PoisonousMushrooms

January 8, 2018

1 Importing the libraries

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import os
        from keras.models import Sequential
        from keras.layers import Dense
```

```
Using TensorFlow backend.
```

```
In [2]: dataset = pd.read_csv('data/mushrooms.csv')
```

```
In [3]: X = dataset.iloc[:, 1:23].values
        y = dataset.iloc[:, 0].values
```

2 Encoding the variables

```
In [4]: from sklearn.preprocessing import LabelEncoder, OneHotEncoder

        #class: 2
        labelencoder_y = LabelEncoder()
        y[:] = labelencoder_y.fit_transform(y[:])
        print(labelencoder_y.inverse_transform([0,1]))
        print("--> 0,1\n")
```

```
['e' 'p']
--> 0,1
```

```
In [5]: #cap-shape: 6
        labelencoder0 = LabelEncoder()
        X[:,0] = labelencoder0.fit_transform(X[:,0])
        print(labelencoder0.inverse_transform([0,1,2,3,4,5]))
        print("--> 0,1,2,3,4,5\n")
```

```
['b' 'c' 'f' 'k' 's' 'x']
--> 0,1,2,3,4,5
```

In [6]:
```python
#cap-surface: 4
labelencoder1 = LabelEncoder()
X[:,1] = labelencoder1.fit_transform(X[:,1])
print(labelencoder1.inverse_transform([0,1,2,3]))
print("--> 0,1,2,3\n")
```

```
['f' 'g' 's' 'y']
--> 0,1,2,3
```

In [7]:
```python
#cap-color: 10
labelencoder2 = LabelEncoder()
X[:,2] = labelencoder2.fit_transform(X[:,2])
print(labelencoder2.inverse_transform([0,1,2,3,4,5,6,7,8,9]))
print("--> 0,1,2,3,4,5,6,7,8,9\n")
```

```
['b' 'c' 'e' 'g' 'n' 'p' 'r' 'u' 'w' 'y']
--> 0,1,2,3,4,5,6,7,8,9
```

In [8]:
```python
#bruises:2
labelencoder3 = LabelEncoder()
X[:,3] = labelencoder3.fit_transform(X[:,3])
print(labelencoder3.inverse_transform([0,1]))
print("--> 0,1\n")
```

```
['f' 't']
--> 0,1
```

In [9]:
```python
#odor:9
labelencoder4 = LabelEncoder()
X[:,4] = labelencoder4.fit_transform(X[:,4])
print(labelencoder4.inverse_transform([0,1,2,3,4,5,6,7,8]))
print("--> 0,1,2,3,4,5,6,7,8\n")
```

```
['a' 'c' 'f' 'l' 'm' 'n' 'p' 's' 'y']
--> 0,1,2,3,4,5,6,7,8
```

In [10]:
```python
#gill-attachment:2
labelencoder5 = LabelEncoder()
X[:,5] = labelencoder5.fit_transform(X[:,5])
print(labelencoder5.inverse_transform([0,1]))
print("--> 0,1\n")
```

```
['a' 'f']
--> 0,1
```

In [11]:
```python
#gill-spacing:2
labelencoder6 = LabelEncoder()
X[:,6] = labelencoder6.fit_transform(X[:,6])
print(labelencoder6.inverse_transform([0,1]))
print("--> 0,1\n")
```

```
['c' 'w']
--> 0,1
```

In [12]:
```python
#gill-size:2
labelencoder7 = LabelEncoder()
X[:,7] = labelencoder7.fit_transform(X[:,7])
print(labelencoder7.inverse_transform([0,1]))
print("--> 0,1\n")
```

```
['b' 'n']
--> 0,1
```

In [13]:
```python
#gill-color:12
labelencoder8 = LabelEncoder()
X[:,8] = labelencoder8.fit_transform(X[:,8])
print(labelencoder8.inverse_transform([0,1,2,3,4,5,6,7,8,9,10,11]))
print("--> 0,1,2,3,4,5,6,7,8,9,10,11\n")
```

```
['b' 'e' 'g' 'h' 'k' 'n' 'o' 'p' 'r' 'u' 'w' 'y']
--> 0,1,2,3,4,5,6,7,8,9,10,11
```

In [14]:
```python
#sta;l-shape:2
labelencoder9 = LabelEncoder()
X[:,9] = labelencoder9.fit_transform(X[:,9])
print(labelencoder9.inverse_transform([0,1]))
print("--> 0,1\n")
```

```
['e' 't']
--> 0,1
```

```
In [15]: #stalk-root:5
         labelencoder10 = LabelEncoder()
         X[:,10] = labelencoder10.fit_transform(X[:,10])
         print(labelencoder10.inverse_transform([0,1,2,3,4]))
         print("--> 0,1,2,3,4\n")
```

```
['?' 'b' 'c' 'e' 'r']
--> 0,1,2,3,4
```

```
In [16]: #stalk-surface-above-ring: 4
         labelencoder11 = LabelEncoder()
         X[:,11] = labelencoder11.fit_transform(X[:,11])
         print(labelencoder11.inverse_transform([0,1,2,3]))
         print("--> 0,1,2,3\n")
```

```
['f' 'k' 's' 'y']
--> 0,1,2,3
```

```
In [17]: #stalk-surface-below-ring: 3
         labelencoder12 = LabelEncoder()
         X[:,12] = labelencoder12.fit_transform(X[:,12])
         print(labelencoder12.inverse_transform([0,1,2]))
         print("--> 0,1,2\n")
```

```
['f' 'k' 's']
--> 0,1,2
```

```
In [18]: #stalk-color-above-ring: 9
         labelencoder13 = LabelEncoder()
         X[:,13] = labelencoder13.fit_transform(X[:,13])
         print(labelencoder13.inverse_transform([0,1,2,3,4,5,6,7,8]))
         print("--> 0,1,2,3,4,5,6,7,8\n")
```

```
['b' 'c' 'e' 'g' 'n' 'o' 'p' 'w' 'y']
--> 0,1,2,3,4,5,6,7,8
```

In [19]:
```
#stalk-color-below-ring: 9
labelencoder14 = LabelEncoder()
X[:,14] = labelencoder14.fit_transform(X[:,14])
print(labelencoder14.inverse_transform([0,1,2,3,4,5,6,7,8]))
print("--> 0,1,2,3,4,5,6,7,8\n")
```

```
['b' 'c' 'e' 'g' 'n' 'o' 'p' 'w' 'y']
--> 0,1,2,3,4,5,6,7,8
```

In [20]:
```
#veil-type: 1
labelencoder15 = LabelEncoder()
X[:,15] = labelencoder15.fit_transform(X[:,15])
print(labelencoder15.inverse_transform([0]))
print("--> 0\n")
```

```
['p']
--> 0
```

In [21]:
```
#veil-color: 4
labelencoder16 = LabelEncoder()
X[:,16] = labelencoder16.fit_transform(X[:,16])
print(labelencoder16.inverse_transform([0,1,2,3]))
print("--> 0,1,2,3\n")
```

```
['n' 'o' 'w' 'y']
--> 0,1,2,3
```

In [22]:
```
#ring-number: 3
labelencoder17 = LabelEncoder()
X[:,17] = labelencoder17.fit_transform(X[:,17])
print(labelencoder17.inverse_transform([0,1,2]))
print("--> 0,1,2\n")
```

```
['n' 'o' 't']
--> 0,1,2
```

In [23]:
```
#ring-type: 5
labelencoder18 = LabelEncoder()
X[:,18] = labelencoder18.fit_transform(X[:,18])
print(labelencoder18.inverse_transform([0,1,2,3,4]))
print("--> 0,1,2,3,4\n")
```

```
['e' 'f' 'l' 'n' 'p']
--> 0,1,2,3,4
```

```
In [24]: #spore-print-color: 9
         labelencoder19 = LabelEncoder()
         X[:,19] = labelencoder19.fit_transform(X[:,19])
         print(labelencoder19.inverse_transform([0,1,2,3,4,5,6,7,8]))
         print("--> 0,1,2,3,4,5,6,7,8\n")
```

```
['b' 'h' 'k' 'n' 'o' 'r' 'u' 'w' 'y']
--> 0,1,2,3,4,5,6,7,8
```

```
In [25]: #population: 6
         labelencoder20 = LabelEncoder()
         X[:,20] = labelencoder20.fit_transform(X[:,20])
         print(labelencoder20.inverse_transform([0,1,2,3,4,5]))
         print("--> 0,1,2,3,4,5\n")
```

```
['a' 'c' 'n' 's' 'v' 'y']
--> 0,1,2,3,4,5
```

```
In [26]: #habitat: 6
         labelencoder21 = LabelEncoder()
         X[:,21] = labelencoder21.fit_transform(X[:,21])
         print(labelencoder21.inverse_transform([0,1,2,3,4,5,6]))
         print("--> 0,1,2,3,4,5\n")
```

```
['d' 'g' 'l' 'm' 'p' 'u' 'w']
--> 0,1,2,3,4,5
```

3 Splitting the dataset into the Training set and Test set

```
In [27]: from sklearn.cross_validation import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(
             X, y, test_size = 0.25, random_state = 0)
```

```
C:\Users\Ripti\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: T
  "This module will be removed in 0.20.", DeprecationWarning)
```

4 Create model

- ReLU tends to converge better than sigmoid activation functions for when calculating gradients the sigmoid function is saturated and tends to take longer.
- Sigmoid function is good for classification at the final layer.

```
In [28]: model = Sequential()
         model.add(Dense(12, input_dim=22, activation='relu'))
         model.add(Dense(22, activation='relu'))
         model.add(Dense(1, activation='sigmoid'))
```

5 Compile model

```
In [29]: model.compile(loss='binary_crossentropy',
                       optimizer='adam', metrics=['accuracy'])
```

6 Fit the model

```
In [30]: model.fit(X_train, y_train, epochs=20, batch_size=10)
```

```
Epoch 1/20
6093/6093 [==============================] - 3s - loss: 0.3348 - acc: 0.8628
Epoch 2/20
6093/6093 [==============================] - 2s - loss: 0.1870 - acc: 0.9350
Epoch 3/20
6093/6093 [==============================] - 2s - loss: 0.1195 - acc: 0.9565     - ETA: 0s - los
Epoch 4/20
6093/6093 [==============================] - 2s - loss: 0.0781 - acc: 0.9737
Epoch 5/20
6093/6093 [==============================] - 2s - loss: 0.0504 - acc: 0.9846
Epoch 6/20
6093/6093 [==============================] - 2s - loss: 0.0319 - acc: 0.9918
Epoch 7/20
6093/6093 [==============================] - 2s - loss: 0.0210 - acc: 0.9962
Epoch 8/20
6093/6093 [==============================] - 2s - loss: 0.0154 - acc: 0.9962
Epoch 9/20
6093/6093 [==============================] - 2s - loss: 0.0084 - acc: 0.9985
Epoch 10/20
6093/6093 [==============================] - 2s - loss: 0.0056 - acc: 0.9995
Epoch 11/20
6093/6093 [==============================] - 2s - loss: 0.0046 - acc: 0.9995
Epoch 12/20
6093/6093 [==============================] - 2s - loss: 0.0046 - acc: 0.9993
Epoch 13/20
6093/6093 [==============================] - 2s - loss: 0.0040 - acc: 0.9993
Epoch 14/20
```

```
6093/6093 [==============================] - 2s - loss: 0.0027 - acc: 0.9997
Epoch 15/20
6093/6093 [==============================] - 2s - loss: 0.0027 - acc: 0.9992
Epoch 16/20
6093/6093 [==============================] - 2s - loss: 0.0011 - acc: 1.0000
Epoch 17/20
6093/6093 [==============================] - 2s - loss: 0.0026 - acc: 0.9997
Epoch 18/20
6093/6093 [==============================] - 2s - loss: 0.0024 - acc: 0.9995
Epoch 19/20
6093/6093 [==============================] - 2s - loss: 4.2112e-04 - acc: 1.0000
Epoch 20/20
6093/6093 [==============================] - 2s - loss: 3.1792e-04 - acc: 1.0000
```

```
Out[30]: <keras.callbacks.History at 0x26c20715550>
```

7 Verify the accuracy

```
In [31]: scores = model.evaluate(X_test, y_test)
         print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
2016/2031 [============================>.] - ETA: 0s
acc: 100.00%
```

8 Conclusion

- Here we dealt with 22 categorical features with over 8000 entries.
- After variable encoding and splitting the data into training and test sets, we ran the deep learning model with RELU and Sigmoid activation functions.
- After twenty epochs, the training data displayed that the model had achieved 100% accuracy, as well as the test data following the verification.
- Although this model may be overfitted, because it was able to predict the test data with high accuracy, we can be comfortable using this model in the future.
- It may be beneficial to test this model on a more diverse/ larger dataset, as high accuracies throw red flags for overfitting, and could be disastrous when we least expect it.