Prediction of the Prescribed Drug on basis of Medical Report

Contents of the Dataset

The dataset contains information fo patients and their respective medical diagnostic report. In curing of the disease on the basis of the this medical report, each patient here responds to one of the 5 Drugs.

The following are the associated columns for the dataset.

1. Age --> Provides the age of the patient
2. Gender --> Gives the gender of the patient
3. BP --> Medical Diagnostic Report for the Blood Pressure of the patient
4. Cholestrol --> Medical Diagnostic Report for the Cholestrol Level of the patient
5. Drug --> The actual Drug to which the patient responded in the treatment

Data Related Information

The following dataset was obtained from CognitiveClass and Coursera's IBM ML Course. The data has labels which makes it well suited for Supervised Learning Methods.

The Problem Statement

The agency in this scenario want's the automate the Drug Assignment Process as in which just by feeding the patients details into the model, the drug which can be given to patient for treatment for the problem can be known. This would make sure that the Drug is provided to the patient as early as possible even if the doctors aren't available on the scene to suggest the prescribed drug for the patient.

The Problem Statement Execution Plan

The following is the execution plan to analyze the problem statement.

Loading the Dataset

The dataset is loaded in the Jupyter Notebook Environment via the !wget method provided by Jupyter.

Data Preprocessing

1. Removing Null Valued Columns
2. Removing Data Type Errors
3. Feature Selection
4. One-Hot Encoding for the categorical Columns
5. Splitting the data into a train-test split model.
6. Normalizing the dataset

Data Analysis and Wrangling

1. Understanding Column correlations.
2. Building visualizations to figure out the pattern/trends.

Building the model

1. Selecting the dependent and Independent variables
2. Builidng ML Classification Models
3. Analyzing the model accuracy for each model.
4. Building Neural Network and analyzing the network.
5. Scaling the model (if required) on Apache Spark.

Problem Conclusions

1. Checking the model with unseen data.
2. Building Visualizations to analyze the performance.
3. Concluding the problem statement with results.

Loading the dataset

In [113]:
```
!wget -O drug200.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv
```

```
--2020-02-20 15:28:53--  https://s3-api.us-geo.objectstorage.softlayer.net/cf
-courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstor
age.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.object
storage.softlayer.net)|67.228.254.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6027 (5.9K) [text/csv]
Saving to: ‘drug200.csv’

100%[======================================>] 6,027       --.-K/s   in 0s

2020-02-20 15:28:54 (657 MB/s) - ‘drug200.csv’ saved [6027/6027]
```

Importing Essential Libraries

In [114]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Viewing the data

In [115]:
```
medical = pd.read_csv('drug200.csv')
medical.head()
```

Out[115]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

Basic Data Preprocessing

In [116]:
```
# Checking for missing values

medical.isnull().sum()
```

Out[116]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

In [117]:
```
#Checking the datatypes of the columns

medical.dtypes
```

Out[117]:
```
Age              int64
Sex             object
BP              object
Cholesterol     object
Na_to_K        float64
Drug            object
dtype: object
```

In [118]:
```
#Summary for the dataset

print(medical.describe()) #Prints statistical Summary
print(medical.shape) #Prints the size of the dataset
```

```
              Age     Na_to_K
count  200.000000  200.000000
mean    44.315000   16.084485
std     16.544315    7.223956
min     15.000000    6.269000
25%     31.000000   10.445500
50%     45.000000   13.936500
75%     58.000000   19.380000
max     74.000000   38.247000
(200, 6)
```

The Data Preprocessing Conclusions

Basic Conclusions

1. The Obtained DataSet has no missing values
2. The dataypes of individual columns in the dataset is correct
3. The dataset has 200 rows and 6 columns

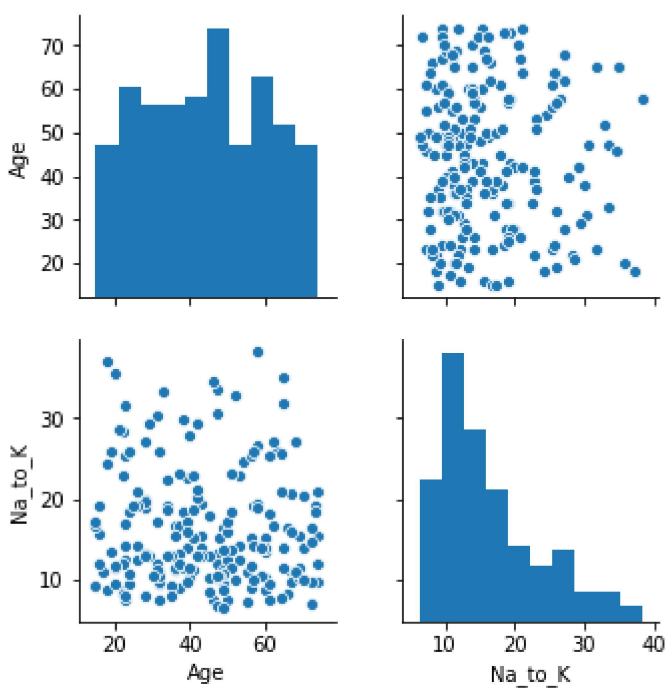
Basic Statistical Conclusions

1. The mean age for the patients in the dataset is 44 years.
2. The recorded age range is from 15-74 years with 50% data recorded is below age 45.
3. Na_to_K (Sodium to potassium level) on average is 16.
4. The maximum and minimum range of Na_to_K levels are (6-38)

Basic Data Visualizations

Plotting the Seaborn Pairplot

In [119]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(medical)
plt.show()
```



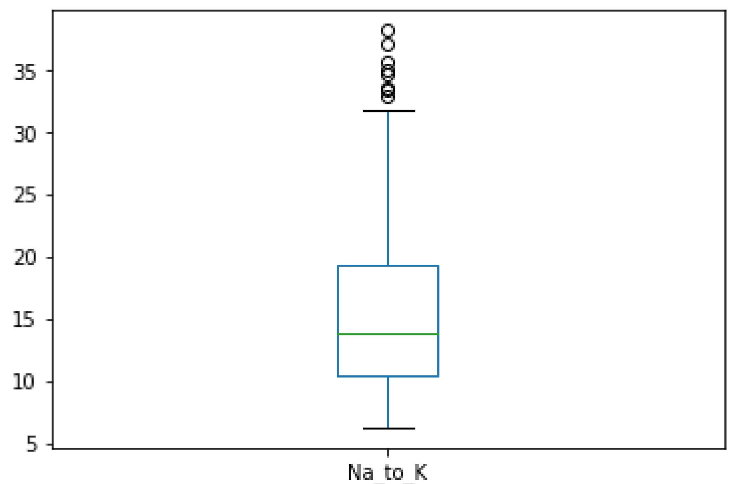
Analysis from the Pairplots

The Na_to_K (Sodium to Pottasium Level) is sparsly populated with Age. No linear/ periodic trend is avaliable between the Na_to_K level and Age. This shows that therese two columns offer a neutral correlation value.

Na_to_K Level histogram shows Positive Skewed dataset with majority of the values between 10-20 range. We would further analyse the Na_to_K level through the help of the box plot to know.

Plotting the Box Plot

In [120]:
```
medical['Na_to_K'].plot(kind='box')
plt.show()
```



The box plot shows that the mean value of Na_to_K Level in the mentioned report is nearly equal to 14. There are outliers present above the range 32 which with the help of further preprocessing is removed.

Data Wrangling - Label Indexing

In [121]:
```
#Label Indexing of all the categorical columns

from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(['NORMAL','HIGH'])
medical['Cholesterol'] = le.transform(medical['Cholesterol'])

le = preprocessing.LabelEncoder()
le.fit(['M','F'])
medical['Sex'] = le.transform(medical['Sex'])

le = preprocessing.LabelEncoder()
le.fit(['LOW','NORMAL','HIGH'])
medical['BP'] = le.transform(medical['BP'])

medical1 = medical

medical.head()
```

Out[121]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	0	0	0	25.355	drugY
1	47	1	1	0	13.093	drugC
2	47	1	1	0	10.114	drugC
3	28	0	2	0	7.798	drugX
4	61	0	1	0	18.043	drugY

Analyzing the Label Indexer Values

1. Label Indexing Sex Column (0 = Male) (1 = Female)
2. Label Indexing BP Column (0 = Low BP) (1 = Normal BP) (2 = High BP)
3. Label Indexing Cholesterol Column (0 = Normal Cholesterol) (1 = High Cholesterol)

Getting the data ready for Model Development - Feature Selection

In [122]:
```
X_data = medical[['Age','Sex','BP','Cholesterol','Na_to_K']]
Y_data = medical['Drug']

#Converting to Numpy Arrays

x = X_data.values
y = Y_data.values
```

Implementing the Train-Test Split

In [123]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random
_state=4)
```

Implementing ML Algorithm - Decision Tree

In [124]:
```
#Training the model

from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier(criterion = "entropy", max_depth=4)
decisiontree.fit(x_train, y_train)

yhat = decisiontree.predict(x_test)
```

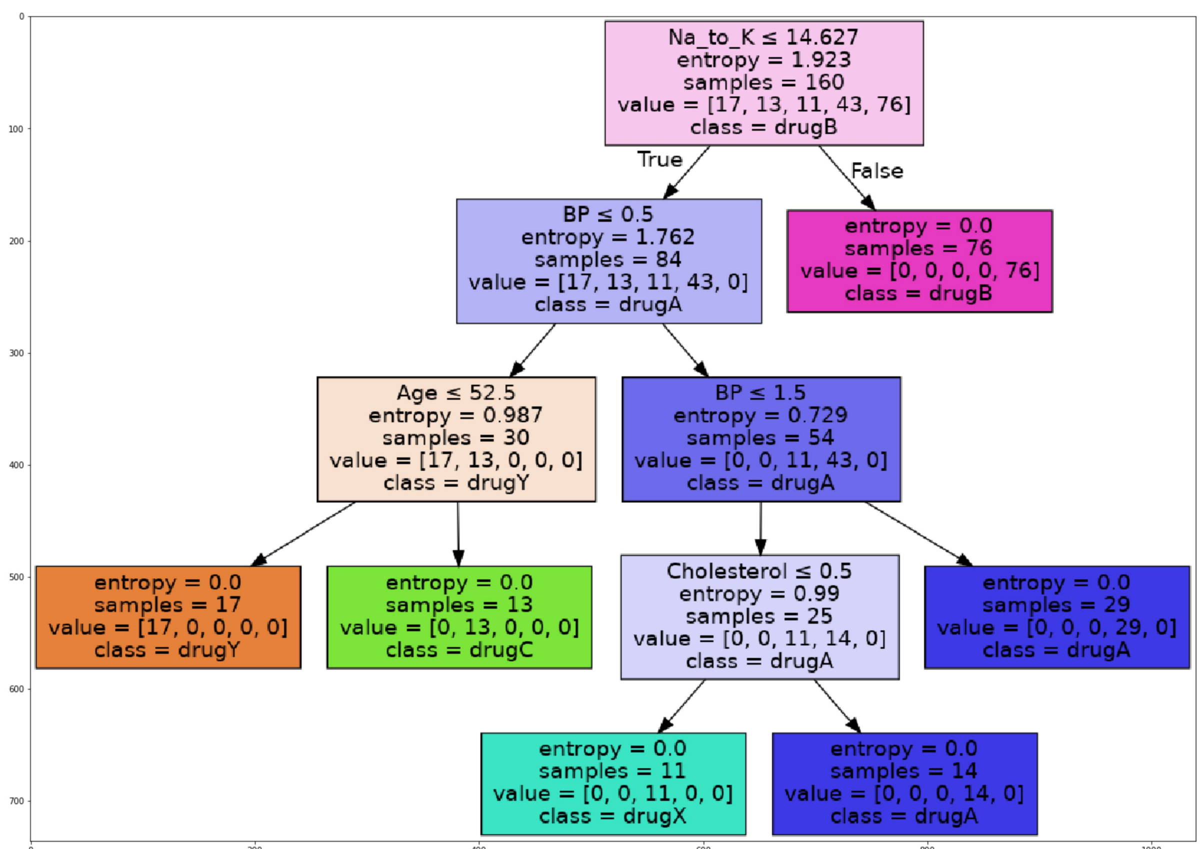
In [125]:
```
#Evalutaing the model

from sklearn import metrics
accuracy = metrics.accuracy_score(y_test,yhat)
print("The Accuracy of the model is : {}".format(accuracy))
```

```
The Accuracy of the model is : 0.95
```

In [126]:

```
#Visualizing the decision Tree

from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree

data = StringIO()
filename = "tree.png"

featureNames = X_data.columns
targetNames = medical['Drug'].unique().tolist()

out = tree.export_graphviz(decisiontree, feature_names=featureNames, out_file=
data, class_names=targetNames, filled=True, special_characters=True, rotate=Fa
lse)

graph = pydotplus.graph_from_dot_data(data.getvalue())

graph.write_png(filename)
img = mpimg.imread(filename)

plt.figure(figsize=(25,25))
plt.imshow(img, interpolation='nearest')
```

Out[126]: <matplotlib.image.AxesImage at 0x7fdc8c45b278>



Conclusions from Decision Tree Algorithm

1. Obtained Accuracy from the model - 95.0%
2. The Sodium to Pottasium Level in blood (Na_to_K) emerged as the biggest factor in Drug Prediction. Any Level greater than 14.627 was presctibed Drug B with 100% Accuracy
3. Further Analysis can be done through the decision tree generated.

Implementing a K-Means Classification Algorithm

In [127]:
```
from sklearn.neighbors import KNeighborsClassifier

k = 5
model = KNeighborsClassifier(n_neighbors=k, metric='minkowski',p=2).fit(x_trai
n,y_train)

yhat = model.predict(x_test)
```

In [128]:
```
accuracy = metrics.accuracy_score(y_test, yhat)

print("The Accuracy of the model is = {}".format(accuracy))
```

```
The Accuracy of the model is = 0.65
```

Finding the best value of K

Since the model accuracy is pretty low we with initialized value of k=5 via the elbow method we try to find the best value of k.

In [129]:
```
kvalues = 20   # Initializing values of k to be from 1-20

mean_value = np.zeros((kvalues-1))
std_value = np.zeros((kvalues-1))

confusionmatrix = []

for n in range(1, kvalues):

    model = KNeighborsClassifier(n_neighbors=k, metric='minkowski',p=2).fit(x_
train,y_train)
    yhat = model.predict(x_test)

    mean_value[n-1] = metrics.accuracy_score(y_test, yhat)
    std_value[n-1] = np.std(yhat == y_test)/np.sqrt(yhat.shape[0])


print("The best value of K = {}".format(mean_value.argmax()+1))
```

```
The best value of K = 1
```

Conclusions from the K-Means Algorithm

The model here predicted the value of K=1, which isn't suggested in the machine learning algorithm, hence the K-Means CLustering method doesn't fits the data well and fails to do classification.

Making a Neural Network

To obatin and search for higher accuracy and better model building methods, we prepare a neural network. The basic configurations of the neural network to classify the drug is as under:

Importing the Required Libraries

In [157]:

```
from keras.layers import Dense
from keras.layers import Dropout
from keras.models import Sequential

from keras.utils import to_categorical
```

Making the data ready

In [160]:

```
!wget -O drug200.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv

medical = pd.read_csv('drug200.csv')

from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(['NORMAL','HIGH'])
medical['Cholesterol'] = le.transform(medical['Cholesterol'])

le = preprocessing.LabelEncoder()
le.fit(['M','F'])
medical['Sex'] = le.transform(medical['Sex'])

le = preprocessing.LabelEncoder()
le.fit(['LOW','NORMAL','HIGH'])
medical['BP'] = le.transform(medical['BP'])

#Creating Dummies for the target variable

dummy = pd.get_dummies(medical['Drug'])
medical = pd.concat([medical,dummy],axis=1)

medical.head()
```

```
--2020-02-20 16:11:23--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)|67.228.254.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6027 (5.9K) [text/csv]
Saving to: ‘drug200.csv’

100%[======================================>] 6,027       --.-K/s   in 0s

2020-02-20 16:11:23 (569 MB/s) - ‘drug200.csv’ saved [6027/6027]
```

Out[160]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug	drugA	drugB	drugC	drugX	drugY
0	23	0	0	0	25.355	drugY	0	0	0	0	1
1	47	1	1	0	13.093	drugC	0	0	1	0	0
2	47	1	1	0	10.114	drugC	0	0	1	0	0
3	28	0	2	0	7.798	drugX	0	0	0	1	0
4	61	0	1	0	18.043	drugY	0	0	0	0	1

Feature Selection for Neural Network

In [162]:
```
X_data = medical[['Age','Sex','BP','Cholesterol','Na_to_K']]
Y_data = medical[['drugA','drugB','drugC','drugX','drugY']]

#Converting to Numpy Arrays

x = X_data.values
y = Y_data.values

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random
_state=4)
```

Training the Neural Network

In [168]:
```
model = Sequential()

model.add(Dense(200, activation='relu', input_shape=(5,)))
model.add(Dropout(0.5))
model.add(Dense(5, activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer='sgd',metrics=['accura
cy'])

model.fit(x_train,y_train, batch_size=10, epochs=50, validation_data=(x_test,y
_test))
```

```
Train on 160 samples, validate on 40 samples
Epoch 1/50
160/160 [==============================] - 18s 110ms/step - loss: 5.2489 - ac
c: 0.3938 - val_loss: 4.0171 - val_acc: 0.3750
Epoch 2/50
160/160 [==============================] - 19s 118ms/step - loss: 4.1009 - ac
c: 0.3688 - val_loss: 1.5445 - val_acc: 0.6000
Epoch 3/50
160/160 [==============================] - 20s 127ms/step - loss: 1.6296 - ac
c: 0.5000 - val_loss: 1.2752 - val_acc: 0.3750
Epoch 4/50
160/160 [==============================] - 20s 126ms/step - loss: 1.1715 - ac
c: 0.5438 - val_loss: 1.2430 - val_acc: 0.4250
Epoch 5/50
160/160 [==============================] - 20s 127ms/step - loss: 1.0428 - ac
c: 0.5813 - val_loss: 1.1638 - val_acc: 0.4500
Epoch 6/50
160/160 [==============================] - 18s 112ms/step - loss: 1.1305 - ac
c: 0.4938 - val_loss: 1.1488 - val_acc: 0.5500
Epoch 7/50
160/160 [==============================] - 19s 117ms/step - loss: 1.0826 - ac
c: 0.5438 - val_loss: 1.1900 - val_acc: 0.4750
Epoch 8/50
160/160 [==============================] - 17s 105ms/step - loss: 1.0851 - ac
c: 0.5313 - val_loss: 1.1478 - val_acc: 0.5250
Epoch 9/50
160/160 [==============================] - 18s 112ms/step - loss: 1.0432 - ac
c: 0.5313 - val_loss: 1.1482 - val_acc: 0.5250
Epoch 10/50
160/160 [==============================] - 19s 118ms/step - loss: 1.0645 - ac
c: 0.5563 - val_loss: 1.1639 - val_acc: 0.4000
Epoch 11/50
160/160 [==============================] - 19s 116ms/step - loss: 1.1421 - ac
c: 0.5688 - val_loss: 1.1433 - val_acc: 0.6000
Epoch 12/50
160/160 [==============================] - 17s 107ms/step - loss: 1.0733 - ac
c: 0.5500 - val_loss: 1.1100 - val_acc: 0.5750
Epoch 13/50
160/160 [==============================] - 19s 120ms/step - loss: 1.0480 - ac
c: 0.4938 - val_loss: 1.1331 - val_acc: 0.5750
Epoch 14/50
160/160 [==============================] - 19s 116ms/step - loss: 1.0857 - ac
c: 0.5625 - val_loss: 1.1324 - val_acc: 0.5750
Epoch 15/50
160/160 [==============================] - 16s 103ms/step - loss: 1.0443 - ac
c: 0.5688 - val_loss: 1.1668 - val_acc: 0.4750
Epoch 16/50
160/160 [==============================] - 20s 122ms/step - loss: 1.0415 - ac
c: 0.5250 - val_loss: 1.1739 - val_acc: 0.6000
Epoch 17/50
160/160 [==============================] - 18s 115ms/step - loss: 1.0854 - ac
c: 0.5500 - val_loss: 1.1525 - val_acc: 0.5000
Epoch 18/50
160/160 [==============================] - 17s 105ms/step - loss: 1.0353 - ac
c: 0.5563 - val_loss: 1.1329 - val_acc: 0.5000
Epoch 19/50
160/160 [==============================] - 15s 96ms/step - loss: 1.0436 - ac
```

```
c: 0.5750 - val_loss: 1.1251 - val_acc: 0.6000
Epoch 20/50
160/160 [==============================] - 14s 86ms/step - loss: 1.0305 - ac
c: 0.6063 - val_loss: 1.0927 - val_acc: 0.6000
Epoch 21/50
160/160 [==============================] - 17s 109ms/step - loss: 0.9826 - ac
c: 0.5625 - val_loss: 1.2112 - val_acc: 0.4500
Epoch 22/50
160/160 [==============================] - 20s 127ms/step - loss: 1.0517 - ac
c: 0.5500 - val_loss: 1.1025 - val_acc: 0.5750
Epoch 23/50
160/160 [==============================] - 19s 120ms/step - loss: 1.0772 - ac
c: 0.5813 - val_loss: 1.1050 - val_acc: 0.6000
Epoch 24/50
160/160 [==============================] - 20s 124ms/step - loss: 1.0893 - ac
c: 0.5687 - val_loss: 1.1520 - val_acc: 0.4750
Epoch 25/50
160/160 [==============================] - 21s 129ms/step - loss: 1.0316 - ac
c: 0.5875 - val_loss: 1.1336 - val_acc: 0.5750
Epoch 26/50
160/160 [==============================] - 19s 122ms/step - loss: 1.0212 - ac
c: 0.6000 - val_loss: 1.0758 - val_acc: 0.6000
Epoch 27/50
160/160 [==============================] - 17s 106ms/step - loss: 1.0265 - ac
c: 0.5625 - val_loss: 1.1074 - val_acc: 0.6000
Epoch 28/50
160/160 [==============================] - 17s 107ms/step - loss: 1.0276 - ac
c: 0.5625 - val_loss: 1.1614 - val_acc: 0.5000
Epoch 29/50
160/160 [==============================] - 16s 99ms/step - loss: 1.0168 - ac
c: 0.5813 - val_loss: 1.1131 - val_acc: 0.4750
Epoch 30/50
160/160 [==============================] - 20s 125ms/step - loss: 1.0083 - ac
c: 0.5813 - val_loss: 1.1566 - val_acc: 0.4500
Epoch 31/50
160/160 [==============================] - 17s 105ms/step - loss: 1.0191 - ac
c: 0.5625 - val_loss: 1.1145 - val_acc: 0.6000
Epoch 32/50
160/160 [==============================] - 16s 98ms/step - loss: 1.0172 - ac
c: 0.5500 - val_loss: 1.1432 - val_acc: 0.4500
Epoch 33/50
160/160 [==============================] - 16s 98ms/step - loss: 1.0512 - ac
c: 0.5750 - val_loss: 1.1565 - val_acc: 0.4500
Epoch 34/50
160/160 [==============================] - 15s 93ms/step - loss: 1.0165 - ac
c: 0.5625 - val_loss: 1.1721 - val_acc: 0.4500
Epoch 35/50
160/160 [==============================] - 15s 95ms/step - loss: 1.0171 - ac
c: 0.5375 - val_loss: 1.0967 - val_acc: 0.6250
Epoch 36/50
160/160 [==============================] - 16s 98ms/step - loss: 1.0180 - ac
c: 0.5688 - val_loss: 1.1086 - val_acc: 0.5250
Epoch 37/50
160/160 [==============================] - 20s 128ms/step - loss: 1.0157 - ac
c: 0.5750 - val_loss: 1.1072 - val_acc: 0.6500
Epoch 38/50
160/160 [==============================] - 17s 109ms/step - loss: 1.0210 - ac
```

```
c: 0.5813 - val_loss: 1.1122 - val_acc: 0.5750
Epoch 39/50
160/160 [==============================] - 16s 99ms/step - loss: 0.9828 - ac
c: 0.5938 - val_loss: 1.0863 - val_acc: 0.5500
Epoch 40/50
160/160 [==============================] - 16s 98ms/step - loss: 1.0242 - ac
c: 0.5625 - val_loss: 1.1574 - val_acc: 0.4750
Epoch 41/50
160/160 [==============================] - 18s 115ms/step - loss: 0.9539 - ac
c: 0.6125 - val_loss: 1.1212 - val_acc: 0.6000
Epoch 42/50
160/160 [==============================] - 18s 111ms/step - loss: 1.0062 - ac
c: 0.5750 - val_loss: 1.1001 - val_acc: 0.6250
Epoch 43/50
160/160 [==============================] - 15s 95ms/step - loss: 1.0280 - ac
c: 0.5875 - val_loss: 1.1183 - val_acc: 0.4750
Epoch 44/50
160/160 [==============================] - 14s 87ms/step - loss: 1.0150 - ac
c: 0.5625 - val_loss: 1.1157 - val_acc: 0.4750
Epoch 45/50
160/160 [==============================] - 16s 103ms/step - loss: 0.9949 - ac
c: 0.6000 - val_loss: 1.0816 - val_acc: 0.6250
Epoch 46/50
160/160 [==============================] - 18s 112ms/step - loss: 0.9813 - ac
c: 0.5563 - val_loss: 1.1013 - val_acc: 0.5250
Epoch 47/50
160/160 [==============================] - 18s 115ms/step - loss: 0.9580 - ac
c: 0.5688 - val_loss: 1.1203 - val_acc: 0.4500
Epoch 48/50
160/160 [==============================] - 21s 131ms/step - loss: 1.0118 - ac
c: 0.5813 - val_loss: 1.0699 - val_acc: 0.6500
Epoch 49/50
160/160 [==============================] - 19s 116ms/step - loss: 0.9969 - ac
c: 0.5813 - val_loss: 1.0638 - val_acc: 0.5500
Epoch 50/50
160/160 [==============================] - 15s 91ms/step - loss: 1.0007 - ac
c: 0.5875 - val_loss: 1.0797 - val_acc: 0.6000
```

Out[168]:
```
<keras.callbacks.History at 0x7fdc042e55f8>
```

Model Evaluation

In [170]:
```python
score = model.evaluate(x_test,y_test, verbose=0)
print(score[0])
print(score[1])
```

```
1.0797407150268554
0.6
```

Analyzing the Basic Neural Network and Conclusions

1. With the neural network trained above the accuracy was 60%
2. The Build Neural Network had 2 Dense Layers and 1 Dropout Layer

We can build a more advanced Neural Network with deeper layers and combination of activation functions to achieve a much higher accuracy.

In [ ]:

```

```