End-to-end example of Supervised learning

Dr Rob Collins

Version 5, 19th July 2021

Introduction

In this workshop session we will be creating a loan credit decision model. That is, if we lend money to somebody - then what is the probability that the money will be paid back?

The model uses a very well known data-set called the "German Credit Data" which is widely available on the Internet. The version of the data supplied to complete this workshop contains some ommisions - so it will beed to be cleaned before use.

The model we will build is a 'logisitc regression'. Logistic Regression is a common choice when it is required to precict probabilities from regression models - since the results of the model are in the range 0 to 1.

Instructions for Students

This workbook includes some empty code-blocks. In those cases you are required to create the code for the block based on your previous learning on this course. This activity is indicated in the workbook below with a '**Student Task**' indicator.

In working through the following notebook, please do the following:

1. Create an empty Jupyter notebook on your own machine
2. Enter all of the **Python code** from this notebook into **code blocks** in your notebook
3. **Execute each of the code blocks** to check your understanding
4. You **do not need to** replicate all of the explanatory / tutorial text in text (markdown) blocks
5. You **may** add your own comments and description into text (markdown) blocks if it helps you remember what the commands do
6. You **may** add further code blocks to experiment with the commands and try out other things
7. Enter and run as many of the code blocks as you can within the time available during class
8. **After class**, enter and run any remaining code blocks that you have not been able to complete in class

The numbers shown in the 'In [n]' text on your Jupyter notebook are likely to be different to the ones shown here because they are updated each time a block is executed.

Getting data into the tool

We frequently use a powerful tool called 'Pandas' to manipulate data. This tool is not built into Python .. so we need to 'import' it before we start

In [1]:
```python
import pandas as pd
```

Having done that, we can use the tool to get the data from a '.csv' file into Python

In [2]:
```python
credit_data = pd.read_csv("german credit data unclean.csv")
```

Do a basic review of the data

Let's get a feel for what this data looks like in Python.

Since we are working in Jupyter - the tool will display a neatly formatted table of data.

Here is have decided to display only the first 20 rows of data .. you may experiment with this to display different portions of the data.

In [3]:
```python
credit_data[0:12]
```

Out[3]:

	check_account_status	duration	credit_history	purpose	credit_amount	savings_account	employment_duration	percent_disposable_income
0	A11	6	A34	A43	1169.0	A65	A75	4
1	A12	48	A32	A43	5951.0	A61	A73	2
2	A14	12	A34	A46	2096.0	A61	A74	2
3	A11	42	A32	A42	7882.0	A61	A74	2
4	A11	24	A33	A40	4870.0	A61	A73	3
5	A14	36	A32	A46	9055.0	A65	A73	2
6	A14	24	A32	A42	2835.0	A63	A75	3
7	A12	36	A32	A41	6948.0	A61	A73	2
8	A14	12	A32	A43	3059.0	A64	A74	2
9	A12	30	A34	A40	5234.0	A61	A71	4
10	A12	12	A32	A40	1295.0	A61	A72	3
11	A11	48	A32	A49	NaN	A61	A72	3

12 rows × 21 columns

You may notice immediately there there is some missing data in this table (Row 11 of the 'credit_amount' feature). This is important and gives us a clue that we need to clean and tidy the data.

Step 1 : Clean and Tidy Data

1.1 FInd missing data

First we need to check if there is any missing data (NaN).

Student Task: Add a function in the following code block that provides a count of the total number of NaNs in the data table. Note that you were shown how to do this during the Session 1 tutorial.

In [ ]:

If we get a count of zero, then we do not have any missing data. However, in this case we do not get zero - so there is clearly missing data.

We now need to discover specifically which features contain missing data. There are a number of ways of doing this ..

Student Task: In the following code block write some Python code that directly counts the number of missing values in each feature and prints the result. For this task, **do not** use the 'missingno' library. Rather, write the code to print the totals yourself.

In [ ]:

Now we are going to visualise missing data using the 'missingno' library.

Student Task: Import the missingno library and create a graphic display that allows you to visualise missing data in the 'credit_data' dataframe.

In [ ]:

Use a second missingno function to display a bar-chart of the number of data items in each feature of the credit_data data-set.

In [ ]:

1.2 Repair missing data

We will use two different strategies to 'repair' the missing data:

1. Removing any data records with a missing 'credit_amount', and 2
2. Imputing missing values for the 'age' feature - using the average age from the data-set

Student Task: In the code-block below, add Python code that removes any row of data that has a 'NaN' in the 'credit_amount' feature. **Hint:** There reference for the Pandas 'dropna' function is here: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html

In [ ]:

Check the above table ... at this point there should be 987 rows remaining in the data-set. The 'Age' feature should still contain 'NaN' values.

Student Task: In the code-block below, add Python code that replaces any missing ('NaN') values in the 'Age' feature with the average of all ages in the data-set.

In [ ]:

1.3 Change all of the category data into numbers

The file includes a number of features that are coded into categories. For example 'check_account_status' is coded as A11, A12, A13,A14 etc. We need to convert those categories into numbers ...

The 'Pandas' library has a function that will do that for us called 'get_dummies'

In [ ]:
```python
one_hot_check_acc_stat = pd.get_dummies(credit_data['check_account_status'], prefix='ch_ac_st')
print( one_hot_check_acc_stat[0:5])
```

We don't need the original column .. so we can delete this ...

In [ ]:
```python
credit_data.drop('check_account_status', axis=1, inplace = True)
```

Then substitue in the new numerical columns we just created

In [ ]:
```python
credit_data = one_hot_check_acc_stat.join(credit_data)
```

Take a look at that ...

In [ ]:
```python
credit_data[0:5]
```

We can now do the same for the 'credit_history' column

In [ ]:
```python
one_hot_credit_hist = pd.get_dummies(credit_data['credit_history'], prefix='cr_hist')
credit_data.drop('credit_history', axis=1, inplace=True)
credit_data = one_hot_credit_hist.join(credit_data)
credit_data[0:5]
```

And then do the same with all of the other category columns ...

purpose ...

In [ ]:
```python
column_name = 'purpose'
one_hot = pd.get_dummies(credit_data[column_name], prefix='purp')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

savings_acccount ..

In [ ]:
```python
column_name = 'savings_account'
one_hot = pd.get_dummies(credit_data[column_name], prefix='sav_acc')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

employment_duration

In [ ]:
```python
column_name = 'employment_duration'
one_hot = pd.get_dummies(credit_data[column_name], prefix='emp_dur')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

gender_marriage

In [ ]:
```python
column_name = 'gender_marriage'
one_hot = pd.get_dummies(credit_data[column_name], prefix='gend_mar')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

other_debtors

In [ ]:
```python
column_name = 'other_debtors'
one_hot = pd.get_dummies(credit_data[column_name], prefix='other_deb')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

property

In [ ]:
```python
column_name = 'property'
one_hot = pd.get_dummies(credit_data[column_name], prefix='prop')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

other_plans

In [ ]:
```python
column_name = 'other_plans'
one_hot = pd.get_dummies(credit_data[column_name], prefix='plans')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

housing

In [ ]:
```python
column_name = 'housing'
one_hot = pd.get_dummies(credit_data[column_name], prefix='house')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

job

In [ ]:
```python
column_name = 'job'
one_hot = pd.get_dummies(credit_data[column_name], prefix='job')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

telephone

In [ ]:
```python
column_name = 'telephone'
one_hot = pd.get_dummies(credit_data[column_name], prefix='tel')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

foreign_worker

In [ ]:
```python
column_name = 'foreign_worker'
one_hot = pd.get_dummies(credit_data[column_name], prefix='fw')
credit_data.drop(column_name, axis=1, inplace=True)
credit_data=one_hot.join(credit_data)
```

Finally, show the first three rows of the set with all the new numerical columns

In [ ]:
```python
credit_data[0:12]
```

Step 2 Select the Algorithm

In this case .. we know that the algorithm we will be using is classification using Logistic Regression .. so we can go ahead and build the model..

Step 3 Build the Model

In this section we will do a quick demonstration example of model building using all of the data. In the next section we will split the data into two parts to enable testing of the model.

First we import the library module that provides logistic regression. This is the 'sklearn' library (See https://scikit-learn.org/stable/index.html }

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
```

The data is split into two parts. 'X' represents the known 'inputs' to the model. 'Y' represents the 'label' or known (expected) output from the model.

In [ ]:
```python
X = credit_data.loc[:,'fw_A201':'dependents']
Y = credit_data['default']
```

We create an instance of the specific modelling algorythm we need (this is a bit like selecting the particular 'machine' or 'tool' that we will use to build the model).

In [ ]:
```python
logModel = LogisticRegression(solver='liblinear')
```

Then we put the data into the 'machine' to build the model:

In [ ]:
```python
logModel = logModel.fit(X,Y)
```

Note The following is not really legitimate since it scores (provides one quality measure of) the model - but it does so using exactly the same data that the original model was built from. This would be a bit like setting a student an exam consisting of questions they had already practiced in class.

However, for now it provides an indication that we have, at least, really built a model!

In [ ]:
```python
print (logModel.score(X, Y))
```

Step 4 Check Model Quality

In practice, before we build our model, we should split the data into two parts. One part will be used for building the model, the other part will be used to test how good it is at making predictions

First we import a new library function that allows us to randomly split the data..

In [ ]:
```python
from sklearn.model_selection import train_test_split
```

We use this function to split the data into two parts in the ratio 80%:20% as we don't need so much data for testing

In [ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

As above .. we build a model, but this time based only on the training data

In [ ]:
```python
logModel = logModel.fit(X_train,y_train)
```

Then generate a set of predictions based on the test data ...

In [ ]:
```python
predictions = logModel.predict(X_test)
```

In [ ]:
```python
predictions
```

We can get the 'score' for the model .. what fraction of the time did we get the right answer?

Note: Because the data we use is split randomly every time the model is built .. you may get different results to the ones shown below.

In [ ]:
```python
print(logModel.score(X_test,y_test))
```

A more useful tool for measuring quality is the 'confusion matrix'

In [ ]:
```python
from sklearn.metrics import confusion_matrix
print (confusion_matrix(y_test, predictions))
```

The score is calculated by adding the diagonal (add top-left to bottom-right), then divide by the total number of cases (sum of all items)

Step 5: Build the model into an application

In the case of Logistic Regression the 'model' is just a list of coefficients for an equation. We can look at the values of these coeficients ..

In [ ]:
```python
print( logModel.intercept_[0])
print(*logModel.coef_[0], sep='\n')
```

As a quick check we can find the data for one person so that we can test our model in another application

In [ ]:
```python
print(X_test[0:1].to_csv(index=False, sep='\n', header=False) )
```

Run the model for that one person so that we know the expected result when testing our application

In [ ]:
```python
print(logModel.predict_proba(X_test[0:1]))
```

Now let's save this model into a file:

In [ ]:
```python
import pickle
model_filename = 'germ_cred_model.pkl'
# Open the file to save as pkl file
the_file = open(model_filename, 'wb')
pickle.dump(logModel, the_file)
# Close the pickle instances
the_file.close()
```


In [ ]: