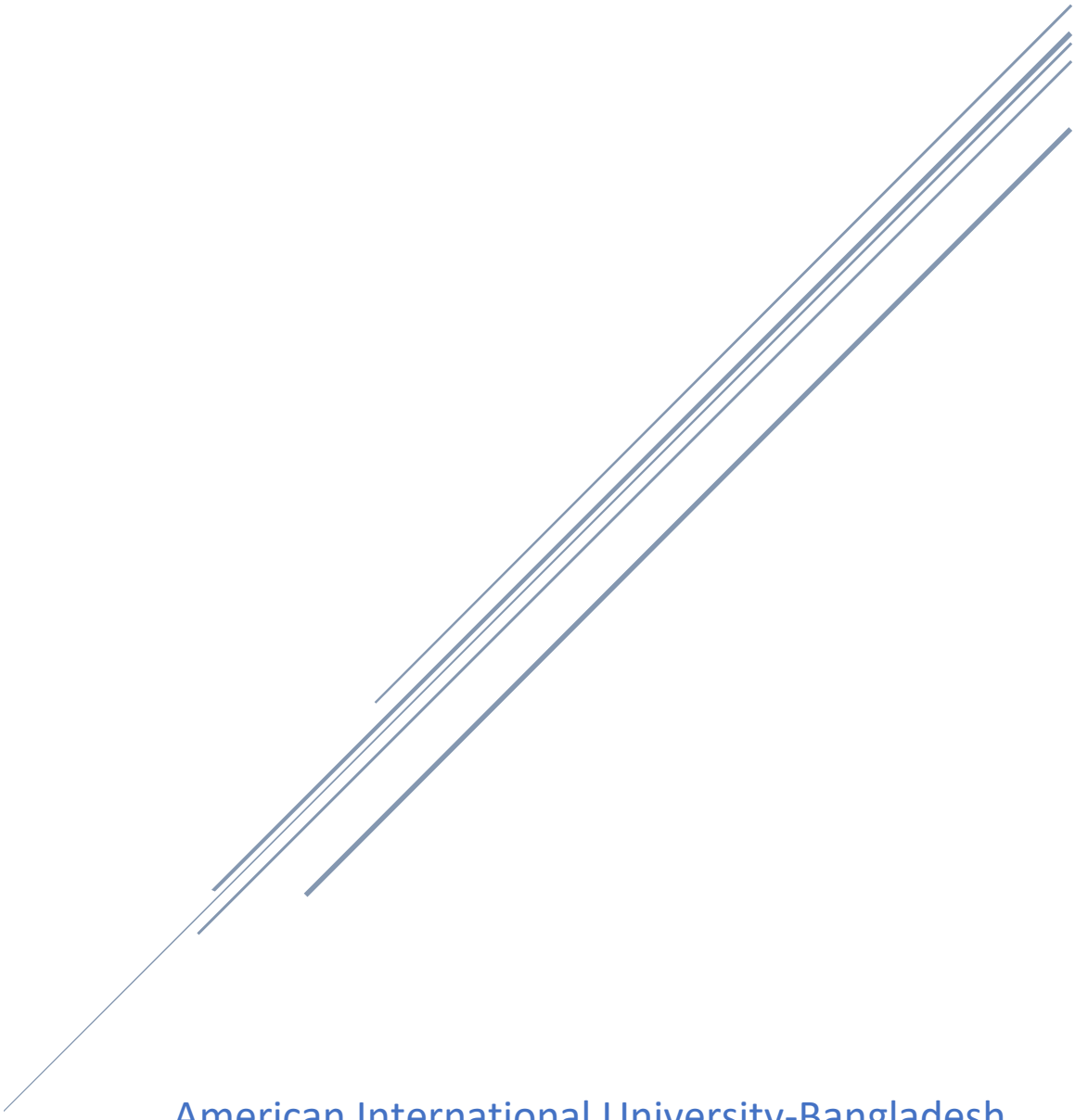
# FINAL TERM ASSIGNMENT

Name:Hossain,Mosharaf – 17-35617-3

Topic: On a connect-4 dataset, NN and KNN perform



American International University-Bangladesh

Programming in Python [A]

# Contents

Background of the problem: ..... 2
Description of Datasets: ..... 2
Explanation of KNN,NN: ..... 3
Implementation of the case in python: ..... 3
Analysis of My result: explaining graphs, plots ..... 8
Discussion: The implemented solution: ..... 2
Discussion: Overall discussion on the problem: ..... 2

## Background of the problem:

The **connect-4** dataset's data is all in string format. We can't use categorical data to construct a model in a neural network and predict K-nearest neighbors. Encoding can be used to translate it to numerical results.

Transforming the win column (class column) to 0 - 1 - 2 classes, as well as converting each layer to a number and splitting the data into features and labels. To make the prediction, train a neural network and prediction using K-nearest neighbors, divide data into training and testing sets (KNN). Finally, use the matplotlib packages to interpret the results.

## Description of Datasets:

The string format is used exclusively in the connect-4 dataset. I discovered that there are (67556, 43) 67556 rows and 43 columns using pandas. The following columns are indexed:

'b', 'b.1', 'b.2', 'b.3', 'b.4', 'b.5', 'b.6', 'b.7', 'b.8', 'b.9',

'b.10', 'b.11', 'x', 'o', 'b.12', 'b.13', 'b.14', 'b.15', 'x.1', 'o.1',

'x.2', 'o.2', 'x.3', 'o.3', 'b.16', 'b.17', 'b.18', 'b.19', 'b.20',

'b.21', 'b.22', 'b.23', 'b.24', 'b.25', 'b.26', 'b.27', 'b.28', 'b.29',

'b.30', 'b.31', 'b.32', 'b.33', ‘win.

The total number of instances is 67556.There are 42 attributes in all, each of which corresponds to one connect-4 rectangle.

Except for the win index, which has win,loss,draw, all of the indexes have three separate unique strings x,o,b. Win is a class mark that uses 42 features to forecast class. There are no null values in this connect-4 dataset.

## Explanation of KNN,NN:

**KNN:**

When all attribute values are continuous, this method is commonly used. It can be modified to deal with categorical attributes as well.

The KNN algorithm is a supervised machine learning algorithm that can solve classification and regression predictive problems. It is, however, mostly used in industry to solve classification and prediction problems.

The grouping is usually based on that of the k closest neighbors, not only the closest one. The system is then referred to as **k-Nearest Neighbour classification or simply k-NN classification.**

KNN flaw: There isn't a fully acceptable way to work with categorical attributes in KNN.

**NN:** A **neural network** is a network or circuit of neurons, or in today's terms, an artificial neural network made up of artificial neurons or nodes. Neural networks are a group of algorithms that are programmed to identify patterns and are loosely modeled after the human brain.

An artificial neural network learning algorithm, also known as a neural network or simply a neural net, is a type of artificial neural network , is a computational learning system that employs a network of functions to comprehend and convert a data input in one form into a desired output, normally in another. Human biology and the way neurons in the human brain operate together to interpret signals from human senses inspired the artificial neural network idea.

## Implementation of the case in python:

Each layer is translated to a number before the encoding process can begin. Divide the data into features and labels after that. To use NN and KNN to predict the class mark, split the data into training and trial sets.

| ##Code in Python |
| --- |
| # In[1]:<br>import numpy as np<br>import pandas as pd<br>import os<br>import tensorflow as tf<br>from tensorflow.keras.models import Sequential<br>from tensorflow.keras.layers import Dense, Conv2D, Flatten, AveragePooling2D, MaxPooling2D<br>from tensorflow.keras import losses<br>from tensorflow.keras import optimizers |

```
from sklearn.model_selection import train_test_split
from sklearn import neighbors
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
data= pd.read_csv('connect-4.data')


# In[2]:
data.head()
# In[3]:
data.tail(5)
# In[4]:
data.shape
# In[5]:
###pop and concat a column
data= pd.read_csv('connect-4.data')
td = data.pop('win')
fullData=pd.concat([data, td], axis=1)
# In[6]:
###check column all unique category
fullData.win.unique()
# In[7]:
####Mapping function encoding
###Manually convert with our own numeric value used
dataMapping={'win':0,'draw':1,'loss':2}
fullData['win']=fullData['win'].map(dataMapping)
# In[8]:
fullData.win.unique()
# In[9]:
data.columns
# In[10]:
## Label enconding
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
##for all data to excat label convert to numeric using label encoding
for x in data.columns:
    data[x]=le.fit_transform(data[x])

data.tail()
# In[11]:
###check for any null have or not
data.isnull().sum()
# In[12]:
####concate win column that are poped before
td = fullData.pop('win')
```

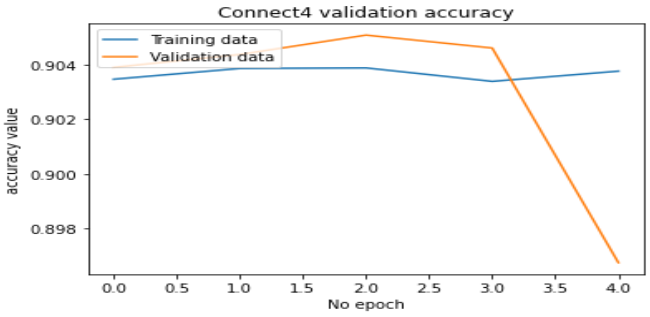
```
data=pd.concat([data, td], axis=1)
data
# In[13]:
#data.win.unique()
data["b.32"].unique()
# In[14]:
data.shape
# In[15]:
# load the dataset
fileName=data
def load_dataset(filename):
    # load the dataset as a pandas DataFrame
    data = filename
   # numpy array retrieval
    dataset = data.values
    # split into input (X) and output (y) variables
    X = dataset[:, :-1]
    y = dataset[:,-1]
    # format all fields as string
    X = X.astype(str)
    # reshape target to be a 2d array
    y = y.reshape((len(y), 1))
    return X, y


X, y = load_dataset(fileName)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, stratify=data['win'])
# summarize
print('Train', X_train.shape, y_train.shape)
print('Test', X_test.shape, y_test.shape)
# X_train
X_train.shape
# In[16]:
#####  Generate a 2D Convolutional Neural Network
def generate_CNN(conv_layers=[], dense_layers=[], lr=0.01):

    # Basic Input Layers : Conv2D layer with 4x4 filter, followed by 2x2 filter
    model = Sequential()
    model.add(Conv2D(42, (4,4), input_shape=(6,7,1), activation='tanh', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2)))

    # Add pre-flattening layers
    if len(conv_layers) != 0:
        for l in conv_layers:
            model.add(l)

    model.add(Flatten())
```

```
    # Add Post flattening layers
    if len(dense_layers) > 0:
        for l in new_layers:
            model.add(l)

    model.add(Dense(1,   activation='sigmoid'))

    # Define optimizer with provided learning rate
    adam_optimizer = optimizers.Adam(lr=lr)

    # compile and return model :
    model.compile(optimizer=adam_optimizer, loss=losses.mean_squared_error, metrics=['accuracy'])
    return model

# In[17]:
demo_model = generate_CNN()
# In[18]:
###Preprocessing Game Data
###Separate data into features and target
dataset = data.values
print(type(dataset))
features = dataset[:,:-1].copy()
target   = dataset[:,-1].copy()
for i, t in enumerate(target):
    if (t != 1):
        target[i] = 0

###
features = features.reshape(features.shape[0], 6, 7, 1)
# In[19]:
# Train test split data
X_train, X_test, y_train, y_test = train_test_split(features, target)
# In[20]:
###Fitting the Model
fittedModel=demo_model.fit(X_train, y_train, epochs=5, verbose=1, validation_data=(X_test, y_test))
fittedModel
# In[21]:
plt.plot(fittedModel.history['accuracy'], label='Training data')
plt.plot(fittedModel.history['val_accuracy'], label='Validation data')
plt.ylabel('accuracy value')
plt.xlabel('No of epoch')
plt.title('Connect4 validation accuracy')
plt.legend(loc='upper left')
plt.show()
# In[22]:
plt.plot(fittedModel.history['loss'], label='loss')
plt.plot(fittedModel.history['val_loss'], label='Validation loss')
plt.ylabel('loss accuracy value')
```

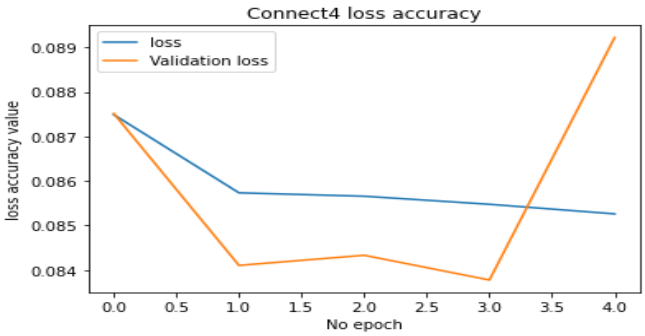
```
plt.xlabel('No of epoch')
plt.title('Connect4 loss accuracy')
plt.legend(loc='upper left')
plt.show()
# In[23]:
print(X_train.shape)
print(X_test.shape)
# In[24]:
#####For KNN
X_train = X_train.reshape(50667,6*7*1)
X_test=X_test.reshape(16889,6*7*1)
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# In[25]:
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)
# In[26]:
y_pred = classifier.predict(X_test)
# In[27]:
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

## Analysis of My result: explaining graphs, plots

In this table, we can see that train data has an accuracy of about 90%. Validation precision is about 90%. Both are very similar, indicating that the convolutional neural network (CNN) model performs well in the connect-4 dataset as well. There are a total of 5 epochs included.




At the beginning of this graph, test loss accuracy is decreasing for validation results. However, the train failure accuracy is increasing at the moment. In this model, the loss accuracy value is becoming less and less than the unforeseen loss accuracy, which is good for the model.



## Discussion: The implemented solution:

**KNN solution result:**

Confusion Matrix:[[14922 376] [ 1559 32]]

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.91 | 0.98 | 0.94 | 15298 |
| 1 | 0.08 | 0.02 | 0.03 | 1591 |
| accuracy | | | 0.89 | 16889 |
| macro avg | 0.49 | 0.50 | 0.49 | 16889 |
| weighted avg | 0.83 | 0.89 | 0.85 | 16889 |

**KNN Accuracy:** 0.8854283853395701

**NN solution result:** It also predicts the result for connect-4 dataset at a rate of about 90%, which is quite good compared to KNN and quite fast.

However, both predicted values are close to each other and reliable. As a result, the NN model is also trustworthy.

## Discussion: Overall discussion on the problem:

To make a categorical variable prediction, we must first pre-process the data by converting all of the data to numeric data and breaking it. Following the compilation of ready-to-use data that can be evaluated using NN and KNN, we can determine the most accurate expected test outcome over the training data by comparing both outcomes.