Multiple Logistical Reg - Australian Rain

February 17, 2020

Data: For this project, I have used a Kaggle dataset (https://www.kaggle.com/jsphyg/weather-dataset-rattle-package), which tries to predict the possibility of rain tomorrow, in Australia. The dataset contains about 10 years worth of data, since for our problem statement, i.e. to find the probablity of rain tomorrow using multiple logistical regression, will benefit from as much data as possible. This is especially true since we may need to drop some rows due to majority missing data. There are 23 features available, and the target variable is 'RainTomorrow'. The dummy variable is 'RainToday', which is binary data and the rest of the features are numerical in nature.

[1]:
```
# imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as sm
import scipy.stats as stats
```

[2]:
```
from sklearn.model_selection import train_test_split # required to split the␣
↪data
from sklearn.feature_selection import SelectKBest # to select best feature
from sklearn.feature_selection import chi2 # for feature selection
from sklearn.metrics import accuracy_score # to get the scores
from sklearn.linear_model import LogisticRegression # for logistic regression
from sklearn.metrics import r2_score # to calculate r2, adj r2
```

[3]:
```
# Load the csv to a dataframe & display
df = pd.read_csv(r'C:\Users\Dell\Desktop\Assignments\Adv␣
↪Stats\weather-dataset-rattle-package\weatherAUS.csv')
print(df.head())
```

```
         Date Location  MinTemp  MaxTemp  Rainfall  Evaporation  Sunshine  \
0  2008-12-01   Albury     13.4     22.9       0.6          NaN       NaN
1  2008-12-02   Albury      7.4     25.1       0.0          NaN       NaN
2  2008-12-03   Albury     12.9     25.7       0.0          NaN       NaN
3  2008-12-04   Albury      9.2     28.0       0.0          NaN       NaN
4  2008-12-05   Albury     17.5     32.3       1.0          NaN       NaN

  WindGustDir  WindGustSpeed WindDir9am  … Humidity3pm  Pressure9am  \
0           W           44.0          W  …        22.0       1007.7
```

```
1          WNW           44.0         NNW  …         25.0        1010.6
2          WSW           46.0           W  …         30.0        1007.6
3           NE           24.0          SE  …         16.0        1017.6
4            W           41.0         ENE  …         33.0        1010.8

   Pressure3pm  Cloud9am  Cloud3pm  Temp9am  Temp3pm  RainToday  RISK_MM  \
0       1007.1       8.0       NaN     16.9     21.8         No      0.0
1       1007.8       NaN       NaN     17.2     24.3         No      0.0
2       1008.7       NaN       2.0     21.0     23.2         No      0.0
3       1012.8       NaN       NaN     18.1     26.5         No      1.0
4       1006.0       7.0       8.0     17.8     29.7         No      0.2

   RainTomorrow
0            No
1            No
2            No
3            No
4            No

[5 rows x 24 columns]
```

[4]:

```
# Drop columns
df = df.drop(columns=['RISK_MM'],axis=1) # Instructed to drop this col in the␣
↪Kaggle instructions
df = df.drop(columns=['Date', 'Location', 'WindGustDir', 'WindDir9am',␣
↪'WindDir3pm'],axis=1) # lot of missing data
```

[5]:

```
print(df.isnull().sum())
```

```
MinTemp            637
MaxTemp            322
Rainfall          1406
Evaporation      60843
Sunshine         67816
WindGustSpeed     9270
WindSpeed9am      1348
WindSpeed3pm      2630
Humidity9am       1774
Humidity3pm       3610
Pressure9am      14014
Pressure3pm      13981
Cloud9am         53657
Cloud3pm         57094
Temp9am            904
Temp3pm           2726
RainToday         1406
RainTomorrow         0
dtype: int64
```

[6]:
```
#lot of missing data from the above result
df = df.drop(columns=['Sunshine', 'Evaporation', 'Cloud3pm', 'Cloud9am'],axis=1)
```

[8]:
```
print(df.head())
```

```
   MinTemp  MaxTemp  Rainfall  WindGustSpeed  WindSpeed9am  WindSpeed3pm  \
0     13.4     22.9       0.6           44.0          20.0          24.0
1      7.4     25.1       0.0           44.0           4.0          22.0
2     12.9     25.7       0.0           46.0          19.0          26.0
3      9.2     28.0       0.0           24.0          11.0           9.0
4     17.5     32.3       1.0           41.0           7.0          20.0

   Humidity9am  Humidity3pm  Pressure9am  Pressure3pm  Temp9am  Temp3pm  \
0         71.0         22.0       1007.7       1007.1     16.9     21.8
1         44.0         25.0       1010.6       1007.8     17.2     24.3
2         38.0         30.0       1007.6       1008.7     21.0     23.2
3         45.0         16.0       1017.6       1012.8     18.1     26.5
4         82.0         33.0       1010.8       1006.0     17.8     29.7

  RainToday RainTomorrow
0        No           No
1        No           No
2        No           No
3        No           No
4        No           No
```

[9]:
```
# Turning target variable and feature into bool
df['RainToday'].replace({'No': 0, 'Yes': 1},inplace = True)
df['RainTomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)
print(df.head())
```

```
   MinTemp  MaxTemp  Rainfall  WindGustSpeed  WindSpeed9am  WindSpeed3pm  \
0     13.4     22.9       0.6           44.0          20.0          24.0
1      7.4     25.1       0.0           44.0           4.0          22.0
2     12.9     25.7       0.0           46.0          19.0          26.0
3      9.2     28.0       0.0           24.0          11.0           9.0
4     17.5     32.3       1.0           41.0           7.0          20.0

   Humidity9am  Humidity3pm  Pressure9am  Pressure3pm  Temp9am  Temp3pm  \
0         71.0         22.0       1007.7       1007.1     16.9     21.8
1         44.0         25.0       1010.6       1007.8     17.2     24.3
2         38.0         30.0       1007.6       1008.7     21.0     23.2
3         45.0         16.0       1017.6       1012.8     18.1     26.5
4         82.0         33.0       1010.8       1006.0     17.8     29.7

   RainToday  RainTomorrow
0        0.0             0
1        0.0             0
```

```
2        0.0              0
3        0.0              0
4        0.0              0
```

[10]:

```python
print(df.describe())
```

```
             MinTemp        MaxTemp       Rainfall  WindGustSpeed  \
count  141556.000000  141871.000000  140787.000000  132923.000000
mean       12.186400      23.226784       2.349974      39.984292
std         6.403283       7.117618       8.465173      13.588801
min        -8.500000      -4.800000       0.000000       6.000000
25%         7.600000      17.900000       0.000000      31.000000
50%        12.000000      22.600000       0.000000      39.000000
75%        16.800000      28.200000       0.800000      48.000000
max        33.900000      48.100000     371.000000     135.000000

        WindSpeed9am   WindSpeed3pm    Humidity9am    Humidity3pm  \
count  140845.000000  139563.000000  140419.000000  138583.000000
mean       14.001988      18.637576      68.843810      51.482606
std         8.893337       8.803345      19.051293      20.797772
min         0.000000       0.000000       0.000000       0.000000
25%         7.000000      13.000000      57.000000      37.000000
50%        13.000000      19.000000      70.000000      52.000000
75%        19.000000      24.000000      83.000000      66.000000
max       130.000000      87.000000     100.000000     100.000000

         Pressure9am    Pressure3pm        Temp9am        Temp3pm  \
count  128179.000000  128212.000000  141289.000000  139467.000000
mean     1017.653758    1015.258204      16.987509      21.687235
std         7.105476       7.036677       6.492838       6.937594
min       980.500000     977.100000      -7.200000      -5.400000
25%      1012.900000    1010.400000      12.300000      16.600000
50%      1017.600000    1015.200000      16.700000      21.100000
75%      1022.400000    1020.000000      21.600000      26.400000
max      1041.000000    1039.600000      40.200000      46.700000

           RainToday   RainTomorrow
count  140787.000000  142193.000000
mean        0.223423       0.224181
std         0.416541       0.417043
min         0.000000       0.000000
25%         0.000000       0.000000
50%         0.000000       0.000000
75%         0.000000       0.000000
max         1.000000       1.000000
```

To clean the data, I could either drop the rows with no data, or find the mean of the feature and replace it with NaN. I am opting to do the latter since I have a large dataset with roughly 140,000 rows. The assumption is that the mean will be a fair representation of the missing data and will not adversely skew the results.

[11]:
```
# Replace missing data with the mean.
df.MinTemp.fillna(df.MinTemp.mean(),inplace=True)
df.MaxTemp.fillna(df.MaxTemp.mean(),inplace=True)
df.Rainfall.fillna(df.Rainfall.mean(),inplace=True)
df.WindSpeed9am.fillna(df.WindSpeed9am.mean(),inplace=True)
df.WindSpeed3pm.fillna(df.WindSpeed3pm.mean(),inplace=True)
df.Humidity9am.fillna(df.Humidity9am.mean(),inplace=True)
df.Temp9am.fillna(df.Temp9am.mean(),inplace=True)
df.Temp3pm.fillna(df.Temp3pm.mean(),inplace=True)
df.RainToday.fillna(df.RainToday.mean(),inplace=True)
df.Humidity3pm.fillna(df.Humidity3pm.mean(),inplace=True)
df.WindGustSpeed.fillna(df.WindGustSpeed.mean(),inplace=True)
df.Pressure9am.fillna(df.Pressure9am.mean(),inplace=True)
df.Pressure3pm.fillna(df.Pressure3pm.mean(),inplace=True)
```

[12]:
```
#replace negative values with 0
df.clip(lower=0)
df[df < 0] = 0
```

Selectkbest from the sklearn package, extracts the best features from the dataset. I am choosing the best 5.

[13]:
```
# From selectkbest find the 5 best score and plot it
X = df.loc[:,df.columns!='RainTomorrow']
y = df[['RainTomorrow']]
selector = SelectKBest(chi2, k=5)
selector.fit(X, y)
print(X.columns[selector.get_support(indices=True)]) #top 5 columns
```

```
Index(['Rainfall', 'WindGustSpeed', 'Humidity9am', 'Humidity3pm', 'Temp3pm'],
dtype='object')
```

I will use these features for the training and testing process

[14]:
```
# Select the selected columns for testing and training
X = df.loc[:, ['Rainfall', 'Humidity3pm', 'Humidity9am', 'WindGustSpeed',␣
↪'Temp3pm']].shift(-1).iloc[:-1].values
y = df.iloc[:-1, -1:].values.astype('int')
```

[15]:
```
# Split the data to appropriate testing sample size, which is a third
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33)
```

I will first fit the model using the logit funnction of the statsmodel package

[16]:
```
X_train_df = pd.DataFrame(X_train, columns= ['Rainfall', 'Humidity3pm',␣
↪'Humidity9am', 'WindGustSpeed', 'Temp3pm'])
```

```
y_train_df = pd.DataFrame(y_train, columns=['RainTomorrow'])
model = sm.logit(formula = 'y_train_df ~ X_train_df', data = X_train_df)
result = model.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.057491
         Iterations 12
                           Logit Regression Results
==============================================================================
Dep. Variable:             y_train_df   No. Observations:                95268
Model:                          Logit   Df Residuals:                    95262
Method:                           MLE   Df Model:                            5
Date:                Wed, 12 Feb 2020   Pseudo R-squ.:                  0.8921
Time:                        22:45:25   Log-Likelihood:                -5477.0
converged:                       True   LL-Null:                       -50750.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Intercept        -7.3153      0.247    -29.647      0.000      -7.799
-6.832
X_train_df[0]     4.1394      0.045     92.960      0.000       4.052
4.227
X_train_df[1]     0.0083      0.002      4.477      0.000       0.005
0.012
X_train_df[2]     0.0189      0.002      8.741      0.000       0.015
0.023
X_train_df[3]     0.0119      0.002      5.785      0.000       0.008
0.016
X_train_df[4]    -0.0323      0.005     -6.671      0.000      -0.042
-0.023
================================================================================
=

Possibly complete quasi-separation: A fraction 0.14 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

We see that all the variables are statistically significant at 5%, with all p values being 0. But we see that only the Rainfall variable has a major impact of the prediction, with a coefficient of 4.14. The rest have values at or below 0.02. Standard error is very small as well We also see that the R-square is 0.89, which means there is a high possiblity of a positive correlation between the features in this model and the target variable. Economic and statistical understanding of the variables 1) Rainfall - if the area typically receives high rainfall today, there is a chance it will rain tomorrow.

Indicating positive relationship 2 & 3) Humidity at 3pm, Humidity at 9am - If there is humidity, it means it might rain soon. Hence again, a positive correlation. 4) WindgustSpeed - High winds, often bring storms. Positive correlation. 5) Temperature at 3pm - Low temperature is seen during rain, hence negative correlation All of this behaviour is seen in the coeffcient result as well. The confidence intervals also do not cross zero anywhere and hence are valid. So we can interpret this model follows- Rainfall is the only variable that has a strong positive correlation on whether it will rain tomorrow. The rest of the variables, except Temp are also positively correlated, but their effect is not very strong. Temperature is also weakly negatively correlated. However, overall the model indicates a positive correlation due to the high 0.8 R2 value

Now, we attempt to test the model with the training data and we see a good accuracy

[18]:
```
# Test the accuracy
model_lr = LogisticRegression(random_state=0, solver='lbfgs')
model_lr.fit(X_train,y_train)
prediction_lr = model_lr.predict(X_test)
print(prediction_lr)
score = accuracy_score(y_test,prediction_lr)
print('Accuracy - Logistic Regression:',score)
```

```
C:\Users\Dell\Anaconda3\lib\site-packages\sklearn\utils\validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

```
[0 0 0 … 0 0 0]
Accuracy - Logistic Regression: 0.9814593811269287
```

Calculating adj_r2 and f-stat

[20]:
```
# Calculating R2 for adjusted R2
prediction_lr = model_lr.predict(X_train)
R2 = r2_score(y_train,prediction_lr)
print('Sklearn R2_Score', R2)
n = y_test.size
p = 5
Adj_R2 = 1-(1-R2)*(n-1)/(n-p-1)
print('Calculated Adj R2', Adj_R2)

# Calculating f statistic
F = np.var(X_train) / np.var(y_train)
df1 = len(X_train) - 1
df2 = len(y_train) - 1
alpha = 0.05
p_value = stats.f.sf(F, df1, df2)  # survival function or 1-cdf
print('p-value of the F statistic',p_value)
if(p_value<alpha):
    print('We reject the null hypothesis')
```

```
else:
    print('We fail to reject the null hypothesis')
```

```
Sklearn R2_Score 0.894653924226544
Calculated Adj R2 0.8946426976103441
p-value of the F statistic 0.0
We reject the null hypothesis
```

We can see that the adjusted r2 value remains close to the r2 value and the p-value of the f stat is also 0. Hence we can say that the model has good fit. Although online web pages do not use adjusted r2 and f stat to gauge the correctness of logistical regressions and use ROC and confusion matrix instead, I have manually calculated this for the project.