ANN Classification Final Project

Deliverable 2

Yiran Sun

4/20/2020

Objective

The objective of this project is to construct an ANN Classifier model to classify the edibility of the samples of 23 species of mushrooms in the Agaricus and Lepiota Family. The class of each sample mushroom edibility can be classified as either edible, or poisonous. The edible class indicates the mushrooms that are definitely without poisonous, whereas the mushrooms that are definitely poisonous or with unknown edibility are both classified as poisonous. The edibility will be predicted based on 22 features such as: cap-shape, cap-surface, cap-color, bruises, odor, gill-attachment, gill-spacing, gill-size, gill-color population, habitat, and so on. The optimal model will be chosen from 8 sets of experiments, with different number of layers, nodes, batch size, and epochs. Each set of experiment will be validated through K-fold cross validations, and the model that provides the highest accuracy as well as efficiency will be determined as the best model.

Data Source URL and Data Description

The full dataset can be found at:

https://www.kaggle.com/uciml/mushroom-classification

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class is combined with the poisonous one. (Kaggle)

There are 23 columns, indicating 23 features of all mushroom samples, and there are 8124 rows of samples. Each feature is recorded with a letter indicating its attribute, for example, as for the classes, “e” indicates edible class, “p” indicates poisonous class.

The full data attribute information is provided as below:

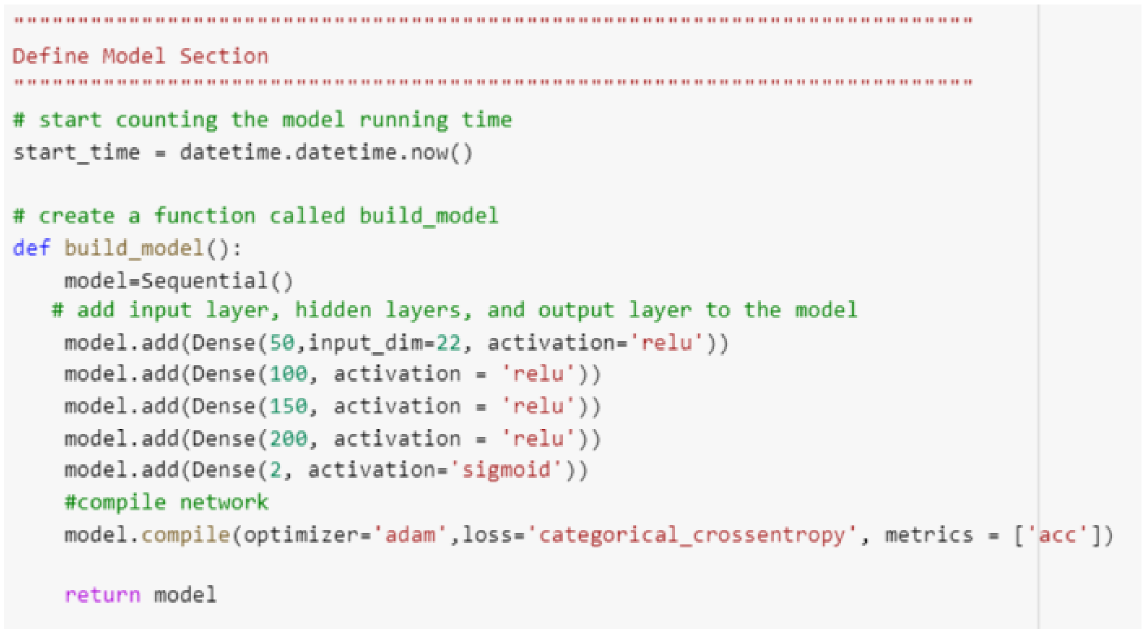
cap-shape	bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
cap-surface	fibrous=f,grooves=g,scaly=y,smooth=s
cap-color	brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
bruises	bruises=t,no=f
odor	almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
gill-attachment	attached=a,descending=d,free=f,notched=n
gill-spacing	close=c,crowded=w,distant=d
gill-size	broad=b,narrow=n
gill-color	black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
stalk-shape	enlarging=e,tapering=t
stalk-root	bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
stalk-surface-above-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-surface-below-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-color-above-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
stalk-color-below-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
veil-type	partial=p,universal=u
veil-color	brown=n,orange=o,white=w,yellow=y
ring-number	none=n,one=o,two=t
ring-type	cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
spore-print-color	black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
population	abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
habitat	grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d
Classes	edible=e, poisonous=p

Final ANN Model Code

The full code of the final ANN model is attached at the end of this deliverable.

Final Model and Training Algorithm

Optimal ANN Model in Code

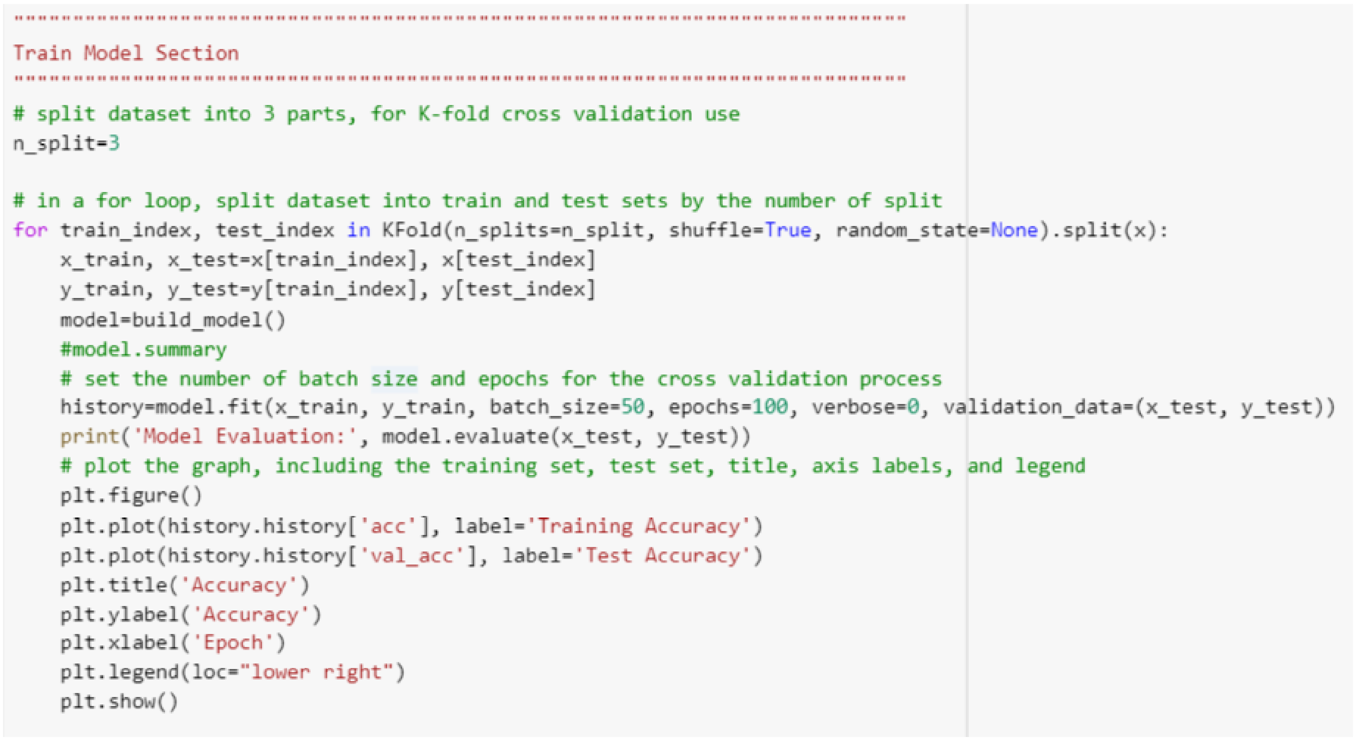


```python
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# start counting the model running time
start_time = datetime.datetime.now()

# create a function called build_model
def build_model():
    model=Sequential()
   # add input layer, hidden layers, and output layer to the model
    model.add(Dense(50,input_dim=22, activation='relu'))
    model.add(Dense(100, activation = 'relu'))
    model.add(Dense(150, activation = 'relu'))
    model.add(Dense(200, activation = 'relu'))
    model.add(Dense(2, activation='sigmoid'))
    #compile network
    model.compile(optimizer='adam',loss='categorical_crossentropy', metrics = ['acc'])

    return model
```

Optimal ANN Model in Words

The final model has total of 5 dense layers, including 1 input layer with 50 nodes, 3 hidden layers of 100, 150, and 200 nodes, and finally the output layer with 2 nodes. The nodes were chosen to avoid underfitting as well as overfitting, and the output layer is set to be 2 layers in accordance with the mushroom class of edibility of either edible or poisonous. The optimizer for this model I chose is Adam, which is usually a good option for classification model since it has less variability. And the activation function I used is ‘relu’ and ‘sigmoid’, since ‘sigmoid’ is usually used in binary classifications. I chose “categorical crossentropy” for loss, as it is generally used for single label categorization, which means only one category is applicable for each data point. And as for metrics, I used accuracy as it is usually applied to classification models. The batch size of this model is set to be 50, and the epochs is set to be 100.

Training Algorithm in Code


```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Train Model Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# split dataset into 3 parts, for K-fold cross validation use
n_split=3

# in a for loop, split dataset into train and test sets by the number of split
for train_index, test_index in KFold(n_splits=n_split, shuffle=True, random_state=None).split(x):
    x_train, x_test=x[train_index], x[test_index]
    y_train, y_test=y[train_index], y[test_index]
    model=build_model()
    #model.summary
    # set the number of batch size and epochs for the cross validation process
    history=model.fit(x_train, y_train, batch_size=50, epochs=100, verbose=0, validation_data=(x_test, y_test))
    print('Model Evaluation:', model.evaluate(x_test, y_test))
    # plot the graph, including the training set, test set, title, axis labels, and legend
    plt.figure()
    plt.plot(history.history['acc'], label='Training Accuracy')
    plt.plot(history.history['val_acc'], label='Test Accuracy')
    plt.title('Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc="lower right")
    plt.show()
```

Training Algorithm in Words

As for training algorithm, I applied K-fold cross validation method where K equals to 3. Through the training process, the K-fold for loop function splits my dataset into 3 parts equally, so the model will run 3 rounds each time, and during each of these 3 rounds, each 1/3 part gets to be the training set for once, and testing set for twice. The K is chosen to be 3 because it provides good enough number of experimental runs to show the reliability of each model. And I tried to keep K as small as possible, because the down sides of choosing a large K includes the model being split into very small training sets, so that each training set only contains small portion of the full dataset and cannot represent all of the characteristics of the dataset; moreover, a larger K leads to much longer training time and lower efficiency.

Experimental Plan

There are total of 4 parts of my experimental plan.

Part 1

In the first part, my plan is to learn the dataset, know the meaning of each column, and determine if all the attributes in the dataset are necessary to include in the neural network model, and then make a professional decision on the choice of response variable. It should be conceptually reasonable and executable.

Part 2

In the second part, my plan is to work on data pretreatment. During my previous projects, I learned that it would be easier to run an ANN classifier model if all input variable are numbers instead of letters. So, I plan to convert all letter records into numbers before loading the dataset into Spyder. And after I read the treated file in Spyder, I will apply more data pretreatment steps including splitting the dataset into x variable and y variable, reshape them, and make them to categorical.

Part 3

In the third part, my plan is to develop a base model and a train algorithm with K-fold cross validation where K equals 3, and from there, I will develop more models by adding number of layers, number of nodes, batch size, and epochs. Each time running an experimental model, the parameters, accuracy outputs, and plots, will all be recorded in an excel file for further comparisons and evaluations. Since there are three outputs to be generated during each experimental model by the 3-fold cross validation, an average accuracy will be calculated from the three outputs at each time.

Part 4

In the last part of my plan, I will make good evaluations based on the results generated from all the experimental models. The evaluating factors include average accuracy, output variance, and total running time.

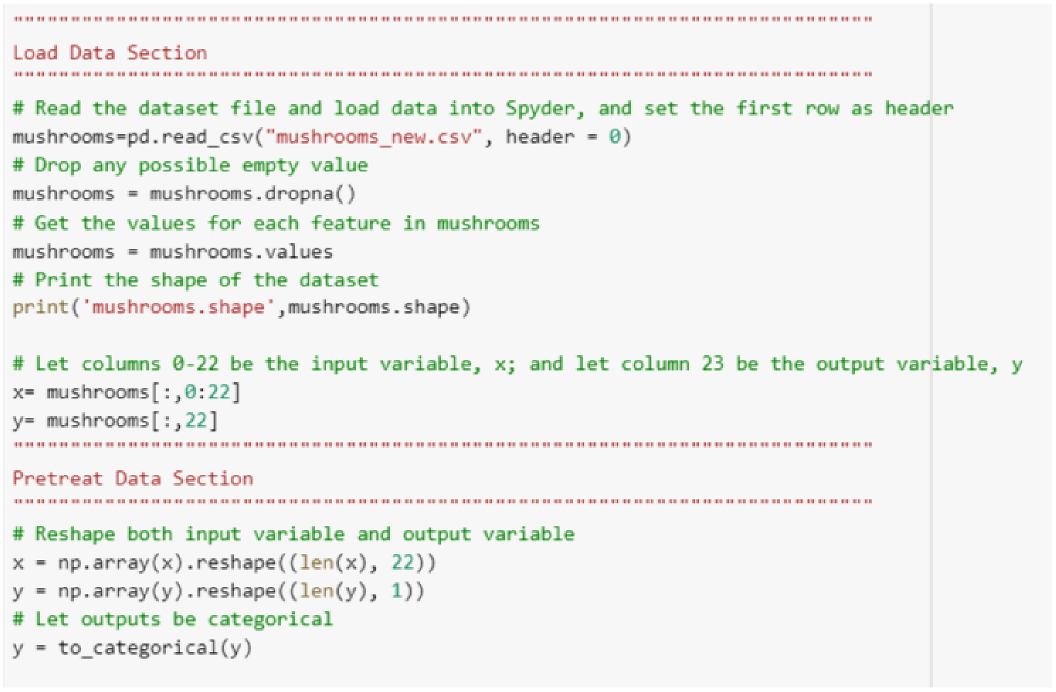
Time Spent for Experiments

The total amount of time I spent on this project and is about 8 hours. Including data pretreatment, extra experiments for the model by trying with different parameters and putting down the representative ones with significant differences.

Explanations of Input Variables

There are total of 22 input variables. Examples are cap-shape, cap-surface, cap-color, bruises, odor, gill-attachment, gill-spacing, gill-size, ring-size, population, and habitat. Each feature is recorded with a letter indicating its attribute, for example, as for cap-shape, “b” indicates bell shape, “c” indicates conical shape, “x” indicates convex shape, “f” indicates knobbed shape, and “s” indicates sunken shape, and as for cap-color, “n” indicates brown, “b” indicates buff, “c” indicates cinnamon, “g” indicates gray, “r” indicates green, “p” indicates pink, “u” indicates purple, “e” indicates red, “w” indicates white, and “y” indicates yellow. As for unknown attributes, they are marked down as “?”. There are no integers or any other kind of numbers in the original file of this dataset.

Data Pretreatment



```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# Read the dataset file and load data into Spyder, and set the first row as header
mushrooms=pd.read_csv("mushrooms_new.csv", header = 0)
# Drop any possible empty value
mushrooms = mushrooms.dropna()
# Get the values for each feature in mushrooms
mushrooms = mushrooms.values
# Print the shape of the dataset
print('mushrooms.shape',mushrooms.shape)

# Let columns 0-22 be the input variable, x; and let column 23 be the output variable, y
x= mushrooms[:,0:22]
y= mushrooms[:,22]
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# Reshape both input variable and output variable
x = np.array(x).reshape((len(x), 22))
y = np.array(y).reshape((len(y), 1))
# Let outputs be categorical
y = to_categorical(y)
```

Step 1 – Before loading the dataset into Spyder, some data cleaning is been done in Excel:

In order to better utilize the dataset, each letter record is converted into a number. More specifically, as for my binary response variable, I will convert “e”, edible, into 1, and “p”, poisonous, into 0. And as for the rest of the columns of my x variables, I will convert each letter into its alphabetic order, for example, “a” is 1, “b” is 2, and “z” is 26. And if there is an “?” in the dataset as unknown attribute, I will convert it to 0.

Step 2 – Load the dataset into Spyder by using the pd.read_csv() function in the panda package.

Step 3 – Drop any possible nulls/ NA in the dataset;

Step 4 – Get the values of the dataset;

Step 5 – Split the dataset into input x and output y;

Specifically, there are 22 features of mushroom in column 1-22, and they are set as input x; and the class/edibility, in column 23 is set as output y.

Step 6 – Reshape the data so that the width of the inputs matches the height of the output.

Step 5 – Set output variable, y, to categorical.

Explanation of Metrics and Justification for Choice

As for metrics in my experiments, I chose to use “categorical_crossentropy” for my loss function, and ‘accuracy’ for metrics. In general, categorical crossentropy is a loss function that is used for single label categorization, which means an example can belong to one class only. And as for accuracy, I use it to monitor the predicting reliability and accuracy of the results I get from the training sets.

And as for number of layers and nodes, I tend to keep them somewhat small but not too small in order to avoid overfitting and underfitting, which happens when the result tend to be exactly the same as the training set that it will only fit for the training set, and if there are a little difference in the test set, then the model won’t fit anymore. The number of layers and number of nodes that give the largest accuracy within the shortest amount of time will be chosen as my optimal model.

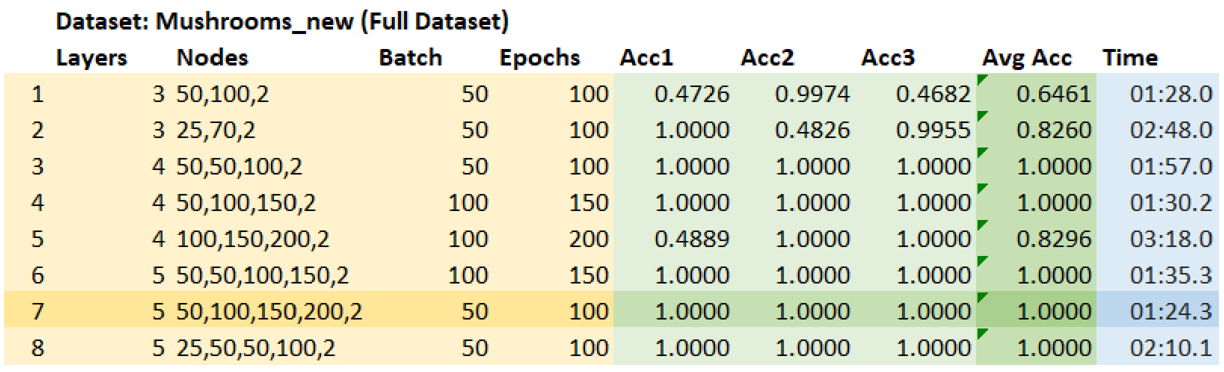
Here I experimented with 3, 4, and 5 layers, with nodes from 25 to 200.

I also change the parameters of batch size and epochs to experiment the models. By giving a smaller number of batch size, each batch of dataset will be generated more closely, resulting in more accuracy; however, if the batch size is too small, the weights will be too frequently updated to each sample, then it will take much longer time to run the model, which lowers the overall efficiency. As for batches, I tried my models with 50 – 100 batches. And as for epochs, I tried from 100 – 200. The larger epochs, the more time consuming, and less efficient.

Validation of Models

As for validation of my models, I used K-Fold Cross Validations to ensure that my choice of layers and nodes for my model was not affected by overfitting or underfitting. I chose k=3, because the smaller number of splits, the shorter time for training, which saves up efficiency; and if the number of splits is too large, then each portion of the training set will only contains a small amount of extreme sets that are not representable of the general training dataset, in this case, there will be bias, leading results to low accuracy. And by changing parameters like number of layers, number of nodes, batch size, and epochs, each set of experimenting models will provide three outputs including test accuracies and plots. In order to evaluate each round of experiment, the largest average accuracy, the smoothness of the accuracy plot, and the shortest total time taken were all taken into account.

Experiment records are listed as below:



Dataset: Mushrooms_new (Full Dataset)

	Layers	Nodes	Batch	Epochs	Acc1	Acc2	Acc3	Avg Acc	Time
1	3	50,100,2	50	100	0.4726	0.9974	0.4682	0.6461	01:28.0
2	3	25,70,2	50	100	1.0000	0.4826	0.9955	0.8260	02:48.0
3	4	50,50,100,2	50	100	1.0000	1.0000	1.0000	1.0000	01:57.0
4	4	50,100,150,2	100	150	1.0000	1.0000	1.0000	1.0000	01:30.2
5	4	100,150,200,2	100	200	0.4889	1.0000	1.0000	0.8296	03:18.0
6	5	50,50,100,150,2	100	150	1.0000	1.0000	1.0000	1.0000	01:35.3
7	5	50,100,150,200,2	50	100	1.0000	1.0000	1.0000	1.0000	01:24.3
8	5	25,50,50,100,2	50	100	1.0000	1.0000	1.0000	1.0000	02:10.1

Best Result:

Layers: **5**

Nodes: **50,100,150,200,2**

Batch Size: **50**

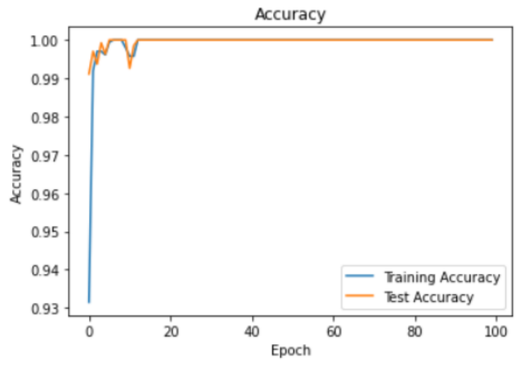
Epochs: **100**

Average Accuracy: **1.0000**

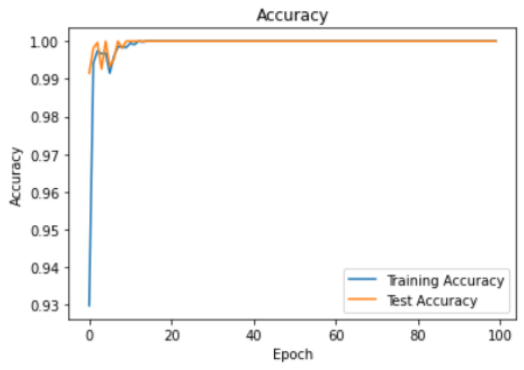
Maximum Accuracy: **1.0000**

Time taken: **01:24.3**

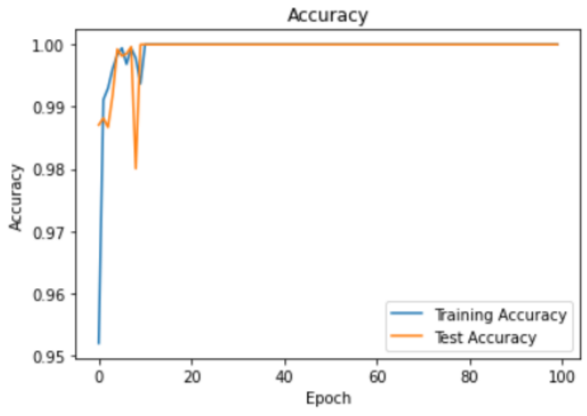
```
mushrooms.shape (8124, 23)
2708/2708 [==============================] - 0s 25us/step
Model Evaluation: [2.3061274577669803e-07, 1.0]
```



```
2708/2708 [==============================] - 0s 25us/step
Model Evaluation: [1.7467631597319377e-07, 1.0]
```



```
2708/2708 [==============================] - 0s 27us/step
Model Evaluation: [1.9475464175083284e-07, 1.0]
```



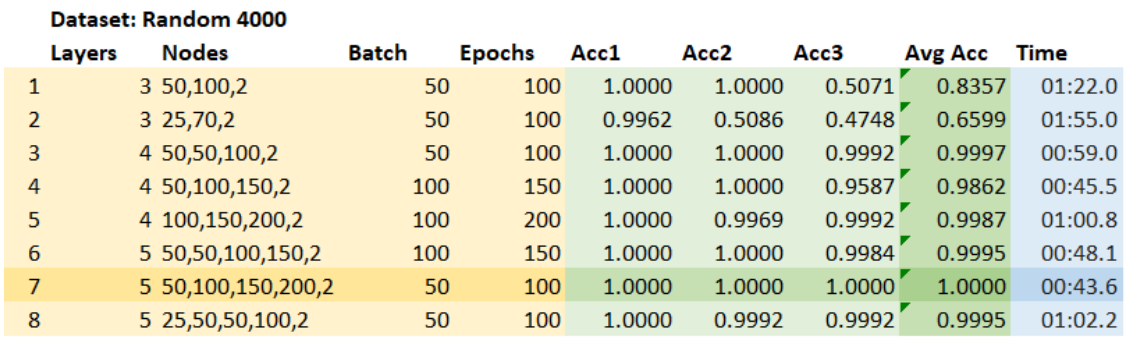
```
Time requried for training: 0:01:24.315296
```

Further Experiments for Validation Purposes

As you can see, with the full dataset, the classification accuracies tend to be 1.0 in a lot of times, so several assumptions can be made, for example, maybe there are too many samples, and less variance, so that the model is fully well trained. In order to validate my choice of model, and make sure that I could keep move forward with the best model chosen from above, I decided to do some further investigations and experiments. I first randomly selected 4000 rows from the original dataset file in excel, and copy them into a different file called "random4000". And then I use the same parameters in the models to experiment with the "random4000" dataset.

As for the procedure of generating the random 4000 rows, I first added a new column in the original data file called Random, and I give each row a random number between 0 to 1 by applying the rand() function in excel, and then I sort all the rows by the random numbers from small to large, finally, copy the first 4000 rows into the new file.

The experimenting records generated from the "Random4000" dataset are listed as below:



Dataset: Random 4000

	Layers	Nodes	Batch	Epochs	Acc1	Acc2	Acc3	Avg Acc	Time
1	3	50,100,2	50	100	1.0000	1.0000	0.5071	0.8357	01:22.0
2	3	25,70,2	50	100	0.9962	0.5086	0.4748	0.6599	01:55.0
3	4	50,50,100,2	50	100	1.0000	1.0000	0.9992	0.9997	00:59.0
4	4	50,100,150,2	100	150	1.0000	1.0000	0.9587	0.9862	00:45.5
5	4	100,150,200,2	100	200	1.0000	0.9969	0.9992	0.9987	01:00.8
6	5	50,50,100,150,2	100	150	1.0000	1.0000	0.9984	0.9995	00:48.1
7	5	50,100,150,200,2	50	100	1.0000	1.0000	1.0000	1.0000	00:43.6
8	5	25,50,50,100,2	50	100	1.0000	0.9992	0.9992	0.9995	01:02.2

Best Result: (The same choice as the previous experiment with full dataset)
Layers: **5**

Nodes: **50,100,150,200,2**

Batch Size: **50**

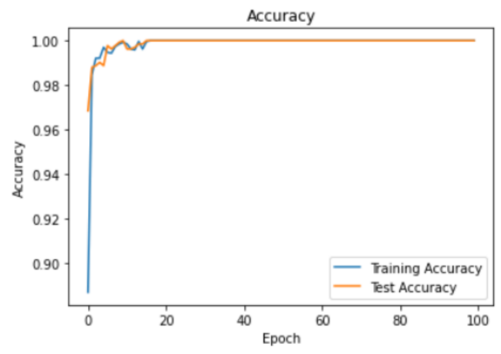
Epochs: **100**

Average Accuracy: **1.0000**

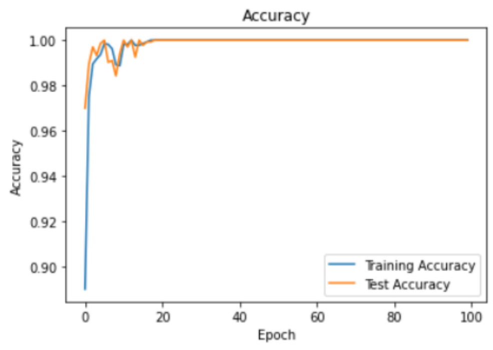
Maximum Accuracy: **1.0000**

Time taken: **00:43.6**

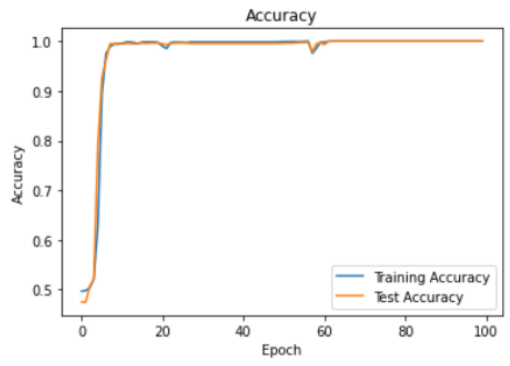
```
mushrooms.shape (4000, 23)
1334/1334 [==============================] - 0s 26us/step
Model Evaluation: [2.0347769848825137e-05, 1.0]
```



```
1333/1333 [==============================] - 0s 24us/step
Model Evaluation: [2.8558286298415275e-06, 1.0]
```



```
1333/1333 [==============================] - 0s 23us/step
Model Evaluation: [0.0002707262907829683, 1.0]
```



```
Time requried for training: 0:00:43.672423
```

Interpretation of the Results

According to the experiments from the full dataset, the optimal model attributes are: layer of 5, nodes of 50,100,150,200, and 2. The batch size is 50, and the epochs is 100. The best average accuracy given from 3-fold cross validation of this model is 1.0000. The total time taken by using this model is 01:24.3, whereas the rest of the models generally run over one and half minute.

And according to further experiments by using the datasets of random 4000 rows, the optimal model also turned out to be the same one as above. The best average accuracy is: 1.0000, and this model takes about 43 seconds to run. This model not only produces the highest accuracy in both experiments, but also runs the quickest of all. In this case, the choice of the optimal model becomes more reliable.

The reason that the ANN classification model produces such high accuracy for this mushroom dataset could be there are 8000+ rows of samples, and there are limited features within each attribute(column), indicating there are limited varieties and variance as well. So, the model is well trained. As the number of samples goes down, the accuracy goes lower too. Another reason could be that there are certain features directly representing the poisonousness of mushrooms, for example, in common sense, the colorful mushrooms are usually poisonous, and the plain white or grey mushrooms are usually edible. So, if the feature of the cap-color column indicates a bright color, then the class of that sample is very likely to be predicted as poisonous hypothetically.

Additionally, the model with very small nodes, like model no.2, with nodes of 25, 70,2, resulted to be overfitting. In this case, some extreme features might be trained too well for the training set, making the predictions for the testing set less fit.

The reason that some of the other models that were not chosen even with high accuracies is because they spent longer time running. In this case, efficiency is very important, because as the dataset becomes larger, the more time consuming it gets, and then the more efficient model can save up much more amount of time.

In conclusion, with the K-fold cross validation method where k=3, and comparing both the accuracy and efficiency of two datasets, the full dataset and the random 4000 dataset, I was able to come to a conclusion with the best Neural Network classification model that produces best accuracy in a short amount of time. In order to improve the experiment, some further investigation can be done in the future, such as splitting the dataset into 0.8 training set and 0.2 testing set, and trying with different optimizer such as 'RMSprop' and 'SGD'. Moreover, as the dataset samples get larger, it would be a great idea to implement this neural network by utilizing AWS, a power cloud ecosystem where data can be stored safely and cheaply, and ready for access as needed.

Further Discussion

In this project, backpropagation method is utilized. It is the method of tuning the weights of a neural net based on the previously obtained errors from the iterated epochs. With proper tuning of the weights of each factor would reduce error and make the model more reliable as increasing its generalization.

There are only supervised training involved in this project, as there are known response variable of each sample.

As for the training algorithm, I did not split the training set and testing set into specific proportions. Instead, I used K-fold cross validation so that the full dataset is split into equally K parts, and generating the results for k times. However, I should consider split them into proportional parts for experimentation.

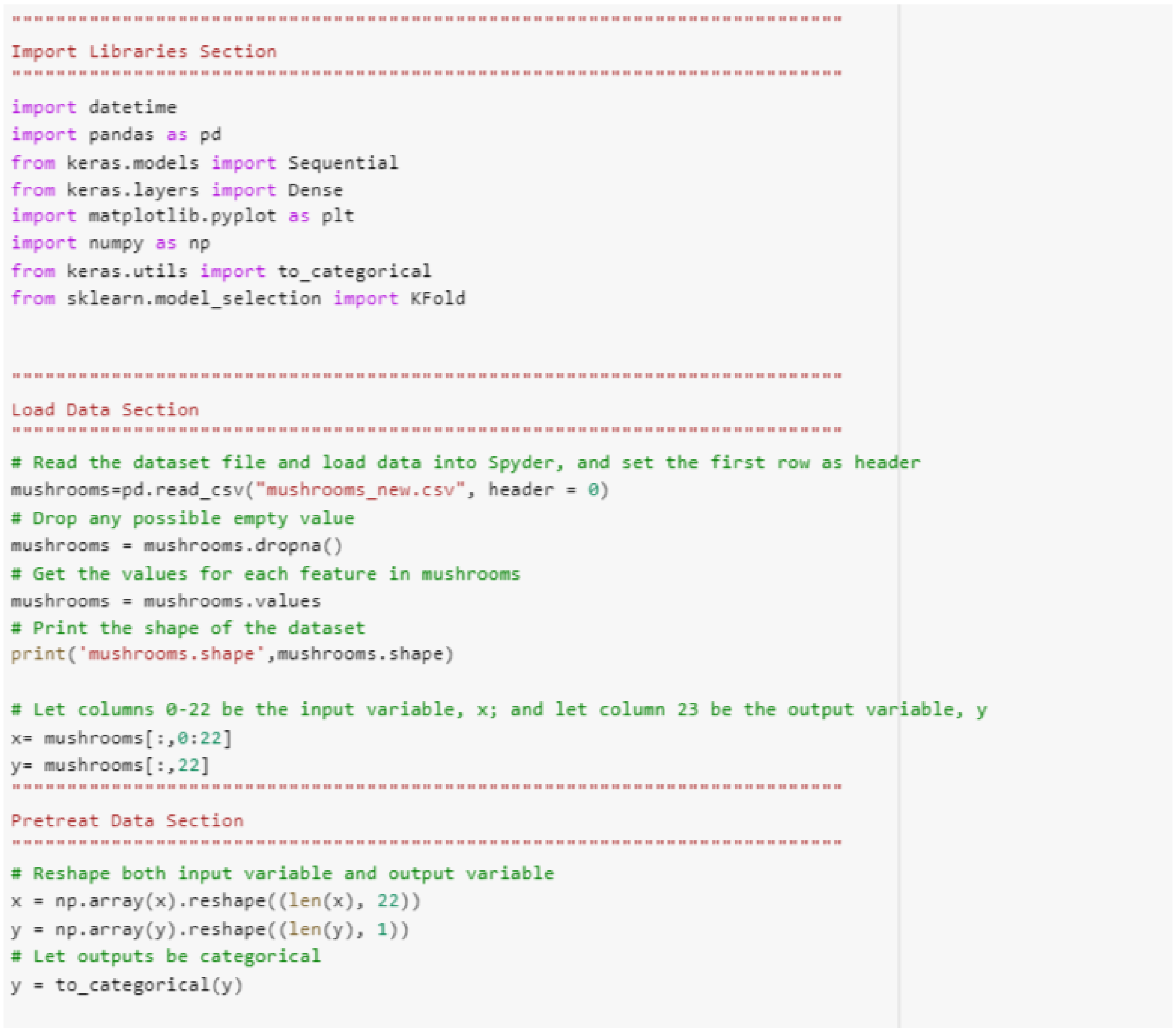
As for the input units, it was letters in the original file, but were converted into numeric units during data pretreatment process, so the output units are numeric as well. The choice of number for each letter is aligned with each letter's alphabetic order.

I change parameters of each model to run the experiments, and the parameters include number of layers, number of nodes, number of batch size, and number of repetitions/epochs. I think this is the most necessary step to apply in my experimentations. However, as for activation, I kept my choice to 'relu' and 'sigmoid'. I did not change the choice of activation, because in this project, the response variable is a binary class, in which only sigmoid is suitable for binary classification situations.

In my validation part of this project, I did try to run this model with a smaller dataset in order to ensure the reliability of my chosen model. The full dataset has 8143 rows of samples, and I generated random 4000 rows from the full dataset for further experiments. As a result, the smaller dataset did prove that my chosen model is the best one among all experimentations.

And lastly, I did not consider learning rate and momentum in this project, because only binary projections will be made, and there will be randomness during the predicting process, so learning rate and momentum would not be suitable factors in this situation.

Full Code



```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
import datetime
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np
from keras.utils import to_categorical
from sklearn.model_selection import KFold


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# Read the dataset file and load data into Spyder, and set the first row as header
mushrooms=pd.read_csv("mushrooms_new.csv", header = 0)
# Drop any possible empty value
mushrooms = mushrooms.dropna()
# Get the values for each feature in mushrooms
mushrooms = mushrooms.values
# Print the shape of the dataset
print('mushrooms.shape',mushrooms.shape)

# Let columns 0-22 be the input variable, x; and let column 23 be the output variable, y
x= mushrooms[:,0:22]
y= mushrooms[:,22]
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# Reshape both input variable and output variable
x = np.array(x).reshape((len(x), 22))
y = np.array(y).reshape((len(y), 1))
# Let outputs be categorical
y = to_categorical(y)
```

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# start counting the model running time
start_time = datetime.datetime.now()

# create a function called build_model
def build_model():
    model=Sequential()
   # add input layer, hidden layers, and output layer to the model
    model.add(Dense(50,input_dim=22, activation='relu'))
    model.add(Dense(100, activation = 'relu'))
    model.add(Dense(150, activation = 'relu'))
    model.add(Dense(200, activation = 'relu'))
    model.add(Dense(2, activation='sigmoid'))
    #compile network
    model.compile(optimizer='adam',loss='categorical_crossentropy', metrics = ['acc'])

    return model

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Train Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# split dataset into 3 parts, for K-fold cross validation use
n_split=3

# in a for loop, split dataset into train and test sets by the number of split
for train_index, test_index in KFold(n_splits=n_split, shuffle=True, random_state=None).split(x):
    x_train, x_test=x[train_index], x[test_index]
    y_train, y_test=y[train_index], y[test_index]
    model=build_model()
    #model.summary
    # set the number of batch size and epochs for the cross validation process
    history=model.fit(x_train, y_train, batch_size=50, epochs=100, verbose=0, validation_data=(x_test, y_test))
    print('Model Evaluation:', model.evaluate(x_test, y_test))
    # plot the graph, including the training set, test set, title, axis labels, and legend
    plt.figure()
    plt.plot(history.history['acc'], label='Training Accuracy')
    plt.plot(history.history['val_acc'], label='Test Accuracy')
    plt.title('Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc="lower right")
    plt.show()
```

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# End the time of model running at this point
stop_time = datetime.datetime.now()
print("Time requried for training:",stop_time - start_time)
```