

y_binary=y
y_test_binary=y_test

y_binary[y_binary<=0]=0
y_binary[y_binary>0]=1
y_test_binary[y_test_binary<=0]=0
y_test_binary[y_test_binary>0]=1

ohe = OneHotEncoder()
y_binary = ohe.fit_transform(np.array(y_binary).reshape(-1,1)).toarray()
y_test_binary=ohe.fit_transform(np.array(y_test_binary).reshape(-1,1)).toarray()

model4 = Sequential()
model4.add(Dense(15, activation='relu', input_shape=(59,)))
model4.add(layers.Dropout(0.3, input_shape=(59,)))
model4.add(Dense(10, activation='relu'))
model4.add(layers.Dropout(0.3, input_shape=(59,)))
model4.add(Dense(5, activation='relu'))
model4.add(layers.Dropout(0.3, input_shape=(59,)))
model4.add(Dense(2,activation='softmax'))


model4.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])

model4.fit(X,y_binary, epochs=30, callbacks=[early_stopping_monitor],batch_size=20)

y_pred_model4=model4.predict(X_test)
```

```
#Converting predictions to label
y_pred_convert = list()
for i in range(len(y_pred_model4)):
   y_pred_convert.append(np.argmax(y_pred_model4[i]))
#Converting one hot encoded test label to label
y_test_convert = list()
for i in range(len(y_test_binary)):
   y_test_convert.append(np.argmax(y_test_binary[i]))


print('The accuracy is:', accuracy_score(y_pred_convert,y_test_convert)*100)


# END
```