NN modelling with oversampling

September 6, 2020

1 How to correct probabilities in NNs?, By Brent Oeyen

In case of oversampling or undersampling, the expected values of predicted probabilities by a Neural Network (NN) will no longer reflect the actual likelihood of events. For this reason, it is important to adjust the weights of events in case a bias exists in the training dataset. In the subsequent steps a straight forward example is provided how to eliminate a bias in the training dataset with respect to the probability of an event occurring. ## Load libraries

[76]:
```
import pandas                 as     pd
import numpy                  as     np
from scipy.stats              import norm
from tensorflow.keras.models  import Sequential
from tensorflow.keras.layers  import Dense
```

1.1 Create a data set with an explanatory variable x which is correlated with a binary dependent variable y

A data set of 40K observations is created from a simulated variable X that follows a standard normal distribution and y binary variable {0, 1} with E[y]=1%.

[81]:
```
N       = 40000;
z       = np.random.normal(0, 1, N)
dataset = pd.DataFrame({'x': z, 'y': norm.ppf(0.99)<(1-.5**2)**.5*np.random.
 ↪normal(0, 1, N)+.5*z})
dataset.head()
```

[81]:

	x	y
0	0.225416	False
1	-0.227001	False
2	1.225429	False
3	0.338792	False
4	-0.084162	False

1.2 Create sample for the independent (X) and dependent variable (y)

Considering the small amount of observations for which the event Y=1 is true, a training dataset is created by retaining all observations with events for which Y=1 and all observations for which the

index (row number) is an odd number. Hence, an example of oversampling observations for which the target variable's class equals to 1.

[82]:
```
X = dataset.x[(dataset.y==1) | (dataset.index % 2 ==0)]
y = dataset.y[(dataset.y==1) | (dataset.index % 2 ==0)]
```

1.3 Define and compile the NN model

[83]:
```
model = Sequential()
model.add(Dense(22, input_dim  = 1, activation='relu'))
model.add(Dense(22, activation ='relu'))
model.add(Dense(1, activation ='sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics =␣
 ↪['accuracy'])
```

1.4 Fit the NN model to the dataset

[85]:
```
model.fit(X, y, epochs = 20, verbose=0, batch_size = 20)
```

[85]:
```
<tensorflow.python.keras.callbacks.History at 0x1a450641d0>
```

1.5 Backtest the keras model

[86]:
```
_, accuracy = model.evaluate(X, y)
print('Accuracy model for training data set: %.2f' % (accuracy*100))
print('Expected value target for training data set: %.2f' % (model.
 ↪predict_proba(X).mean()*100))
print('Expected value model for training data set: %.2f' % (y.mean()*100))
print('Expected value target full data set: %.2f' % (dataset.y.mean()*100))
```

```
Accuracy model for training data set: 97.98
Expected value target for training data set: 2.51
Expected value model for training data set: 2.02
Expected value target full data set: 1.02
```

The above statistics demonstrates that the NN model calibrates probabilities in such a way that the expected value of the probabilities equals the expected value of y in the training data set. In case the expected value of the probabilities are not comparable with the expected value of the target in the training dataset, it is considered a bad fit. Since class 1 has been oversampled, it is preferred to modify the fit in such a way that the expected value of the probabilities equals that of the full dataset. ## Fitting the NN model with redefined weights of the class values Lets refit the NN model but this time allocate a weight of 1 to class 0 and a weight of E[y]/E[y|x] (i.e. divide the expected value of the training dataset with that of the total population) to class 1 of the y variable.

[88]:
```
model.fit(X, y, epochs = 20, verbose=0, batch_size = 20, class_weight={0:1, 1:
 ↪dataset.y.mean()/y.mean()})
_, accuracy = model.evaluate(dataset.x, dataset.y)
```

```
print('Expected value model, with class weights, for training data set: %.2f' %␣
 ↪(model.predict_proba(X).mean()*100))
print('Accuracy model, with class weights, for entire data set: %.2f' %␣
 ↪(accuracy*100))
```

```
Expected value model, with class weights, for training data set: 1.09
Accuracy model, with class weights, for entire data set: 98.98
```

1.6 Conclusion

The probabilities calculated by a NN can easily be modified so that the probability of a given event/class match that of all observation and not the observations chosen for a training dataset.