Project 2

ET1550 Introduction to Machine Learning and Artificial Intelligence

Kevin Rasmusson

Lund, 07/14/2022

Question 1

Code

```
# dataset.corr() # for entire corr() matrix
print(dataset.corr()["target"].to_string()) # smoothness_error 0.1
```

Line 1 was only to see the entire correlation matrix. However, the question asked specifically for the correlations with regards to target, why the second line was written. If I only would've written dataset.corr()["target"] pandas output would've replaced a few lines with '...', why I decided to use the to_string() method instead.

Output

mean_radius	-0.7
mean_texture	-0.4
mean_perimeter	-0.7
mean_area	-0.7
mean_smoothness	-0.4
mean_compactness	-0.6
mean_concavity	-0.7
mean_concave_points	-0.8
mean_symmetry	-0.3
mean_fractal_dimension	0.0
radius_error	-0.6
texture_error	0.0
perimeter_error	-0.6
area_error	-0.5
smoothness_error	**0.1**
compactness_error	-0.3
concavity_error	-0.3
concave_points_error	-0.4
symmetry_error	0.0
fractal_dimension_error	-0.1
worst_radius	-0.8
worst_texture	-0.5
worst_perimeter	-0.8
worst_area	-0.7
worst_smoothness	-0.4
worst_compactness	-0.6
worst_concavity	-0.7
worst_concave_points	-0.8
worst_symmetry	-0.4
worst_fractal_dimension	-0.3
target	1.0

Comments
We were looking for the feature most correlated with our target, i.e., the feature with a correlation as close to 1 as possible (seeing that correlation is a measure from -1 to 1, where -1 implies inversely correlated). This came out to be the feature smoothness_error with a correlation of positive 0.1.

Question 2

Code

```
y_dataset = dataset[["target"]]
X_dataset = dataset.iloc[:, :-1]
```

Comments
Getting the 'y_dataset', we face the problem of selecting only a specific column of data. The hint was to use pop(), however, this implied having to reload the values in jupyter if something went wrong. Thus, I decided to use the built in selector on the dataset ([]). In there you can pass many different expressions, e.g. filtering using lambda expression, or as in our case, an array of string values whose features you want to extract.

With regards to the 'X_dataset', if we would've used the .pop() function earlier, I suppose this would be equal to the "remaining" dataset. However, since I went a different route, I had to use iloc, a selector that works kind of like the selector when handling a numpy array (that is my understanding, though). Using expressions from the YouTube video provided in the canvas module "Programming readings", we want to make a '2D selection', why we need a comma separating how we handle our rows, and then columns. Firstly, the ':' on the left hand side, means we select everything inside a column. On the right hand side, we specify which columns, and by writing 0:-1, we select everything from index 0 to index -2 (index -1 is excluded), meaning we get all columns except the last one, which is the targets.

Output
Dataset separated.

Question 3

Code

```
X_dataset_mean = X_dataset.mean()
X_dataset_std = X_dataset.std()
X_dataset_norm = (X_dataset - X_dataset_mean)/X_dataset_std
```

Comments
Not much to add since this is the exact same operations as performed in the previous project.

Output
Dataset normalized.

Question 4

Code

```
X_train_norm, X_test_norm, y_train, y_test = train_test_split(X_dataset,
y_dataset, train_size=0.9, test_size=0.1, random_state=110)
```

Output
Dataset split.

Question 5

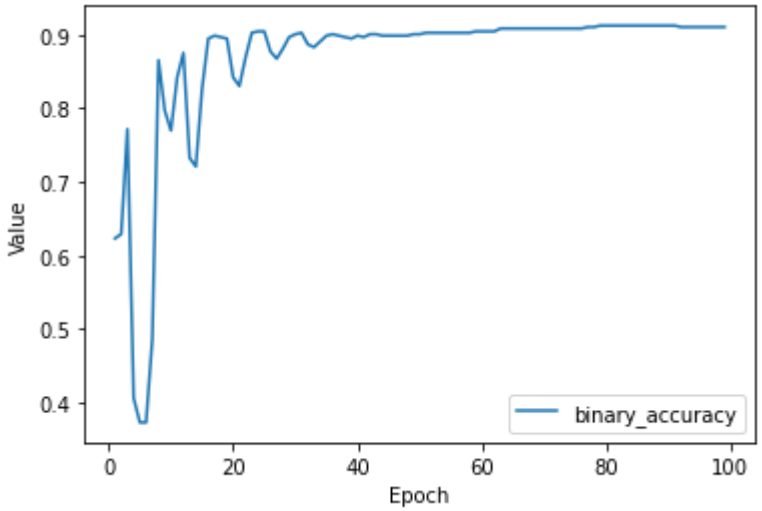
Code

```
model.add(tf.keras.layers.Dense(units=1 , activation='sigmoid'))
```

Output
Defined the create_model and train_model functions.

Question 6

Result of logistic regression



2/2 [==============================] - 0s 2ms/step - loss: 0.6483 - binary_accuracy: 0.8596
[0.6483106017112732, 0.859649121761322]

Question 7

Code

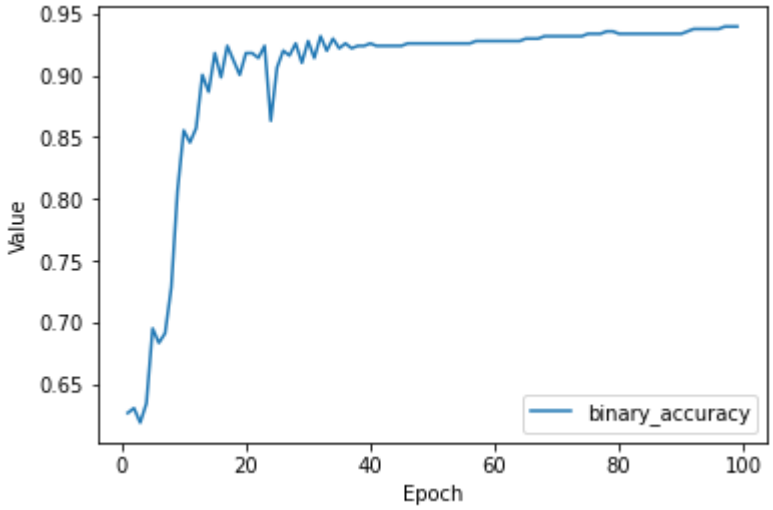
```
# Define the second hidden layer with 8 nodes.
model.add(tf.keras.layers.Dense(
      units=8,
      activation='relu',
      kernel_regularizer=tf.keras.regularizers.l2(l=0.001),
      name='Hidden2'))

# Define the third hidden layer with 6 nodes.
model.add(tf.keras.layers.Dense(
      units=6,
      activation='relu',
      kernel_regularizer=tf.keras.regularizers.l2(l=0.001),
      name='Hidden3'))
```

Output
Defined the create_model and train_model functions.

Question 8

NN model performance



2/2 [==============================] - 0s 2ms/step - loss: 0.2154 - binary_accuracy: 0.8246
[0.21540695428848267, 0.8245614171028137]

Question 9

Performance differences

Comparing performance, question six and eight gave the following results:

Architecture	Loss	Binary Accuracy
Logistic Regression	0.64831	0.86965
Neural Network	0.21541	0.82456

Table I. Performance of two different models with regards to loss and binary accuracy.

This shows that whilst the logistic regression had a bigger loss, it had a slightly higher binary accuracy. The neural network on the other hand, had a small loss, at the cost of a slightly lower binary accuracy.

Overfitting

Regarding overfitting, both models showed a significant difference in both loss and binary accuracy between the last epoch's performance and the test set's performance. To concretize, the logistic regression's last epoch loss and binary accuracy were 0.4857 and 0.9102 respectively. Comparing this to the test set, the logistic regression showed a significant increase in loss and a noticeable decrease in binary accuracy, both indicative of an overfitting problem.

The neural network's last epoch loss and binary accuracy were 0.1829 and 0.9395 respectively, compared to 0.2153 and 0.8245 with its test set. These results imply a high variance between the dev set and the training set, which could be explained by and overfitting problem. To counteract this, three different methods can be used: (1) obtain and train on more data, (2) regularization and/or (3) different NN architecture. However, the loss stayed fairly constant, indicating that the overfitting is not as extreme as the case with the logistic regression.