CSE 4020 Machine Learning
Lab Assessment - 3

Sujay Kumar M 20BDS0294
Computer Science Engineering with Specialization with DataScience
sujaykumarreddy.m2020@vitstudent.ac.in
https://github.com/sujaykumarmag/CSE4020

March 1, 2023

1 Making a DataSet

I made a Dataset with 21 rows

Out[3]:

	Roll No	Name	Age	DOB	CGPA	Courses	Graduation Year	Placements	M.Tech/MS	Startup
0	19BCI0876	Akhil	19	23-12-2003	8.45	8.0	2023	Yes	Yes	No
1	20BCE0076	Ram	20	3-10-2002	6.75	7.0	2024	Yes	No	Yes
2	20BDS0957	Rishab	21	2-12-2001	7.16	6.0	2024	Yes	No	No
3	20BDS0294	Sujay	20	02-12-2002	8.02	9.0	2024	No	No	Yes
4	20BCI0805	Atul	19	12-07-2003	9.14	12.0	2024	Yes	Yes	No
5	20BKT0012	Nivas	20	3-1-2002	9.54	6.0	2024	No	No	Yes
6	20BCT0121	Harshil	19	2-06-2003	8.90	5.0	2024	Yes	No	No
7	20BCI0234	Robert	20	23-1-2002	5.56	9.0	2024	No	Yes	Yes
8	20BCE0294	Richard	21	13-09-2001	6.98	8.0	2024	Yes	Yes	Yes
9	20BCE2265	Nicolas	21	17-08-2001	7.23	13.0	2024	No	No	No
10	20BCE2095	Bernard	22	27-10-2000	7.56	6.0	2024	Yes	Yes	No
11	20BCE1067	Steve	20	19-11-2002	6.90	8.0	2024	No	Yes	No
12	20BDS0398	Sanjana	20	12-05-2002	9.30	10.0	2024	Yes	No	Yes
13	20BCT0081	Misha	19	20-09-2003	8.30	12.0	2024	No	No	Yes
14	20BCI0405	Maya	20	19-10-2002	8.75	7.0	2024	No	No	NO
15	20BCI0417	Priya	19	23-07-2003	6.90	7.0	2024	NaN	NaN	NaN
16	19BDS0412	Pragun	21	13-12-2001	NaN	NaN	2023	Yes	Yes	Yes
17	19MIC0020	Telavu	21	12-08-2001	9.78	6.0	2023	Yes	No	No
18	20BCE2075	Karishma	22	10-08-2000	9.67	10.0	2024	NaN	NaN	NaN
19	20BCE1099	Lavanya	20	23-09-2002	8.50	9.0	2024	Yes	No	Yes
20	20BCE2222	Preetha	20	13-11-2002	8.23	9.0	2024	NaN	NaN	NaN
21	20BDS0165	Navya	20	13-11-2002	7.98	10.0	2024	No	Yes	Yes

2 Data Pre-Processing

1. I used Indexing Order to identify each Student name.

2.1 I want to predict the guy/girl will be placed or not

1. The attributes are Age, CGPA, Courses, GradYear
2. The predictor Placements

```
data = data.drop(["Roll No","Name","DOB","M.Tech/MS","Startup"],axis=1)
data = data.dropna(axis=0)
data.isnull().sum()
X = data.iloc[:,:4]
y = data.iloc[:,4:]
# Target Variable
y = y["Placements"].apply(lambda x : 1 if x=='Yes' else 0)
```

ATTRIBUTES

```
Age
CGPA
Courses
GradYear
```

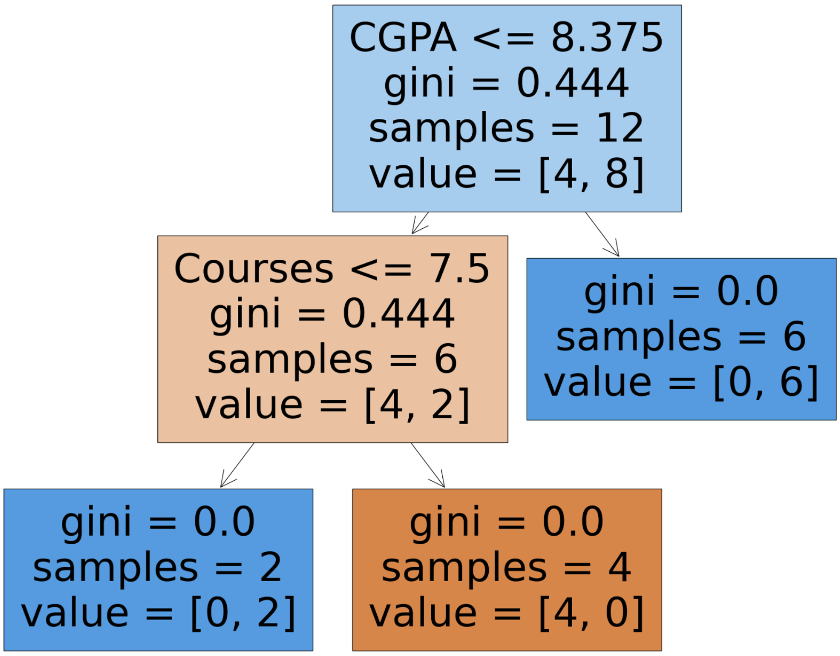
TARGET VARIABLE

```
Placements
```

3 Decision Tree Classifier by SKLEARN

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 20,
train_size = 0.7)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred_test =  clf.predict(X_test)
y_pred_train=clf.predict(X_train)


#pre-pruning



max_depth = []
acc = []
for i in range(1,30):
dt_classifier = DecisionTreeClassifier(max_depth=i, random_state = 30)
dt_classifier.fit(X_train, y_train)
pred = dt_classifier.predict(X_test)
acc.append(accuracy_score(y_test, pred))
max_depth.append(i)
print(acc)
print(max(acc))

depth = acc.index(max(acc)) + 1
dt_classifier = DecisionTreeClassifier(max_depth=depth, random_state = 20)
dt_classifier.fit(X_train, y_train)
pred = dt_classifier.predict(X_test)

#pred
accuracy_score(y_test, pred)
```

```
#params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
params = {"criterion": ["gini", "entropy"],
  "min_samples_split": [2, 5],
  "max_depth": [7],
  "min_samples_leaf": [1, 3, 5, 7],
  "max_leaf_nodes": [None, 3, 5, 7],
}
grid_search_cv = GridSearchCV(DecisionTreeClassifier(random_state=20), params,cv=3,
scoring='accuracy')

grid_search_cv.fit(X_train, y_train)


# By default, GridSearchCV trains the best model found on the whole training set (you
 can change this by setting refit=False),
#so we don't need to do it again. We can simply evaluate the model's accuracy:
y_pred = grid_search_cv.predict(X_test)
accuracy_score(y_test, y_pred)
```



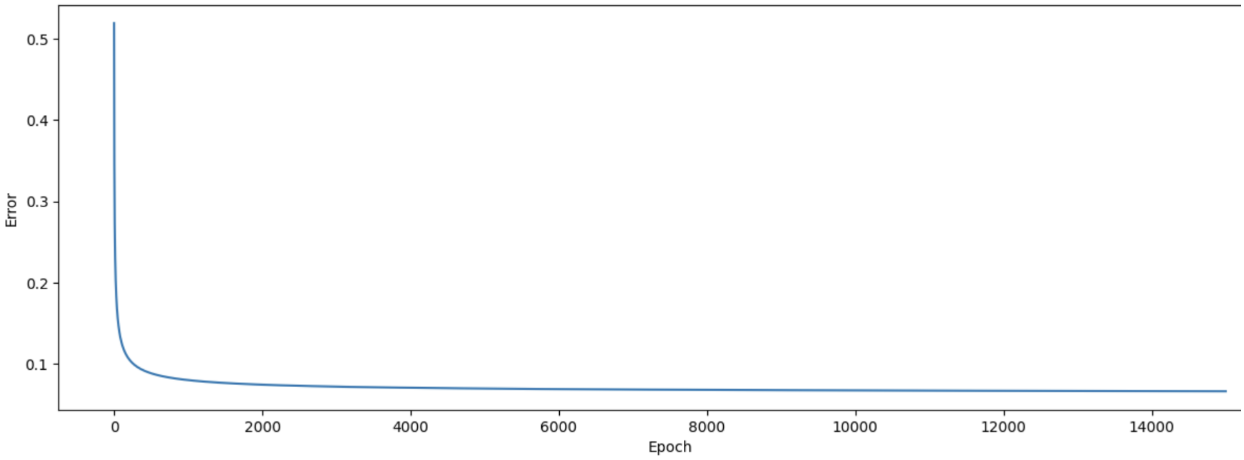
4 Neural Networks

```
class NeuralNetwork:

# constructs the neural network
def __init__(self, nn_inputs, nn_outputs, nn_epochs):
self.inputs = nn_inputs
self.outputs = nn_outputs
self.epochs = nn_epochs
self.hidden = 0
self.error = 0

# seeds e random number generator
np.random.seed(1)
# gets synaptic weights from -1 ro 1
```

```
self.synaptic_weights = 2 * np.random.random((3, 1)) - 1
self.error_history = []
self.epoch_list = []

# Using the sigmoid function
def sigmoid(self, x, derivative=False):
if not derivative:
return 1 / (1 + np.exp(-x))
else:
# returns derivative of sigmoid function
return x * (1 - x)

# data will flow through the neural network.
def feed_forward(self):
self.hidden = self.sigmoid(np.dot(self.inputs, self.synaptic_weights))

# going backwards through the network to update weights
def backpropagation(self):
self.error = self.outputs - self.hidden
delta = self.error * self.sigmoid(self.hidden, derivative=True)
self.synaptic_weights += np.dot(self.inputs.T, delta)

# trains model to make accurate predictions while continually adjusting weights
def train(self):
for epoch in range(self.epochs):
# go forward and produce an output
self.feed_forward()
# go back through the network and make corrections based on the output
self.backpropagation()

# keep track of input data
self.error_history.append(np.average(np.abs(self.error)))
self.epoch_list.append(epoch)

# function to predict output on new and unseen input data
def predict(self, new_input):
prediction = self.sigmoid(np.dot(new_input, self.synaptic_weights))
return prediction


def run(run_inputs, run_outputs, run_iterations, run_new_inputs):
# initializes neural network class
neural_network = NeuralNetwork(run_inputs, run_outputs, run_iterations)

# trains network
neural_network.train()

# print the predictions for new inputs
for i in range(len(run_new_inputs)):
print(run_new_inputs[i])
print(neural_network.predict(run_new_inputs[i]), ' - Correct: ', run_new_inputs[i][0])

# plot the error over the entire training duration
plt.figure(figsize=(15, 5))
plt.plot(neural_network.epoch_list, neural_network.error_history)
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.show()


# provides all possible datasets
def data():
data_input = []

for i1 in range(0, 2):
  for i2 in range(0, 2):
    for i3 in range(0, 2):
      data_input.append([i1, i2, i3])
  return data_input


# gets the outputs for a set of inputs
def get_outputs(get_outputs_inputs):
```

```python
    get_outputs_outputs = []
  for i in range(0, len(get_outputs_inputs)):
    get_outputs_outputs.append([get_outputs_inputs[i][0]])
  return get_outputs_outputs



```

```
[0 0 1]
[0.01179395]  - Correct:  0
[1 0 1]
[0.99987697]  - Correct:  1
[0 0 0]
[0.5]  - Correct:  0
[1 1 1]
[0.98979563]  - Correct:  1
[1 1 0]
[0.99987697]  - Correct:  1
[0 1 1]
[0.00014242]  - Correct:  0
[1 0 1]
[0.99987697]  - Correct:  1
[1 0 1]
[0.99987697]  - Correct:  1
```



5 K Nearest Neighbors

```python
from math import sqrt
class KNN():
def __init__(self,k):
self.k=k
print(self.k)

def fit(self,X_train,y_train):
self.x_train=X_train
self.y_train=y_train

def calculate_euclidean(self,sample1,sample2):
distance=0.0


def nearest_neighbors(self,test_sample):
distances=[]#calculate distances from a test sample to every sample in a training set
for i in range(len(self.x_train)):
distances.append((self.y_train[i],self.calculate_euclidean(self.x_train[i],
test_sample)))
distances.sort(key=lambda x:x[1])#sort in ascending order, based on a distance value
neighbors=[]
for i in range(self.k): #get first k samples
neighbors.append(distances[i][0])
return neighbors
```

```
def predict(self,test_set):
predictions=[]
test_set= np.array(test_set)
for test_sample in test_set:
neighbors=self.nearest_neighbors(test_sample)
labels=[sample for sample in neighbors]
prediction=max(labels,key=labels.count)
predictions.append(prediction)
return predictions
```

```
In [53]: model=KNN(5) #our model
         model.fit(np.array(X_train),np.array(y_train))

         5
```

```
In [54]: model.predict(X_test)

Out[54]: [1, 1, 1, 1, 1, 1]
```