Kartikeya Agarwal 2019UCO1692 COE-3 Machine Learning

DECISION TREE

Importing Libraries

In [ ]:

```
import pandas as pd
import numpy as np
```

Reading Dataset

In [ ]:

```
df = pd.read_csv("Train.csv")
```

In [ ]:

```
df.tail()
```

Out[ ]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked
1004	1.0	1.0	Blank, Mr. Henry	male	40.0	0.0	0.0	112277	31.0000	A31	C
1005	3.0	0.0	Laitinen, Miss. Kristina Sofia	female	37.0	0.0	0.0	4135	9.5875	NaN	S
1006	1.0	1.0	Newell, Miss. Marjorie	female	23.0	1.0	0.0	35273	113.2750	D36	C
1007	3.0	1.0	Nicola-Yarred, Master. Elias	male	12.0	1.0	0.0	2651	11.2417	NaN	C
1008	3.0	0.0	Corn, Mr. Harry	male	30.0	0.0	0.0	SOTON/OQ 392090	8.0500	NaN	S

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   pclass     1009 non-null   float64
 1   survived   1009 non-null   float64
 2   name       1009 non-null   object 
 3   sex        1009 non-null   object 
 4   age        812 non-null    float64
 5   sibsp      1009 non-null   float64
 6   parch      1009 non-null   float64
 7   ticket     1009 non-null   object 
 8   fare       1008 non-null   float64
 9   cabin      229 non-null    object 
 10  embarked   1008 non-null   object 
 11  boat       374 non-null    object 
 12  body       98 non-null     float64
```

```
 13  home.dest  582 non-null    object
dtypes: float64(7), object(7)
memory usage: 110.5+ KB
```

Dropping Redundant Columns

In [ ]:
```python
columns_to_drop = ["cabin","embarked","home.dest","name","body","boat", "ticket"]
```

In [ ]:
```python
data_clean = df.drop(columns_to_drop,axis=1)
```

In [ ]:
```python
data_clean.head()
```

Out[ ]:

	pclass	survived	sex	age	sibsp	parch	fare
0	3.0	0.0	female	NaN	0.0	0.0	7.750
1	2.0	0.0	male	39.0	0.0	0.0	26.000
2	2.0	1.0	female	40.0	0.0	0.0	13.000
3	3.0	1.0	female	31.0	1.0	1.0	20.525
4	3.0	1.0	female	NaN	2.0	0.0	23.250

Encoding Class Labels to Numeric Labels

In [ ]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_clean['sex'] = le.fit_transform(data_clean['sex'])
```

In [ ]:
```python
data_clean.head()
```

Out[ ]:

	pclass	survived	sex	age	sibsp	parch	fare
0	3.0	0.0	0	NaN	0.0	0.0	7.750
1	2.0	0.0	1	39.0	0.0	0.0	26.000
2	2.0	1.0	0	40.0	0.0	0.0	13.000
3	3.0	1.0	0	31.0	1.0	1.0	20.525
4	3.0	1.0	0	NaN	2.0	0.0	23.250

In [ ]:
```python
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   pclass    1009 non-null   float64
 1   survived  1009 non-null   float64
 2   sex       1009 non-null   int32  
 3   age       812 non-null    float64
```

```
 4   sibsp     1009 non-null   float64
 5   parch     1009 non-null   float64
 6   fare      1008 non-null   float64
dtypes: float64(6), int32(1)
memory usage: 51.4 KB
```

In [ ]:

```
data_clean = data_clean.fillna(data_clean['age'].mean())
data_clean = data_clean.fillna(data_clean['fare'].mode())
```

In [ ]:

```
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   pclass    1009 non-null   float64
 1   survived  1009 non-null   float64
 2   sex       1009 non-null   int32  
 3   age       1009 non-null   float64
 4   sibsp     1009 non-null   float64
 5   parch     1009 non-null   float64
 6   fare      1009 non-null   float64
dtypes: float64(6), int32(1)
memory usage: 51.4 KB
```

Dividing Data into X and Y

In [ ]:

```
input_cols = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare']
output_cols = ['survived']
```

In [ ]:

```
X = data_clean[input_cols]
Y = data_clean[output_cols]
```

In [ ]:

```
X.shape,Y.shape
```

Out[ ]:

```
((1009, 6), (1009, 1))
```

Defining Entropy and Information Gain

In [ ]:

```
def entropy(col):
    count = np.unique(col,return_counts=True)
    N = float(col.shape[0])
    ent = 0.0
    for ix in count[1]:
        p = ix/N
        ent+=(-1.0 * p*np.log2(p))
    return ent

def divide_data(x_data,fkey,fval):
    x_right = pd.DataFrame([],columns=x_data.columns)
    x_left = pd.DataFrame([],columns=x_data.columns)
    for ix in range(x_data.shape[0]):
        val = x_data[fkey].loc[ix]
```

```
        if val>fval:
            x_right = x_right.append(x_data.loc[ix])
        else:
            x_left = x_left.append(x_data.loc[ix])
    return x_left,x_right

def info_gain(x_data,fkey,fval):
    left,right = divide_data(x_data,fkey,fval)

    l = float(left.shape[0])/x_data.shape[0]
    r = float(right.shape[0])/x_data.shape[0]

    if(left.shape[0] == 0 or right.shape[0] == 0):
        return -100000
    i_gain = entropy(x_data.survived) - (l*entropy(left.survived) + r*entropy(right.
    return i_gain
```

In [ ]:

```
for fx in X.columns:
    print(fx)
    print(info_gain(data_clean,fx,data_clean[fx].mean()))
```

```
pclass
0.055456910002982474
sex
0.19274737190850932
age
0.001955929827451075
sibsp
0.006492394392888956
parch
0.01975608012294816
fare
0.04242793401428169
```

Implementing Decision Tree Class

In [ ]:

```
class DecisionTree:
    def __init__(self,depth = 0,max_depth = 5):
        self.left = None
        self.right = None
        self.fkey = None
        self.fval = None
        self.max_depth = max_depth
        self.depth = depth
        self.target = None

    def train(self,X_train):
        features = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare']
        info_gains = []
        for ix in features:
            i_gain = info_gain(X_train,ix,X_train[ix].mean())
            info_gains.append(i_gain)
        self.fkey = features[np.argmax(info_gains)]
        self.fval = X_train[self.fkey].mean()

        print("Making Decision Tree, Current Node is: ",self.fkey)

        data_left,data_right = divide_data(X_train,self.fkey,self.fval)
        data_left = data_left.reset_index(drop=True)
        data_right = data_right.reset_index(drop=True)
```

```
        if data_left.shape[0] == 0 or data_right.shape[0] == 0:
            if(X_train.survived.mean()>0.5):
                self.target = "Survived"
            else:
                self.target = "Dead"
            return
        if self.depth >= self.max_depth:
            if(X_train.survived.mean()>0.5):
                self.target = "Survived"
            else:
                self.target = "Dead"
            return

        self.left = DecisionTree(depth=self.depth+1,max_depth=self.max_depth)
        self.left.train(data_left)

        self.right = DecisionTree(depth=self.depth+1,max_depth=self.max_depth)
        self.right.train(data_right)

        if(X_train.survived.mean()>0.5):
            self.target = "Survived"
        else:
            self.target = "Dead"
        return

    def predict(self,test):
        if test[self.fkey] > self.fval:
            if self.right is None:
                return self.target
            return self.right.predict(test)
        else:
            if self.left is None:
                return self.target
            return self.left.predict(test)
```

Creating test and train split

In [ ]:

```
split = int(0.7*data_clean.shape[0])
train_data = data_clean[:split]
test_data = data_clean[split:]
test_data = test_data.reset_index(drop=True)
```

In [ ]:

```
print(train_data.shape)
```

```
(706, 7)
```

Training Decsion Tree

In [ ]:

```
dt = DecisionTree(max_depth=5)
dt.train(train_data)
```

```
Making Decision Tree, Current Node is:  sex
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
```

```
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
```

Predicting test data

In [ ]:

```
y_pred = []
for ix in range(test_data.shape[0]):
    y_pred.append(dt.predict(test_data.loc[ix]))
```

In [ ]:

```
y_actual = test_data[output_cols]
```

In [ ]:

```
le = LabelEncoder()
y_pred = le.fit_transform(y_pred)
```

In [ ]:

```
y_pred = np.array(y_pred).reshape((-1,1))
print(y_pred.shape)
acc = np.sum((y_pred==y_actual))/y_pred.shape[0]
```

```
(303, 1)
```

In [ ]:

```
print(acc)
```

```
survived    0.752475
dtype: float64
```

ENSEMBLE METHODS

Importing Dataset

In [ ]:

```
import pandas as pd
import numpy as np
```

In [ ]:

```
data = pd.read_csv("train.csv")
```

In [ ]:

```
data.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

Data Description

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
```

```
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Data Cleaning and Preprocessing

In [ ]:

```python
columns_to_drop = ["PassengerId", "Name", "Ticket", "Cabin", "Embarked"]
data_clean = data.drop(columns_to_drop, axis=1)
data_clean.head()
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500

Label Encoding

In [ ]:

```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_clean["Sex"] = le.fit_transform(data_clean["Sex"])
```

In [ ]:

```python
data_clean.head()
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	0	38.0	1	0	71.2833
2	1	3	0	26.0	0	0	7.9250
3	1	1	0	35.0	1	0	53.1000
4	0	3	1	35.0	0	0	8.0500

In [ ]:

```python
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Sex       891 non-null    int32  
 3   Age       714 non-null    float64
 4   SibSp     891 non-null    int64  
 5   Parch     891 non-null    int64  
 6   Fare      891 non-null    float64
dtypes: float64(2), int32(1), int64(4)
memory usage: 45.4 KB
```

In [ ]:

```
data_clean.describe()
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	0.647587	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	0.477990	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	0.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	1.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	1.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	1.000000	80.000000	8.000000	6.000000	512.329200

In [ ]:

```
data_clean = data_clean.fillna(data_clean["Age"].mean()) #Imputer can also be used
```

In [ ]:

```
data_clean.describe()
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	0.647587	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	0.477990	13.002015	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	0.000000	22.000000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	1.000000	29.699118	0.000000	0.000000	14.454200
75%	1.000000	3.000000	1.000000	35.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	1.000000	80.000000	8.000000	6.000000	512.329200

In [ ]:

```
input_cols = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare"]
output_cols = ["Survived"]
```

In [ ]:
```
X = data_clean[input_cols]
Y = data_clean[output_cols]
```

In [ ]:
```
X.shape,Y.shape
```

Out[ ]:
```
((891, 6), (891, 1))
```

Creating Test-Train Split

In [ ]:
```
split = int(0.7*data_clean.shape[0])
train_data = data_clean[:split]
test_data = data_clean[split:]
test_data = test_data.reset_index(drop=True)
```

In [ ]:
```
X_train = train_data[input_cols]
Y_train = np.array(train_data[output_cols]).reshape((-1,))
X_test = test_data[input_cols]
Y_test = np.array(test_data[output_cols]).reshape((-1,))
```

Random Forest Classfier

- Ensemble Learning

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
rf  = RandomForestClassifier(n_estimators=10, criterion='entropy', max_depth=8)
rf.fit(X_train,Y_train)
```

Out[ ]:
```
RandomForestClassifier(criterion='entropy', max_depth=8, n_estimators=10)
```

Train Accuracy

In [ ]:
```
rf.score(X_train,Y_train)
```

Out[ ]:
```
0.8940609951845907
```

Test Accuracy

In [ ]:
```
rf.score(X_test,Y_test)
```

Out[ ]:
```
0.8432835820895522
```

Cross Validation

In [ ]:

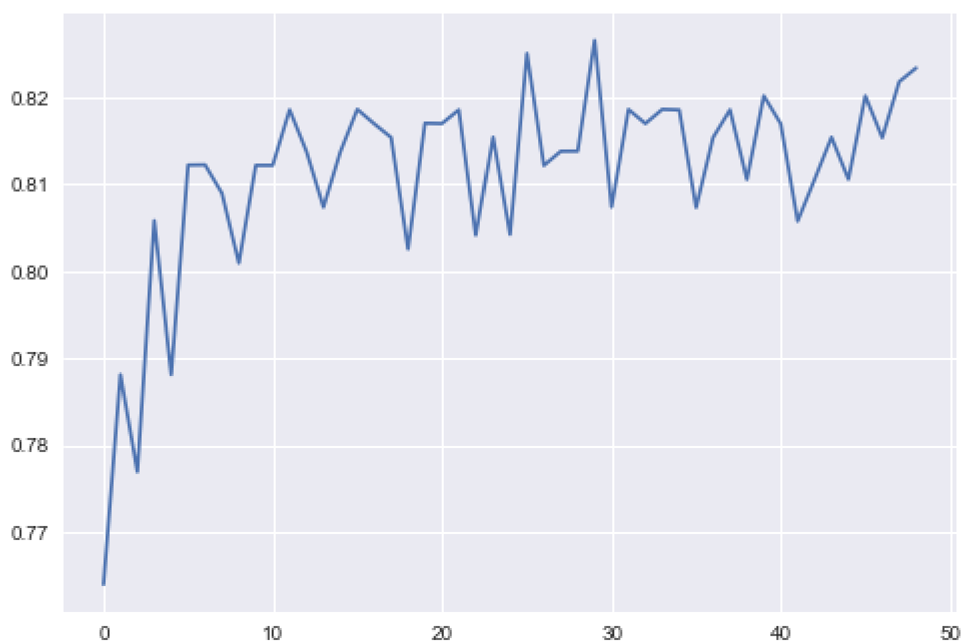
```
from sklearn.model_selection import cross_val_score
```

In [ ]:

```
acc_list = []
for i in range(1,50):
    acc = cross_val_score(RandomForestClassifier(n_estimators=i,max_depth=5,criterion='
    acc_list.append(acc)
```

In [ ]:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
plt.plot(acc_list)
plt.show()
```



In [ ]:

```
print(np.argmax(acc_list))
```

29

In [ ]:

```
rf_test = RandomForestClassifier(n_estimators=29,max_depth=5,criterion='entropy')
```

In [ ]:

```
rf_test.fit(X_train,Y_train)
rf_test.score(X_train,Y_train)
```

Out[ ]: 0.8619582664526485

In [ ]:

```
rf_test.score(X_test,Y_test)
```

Out[ ]: 0.8246268656716418

```
In [ ]:
```

BAYESIAN CLASSIFIER

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:

```
df = pd.read_csv('mushrooms.csv')
df.head(n=10)
```

Out[ ]:

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_size	gill_co
0	p	x	s	n	t	p	f	c	n	
1	e	x	s	y	t	a	f	c	b	
2	e	b	s	w	t	l	f	c	b	
3	p	x	y	w	t	p	f	c	n	
4	e	x	s	g	f	n	f	w	b	
5	e	x	y	y	t	a	f	c	b	
6	e	b	s	w	t	a	f	c	b	
7	e	b	y	w	t	l	f	c	b	
8	p	x	y	w	t	p	f	c	n	
9	e	b	s	y	t	a	f	c	b	

10 rows × 23 columns

In [ ]:

```
df.shape
```

Out[ ]:

```
(8124, 23)
```

In [ ]:

```
df.describe()
```

Out[ ]:

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_size
count	8124	8124	8124	8124	8124	8124	8124	8124	8124
unique	2	6	4	10	2	9	2	2	2
top	e	x	y	n	f	n	f	c	b
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612

4 rows × 23 columns

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

In [ ]:

```
le = LabelEncoder()
```

In [ ]:

```
ds = df.apply(le.fit_transform) # Applies transform on each collumn
```

In [ ]:

```
ds
```

Out[ ]:

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_size	gi
0	1	5	2	4	1	6	1	0	1	
1	0	5	2	9	1	0	1	0	0	
2	0	0	2	8	1	3	1	0	0	
3	1	5	3	8	1	6	1	0	1	
4	0	5	2	3	0	5	1	1	0	
...	...	...	...	...	...	...	...	...	...	
8119	0	3	2	4	0	5	0	0	0	
8120	0	5	2	4	0	5	0	0	0	
8121	0	2	2	4	0	5	0	0	0	
8122	1	3	3	4	0	8	1	0	1	
8123	0	5	2	4	0	5	0	0	0	

8124 rows × 23 columns

In [ ]:

```
data = ds.values
print(data.shape)
```

```
(8124, 23)
```

In [ ]:

```
print(data[:5,:])
data_y = data[:,0]
data_x = data[:,1:]
```

```
[[1 5 2 4 1 6 1 0 1 4 0 3 2 2 7 7 0 2 1 4 2 3 5]
 [0 5 2 9 1 0 1 0 0 4 0 2 2 2 7 7 0 2 1 4 3 2 1]
 [0 0 2 8 1 3 1 0 0 5 0 2 2 2 7 7 0 2 1 4 3 2 3]
 [1 5 3 8 1 6 1 0 1 5 0 3 2 2 7 7 0 2 1 4 2 3 5]
 [0 5 2 3 0 5 1 1 0 4 1 3 2 2 7 7 0 2 1 0 3 0 1]]
```

In [ ]:

```
x_train,x_test,y_train,y_test = train_test_split(data_x,data_y, test_size = 0.2)
```

In [ ]:

```
x_train.shape,y_train.shape
```

Out[ ]:

```
((6499, 22), (6499,))
```

In [ ]:
```
x_test.shape,y_test.shape
```
Out[ ]:
```
((1625, 22), (1625,))
```
In [ ]:
```
np.unique(y_train)
```
Out[ ]:
```
array([0, 1])
```
In [ ]:
```
a = np.array([0,0,0,1,1,0])
np.sum(a==1)
```
Out[ ]:
```
2
```
In [ ]:
```
def prior_prob(y_train,label):
    total_ex = y_train.shape[0]
    class_ex = np.sum(y_train==label)
    return (class_ex/float(total_ex))
```
In [ ]:
```
y = np.array([0,0,5,5,1,1,1,1,0,0])
```
In [ ]:
```
prior_prob(y,5)
```
Out[ ]:
```
0.2
```
In [ ]:
```
def cond_prob(x_train,y_train,feature_col,feature_val,label):
    x_filtered = x_train[y_train==label]
    numerator = np.sum(x_filtered[:,feature_col] == feature_val)
    denom = np.sum(y_train == label)
    return numerator/float(denom)
```
In [ ]:
```
def predict(x_train,y_train,xtest):
    classes = np.unique(y_train)
    n_features = x_train.shape[1]
    post_probs = []
    for label in classes:
        likelihood = 1.0
        for f in range(n_features):
            cond = cond_prob(x_train,y_train,f,xtest[f],label)
            likelihood *= cond
        prior = prior_prob(y_train,label)
        post = prior*likelihood
        post_probs.append(post)
    pred = np.argmax(post_probs)
    return pred
```
In [ ]:
```
output = predict(x_train,y_train,x_test[1])
print(output)
```
```
1
```

In [ ]:

```
print(y_test[1])
```

```
1
```

In [ ]:

```
def score(x_train,y_train,x_test,y_test):
    pred = []

    for i in range(x_test.shape[0]):
        pred_label = predict(x_train,y_train,x_test[i])
        pred.append(pred_label)
    pred = np.array(pred)
    accuracy = np.sum(pred==y_test)/y_test.shape[0]
    return accuracy
```

In [ ]:

```
print(score(x_train,y_train,x_test,y_test))
```

```
0.9987692307692307
```

In [ ]:

```

```

MULTI-LAYER PERCEPTRON

Layered Neural Network

In [ ]:
```
import numpy as np
```

In [ ]:
```
input_size = 2
layers = [4,3]
output_size = 2
```

In [ ]:
```
def softmax(a):
    ea = np.exp(a)
    ans = ea/np.sum(ea,axis=1,keepdims=True) # To preserves the dimensions
    return ans
```

In [ ]:
```
a = np.array([[20,30],[20,20]])
a_ = softmax(a)
print(a_)
```

```
[[4.53978687e-05 9.99954602e-01]
 [5.00000000e-01 5.00000000e-01]]
```

In [ ]:
```
class NeuralNetwork:
    def __init__(self, input_size, layers, output_size):
        np.random.seed(0)

        model = {}

        model['W1'] = np.random.randn(input_size, layers[0])
        model['b1'] = np.zeros((1,layers[0]))
        model['W2'] = np.random.randn(layers[0], layers[1])
        model['b2'] = np.zeros((1,layers[1]))
        model['W3'] = np.random.randn(layers[1], output_size)
        model['b3'] = np.zeros((1,output_size))

        self.model = model
        self.activation_outputs = None

    def forward(self,x):
        W1,W2,W3 = self.model['W1'],self.model['W2'],self.model['W3']
        b1,b2,b3 = self.model['b1'],self.model['b2'],self.model['b3']

        z1 = np.dot(x,W1) + b1
        a1 = np.tanh(z1)

        z2 = np.dot(a1,W2) + b2
        a2 = np.tanh(z2)

        z3 = np.dot(a2,W3) + b3
        y_ = softmax(z3)

        self.activation_outputs = (a1,a2,y_)
        return y_
```

```
    def backward(self,x,y,learning_rate=0.001):
        W1,W2,W3 = self.model['W1'],self.model['W2'],self.model['W3']
        b1,b2,b3 = self.model['b1'],self.model['b2'],self.model['b3']

        a1,a2,y_ = self.activation_outputs

        m = x.shape[0]

        delta3 = y_-y
        dw3 = np.dot(a2.T,delta3)
        db3 = np.sum(delta3,axis=0)

        delta2 = (1-np.square(a2))*np.dot(delta3,W3.T)
        dw2 = np.dot(a1.T,delta2)
        db2 = np.sum(delta2,axis=0)

        delta1 = (1-np.square(a1))*np.dot(delta2,W2.T)
        dw1 = np.dot(x.T,delta1)
        db1 = np.sum(delta1,axis=0)

        self.model["W1"] -= learning_rate*dw1
        self.model["b1"] -= learning_rate*db1

        self.model["W2"] -= learning_rate*dw2
        self.model["b2"] -= learning_rate*db2

        self.model["W3"] -= learning_rate*dw3
        self.model["b3"] -= learning_rate*db3

    def predict(self,x):
        y_out = self.forward(x)
        return np.argmax(y_out,axis=1)
    def summary(self):
        W1,W2,W3 = self.model['W1'],self.model['W2'],self.model['W3']
        a1,a2,y_ = self.activation_outputs

        print("W1 ",W1.shape)
        print("A1 ",a1.shape)

        print("W2 ",W2.shape)
        print("A2 ",a2.shape)

        print("W3 ",W3.shape)
        print("Y_ ",y_.shape)
```

In [ ]:

```
def loss(y_oht, p):
    l = -np.mean(y_oht*np.log(p))
    return l

def one_hot(y,depth):
    m = y.shape[0]
    y_oht = np.zeros((m,depth))
    y_oht[np.arange(m),y] = 1

    return y_oht
```

In [ ]:

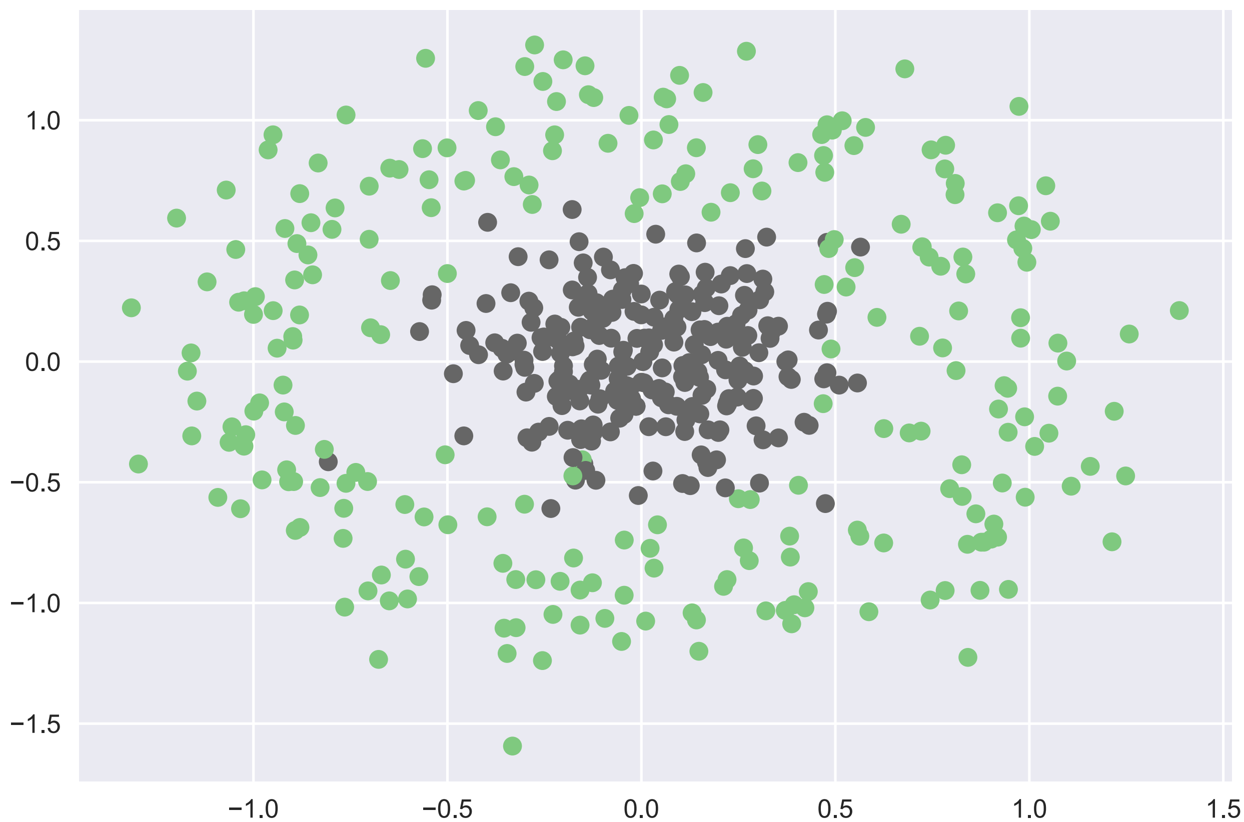
```
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
```

In [ ]:

```
x,y = make_circles(n_samples=500, shuffle=True, noise=0.2, random_state=1, factor=0.2)
```

In [ ]:

```
plt.style.use('seaborn')
plt.scatter(x[:,0],x[:,1],c=y,cmap=plt.cm.Accent)
plt.show()
```



In [ ]:

```
model = NeuralNetwork(input_size=2,layers=[10,5],output_size=2)
```

In [ ]:

```
model.forward(x)
```

Out[ ]:

```
array([[0.52335135, 0.47664865],
       [0.53144257, 0.46855743],
       [0.57726974, 0.42273026],
       [0.28383524, 0.71616476],
       [0.63877346, 0.36122654],
       [0.50841174, 0.49158826],
       [0.37442957, 0.62557043],
       [0.55888858, 0.44111142],
       [0.5711453 , 0.4288547 ],
       [0.3442594 , 0.6557406 ],
       [0.49498476, 0.50501524],
       [0.4336066 , 0.5663934 ],
       [0.36449759, 0.63550241],
       [0.37609645, 0.62390355],
       [0.51318589, 0.48681411],
       [0.40138333, 0.59861667],
       [0.4729603 , 0.5270397 ],
```

```
[0.6071061 , 0.3928939 ],
[0.45184873, 0.54815127],
[0.43976498, 0.56023502],
[0.48953223, 0.51046777],
[0.55503744, 0.44496256],
[0.58933294, 0.41066706],
[0.64427947, 0.35572053],
[0.45240396, 0.54759604],
[0.56502192, 0.43497808],
[0.5600346 , 0.4399654 ],
[0.47502227, 0.52497773],
[0.41733084, 0.58266916],
[0.70560946, 0.29439054],
[0.52971774, 0.47028226],
[0.44078622, 0.55921378],
[0.57020014, 0.42979986],
[0.55940816, 0.44059184],
[0.36558536, 0.63441464],
[0.41960119, 0.58039881],
[0.56349004, 0.43650996],
[0.55318148, 0.44681852],
[0.50403657, 0.49596343],
[0.49109595, 0.50890405],
[0.38434579, 0.61565421],
[0.51026442, 0.48973558],
[0.55430134, 0.44569866],
[0.61284912, 0.38715088],
[0.34546613, 0.65453387],
[0.38106009, 0.61893991],
[0.54236019, 0.45763981],
[0.40291746, 0.59708254],
[0.51240386, 0.48759614],
[0.31771351, 0.68228649],
[0.42921423, 0.57078577],
[0.60084599, 0.39915401],
[0.55203271, 0.44796729],
[0.49556724, 0.50443276],
[0.64412583, 0.35587417],
[0.41481768, 0.58518232],
[0.38035052, 0.61964948],
[0.51230355, 0.48769645],
[0.55415907, 0.44584093],
[0.47250842, 0.52749158],
[0.40965335, 0.59034665],
[0.33915979, 0.66084021],
[0.54112095, 0.45887905],
[0.41296734, 0.58703266],
[0.42219455, 0.57780545],
[0.5711154 , 0.4288846 ],
[0.47646583, 0.52353417],
[0.4200455 , 0.5799545 ],
[0.43884191, 0.56115809],
[0.4352312 , 0.5647688 ],
[0.4192126 , 0.5807874 ],
[0.35112087, 0.64887913],
[0.43873894, 0.56126106],
[0.56067163, 0.43932837],
[0.43146084, 0.56853916],
[0.53291098, 0.46708902],
[0.44206558, 0.55793442],
[0.67279089, 0.32720911],
```

```
[0.50581365, 0.49418635],
[0.44963369, 0.55036631],
[0.46744982, 0.53255018],
[0.43426392, 0.56573608],
[0.44675327, 0.55324673],
[0.41951816, 0.58048184],
[0.42563399, 0.57436601],
[0.59430535, 0.40569465],
[0.5142463 , 0.4857537 ],
[0.55537079, 0.44462921],
[0.50570606, 0.49429394],
[0.531001  , 0.468999  ],
[0.59301672, 0.40698328],
[0.63257471, 0.36742529],
[0.55667236, 0.44332764],
[0.54663711, 0.45336289],
[0.48906435, 0.51093565],
[0.55108849, 0.44891151],
[0.48778506, 0.51221494],
[0.3955591 , 0.6044409 ],
[0.70395266, 0.29604734],
[0.41007903, 0.58992097],
[0.595206  , 0.404794  ],
[0.57502802, 0.42497198],
[0.49990861, 0.50009139],
[0.55453143, 0.44546857],
[0.41336632, 0.58663368],
[0.32657767, 0.67342233],
[0.31982065, 0.68017935],
[0.56647183, 0.43352817],
[0.60118316, 0.39881684],
[0.40200879, 0.59799121],
[0.31921904, 0.68078096],
[0.45536278, 0.54463722],
[0.43922157, 0.56077843],
[0.5487439 , 0.4512561 ],
[0.57248702, 0.42751298],
[0.45497747, 0.54502253],
[0.58952709, 0.41047291],
[0.55919   , 0.44081   ],
[0.57752436, 0.42247564],
[0.51376349, 0.48623651],
[0.57695907, 0.42304093],
[0.57507974, 0.42492026],
[0.43284031, 0.56715969],
[0.55394398, 0.44605602],
[0.43899837, 0.56100163],
[0.53539562, 0.46460438],
[0.52794405, 0.47205595],
[0.39111917, 0.60888083],
[0.51637729, 0.48362271],
[0.51609582, 0.48390418],
[0.65947654, 0.34052346],
[0.35990678, 0.64009322],
[0.55377773, 0.44622227],
[0.29888067, 0.70111933],
[0.55737905, 0.44262095],
[0.36401406, 0.63598594],
[0.61843237, 0.38156763],
[0.63897375, 0.36102625],
[0.43663064, 0.56336936],
```

```
[0.59717216, 0.40282784],
[0.44266798, 0.55733202],
[0.41727644, 0.58272356],
[0.59589961, 0.40410039],
[0.57288091, 0.42711909],
[0.68752219, 0.31247781],
[0.44058441, 0.55941559],
[0.62877559, 0.37122441],
[0.37827391, 0.62172609],
[0.43527903, 0.56472097],
[0.55733961, 0.44266039],
[0.55462427, 0.44537573],
[0.2761095 , 0.7238905 ],
[0.51072047, 0.48927953],
[0.57828978, 0.42171022],
[0.33701561, 0.66298439],
[0.50526907, 0.49473093],
[0.6105424 , 0.3894576 ],
[0.45919344, 0.54080656],
[0.37358681, 0.62641319],
[0.46380256, 0.53619744],
[0.56604433, 0.43395567],
[0.44120016, 0.55879984],
[0.48046327, 0.51953673],
[0.51173405, 0.48826595],
[0.57652093, 0.42347907],
[0.55270607, 0.44729393],
[0.4358934 , 0.5641066 ],
[0.3702387 , 0.6297613 ],
[0.5501342 , 0.4498658 ],
[0.53081487, 0.46918513],
[0.5521419 , 0.4478581 ],
[0.38583015, 0.61416985],
[0.42502293, 0.57497707],
[0.31282538, 0.68717462],
[0.47131779, 0.52868221],
[0.41913159, 0.58086841],
[0.43190972, 0.56809028],
[0.36929662, 0.63070338],
[0.60016663, 0.39983337],
[0.38816066, 0.61183934],
[0.45662124, 0.54337876],
[0.50935841, 0.49064159],
[0.47023299, 0.52976701],
[0.44732297, 0.55267703],
[0.55548569, 0.44451431],
[0.56383585, 0.43616415],
[0.31585793, 0.68414207],
[0.43416303, 0.56583697],
[0.5648273 , 0.4351727 ],
[0.51021492, 0.48978508],
[0.61377007, 0.38622993],
[0.60239468, 0.39760532],
[0.59040181, 0.40959819],
[0.43992437, 0.56007563],
[0.50486694, 0.49513306],
[0.53652162, 0.46347838],
[0.4468124 , 0.5531876 ],
[0.68252344, 0.31747656],
[0.61064541, 0.38935459],
[0.48163448, 0.51836552],
```

```
[0.42989441, 0.57010559],
[0.54259709, 0.45740291],
[0.38371045, 0.61628955],
[0.61371916, 0.38628084],
[0.36426058, 0.63573942],
[0.47323891, 0.52676109],
[0.44393657, 0.55606343],
[0.42909281, 0.57090719],
[0.42828403, 0.57171597],
[0.53581974, 0.46418026],
[0.650624  , 0.349376  ],
[0.55209152, 0.44790848],
[0.48851104, 0.51148896],
[0.60575913, 0.39424087],
[0.44854914, 0.55145086],
[0.44549391, 0.55450609],
[0.40047992, 0.59952008],
[0.51205458, 0.48794542],
[0.39866375, 0.60133625],
[0.56288314, 0.43711686],
[0.54956091, 0.45043909],
[0.55251605, 0.44748395],
[0.51404303, 0.48595697],
[0.55390522, 0.44609478],
[0.37969402, 0.62030598],
[0.43522584, 0.56477416],
[0.547194  , 0.452806  ],
[0.42713292, 0.57286708],
[0.67733586, 0.32266414],
[0.50494876, 0.49505124],
[0.41717163, 0.58282837],
[0.3066638 , 0.6933362 ],
[0.53635615, 0.46364385],
[0.5838319 , 0.4161681 ],
[0.45887186, 0.54112814],
[0.56058086, 0.43941914],
[0.4450432 , 0.5549568 ],
[0.36468328, 0.63531672],
[0.68659253, 0.31340747],
[0.44043335, 0.55956665],
[0.41331189, 0.58668811],
[0.60855739, 0.39144261],
[0.61809255, 0.38190745],
[0.56432002, 0.43567998],
[0.52412647, 0.47587353],
[0.58769525, 0.41230475],
[0.44882967, 0.55117033],
[0.44843924, 0.55156076],
[0.44319921, 0.55680079],
[0.60475934, 0.39524066],
[0.45516791, 0.54483209],
[0.53556061, 0.46443939],
[0.56312397, 0.43687603],
[0.43380368, 0.56619632],
[0.6502643 , 0.3497357 ],
[0.57603561, 0.42396439],
[0.45522564, 0.54477436],
[0.5855823 , 0.4144177 ],
[0.38020638, 0.61979362],
[0.43517523, 0.56482477],
[0.53012694, 0.46987306],
```

```
[0.47631924, 0.52368076],
[0.6119138 , 0.3880862 ],
[0.56627003, 0.43372997],
[0.60451162, 0.39548838],
[0.38644175, 0.61355825],
[0.43440873, 0.56559127],
[0.57399026, 0.42600974],
[0.61371211, 0.38628789],
[0.56140807, 0.43859193],
[0.58639751, 0.41360249],
[0.43168955, 0.56831045],
[0.51613309, 0.48386691],
[0.44010485, 0.55989515],
[0.54112881, 0.45887119],
[0.56940601, 0.43059399],
[0.44631786, 0.55368214],
[0.38523074, 0.61476926],
[0.65044178, 0.34955822],
[0.43595678, 0.56404322],
[0.42161853, 0.57838147],
[0.27617984, 0.72382016],
[0.61656654, 0.38343346],
[0.38517627, 0.61482373],
[0.54120461, 0.45879539],
[0.57752653, 0.42247347],
[0.38325038, 0.61674962],
[0.68375373, 0.31624627],
[0.56466446, 0.43533554],
[0.58100868, 0.41899132],
[0.56263744, 0.43736256],
[0.4529878 , 0.5470122 ],
[0.37284855, 0.62715145],
[0.53556678, 0.46443322],
[0.56554551, 0.43445449],
[0.61656084, 0.38343916],
[0.5291043 , 0.4708957 ],
[0.57152611, 0.42847389],
[0.4950412 , 0.5049588 ],
[0.45924285, 0.54075715],
[0.56123121, 0.43876879],
[0.5628931 , 0.4371069 ],
[0.50660587, 0.49339413],
[0.68721729, 0.31278271],
[0.37153713, 0.62846287],
[0.56281201, 0.43718799],
[0.63482295, 0.36517705],
[0.5834555 , 0.4165445 ],
[0.4203576 , 0.5796424 ],
[0.46778673, 0.53221327],
[0.61624911, 0.38375089],
[0.51732052, 0.48267948],
[0.49785893, 0.50214107],
[0.54933002, 0.45066998],
[0.55591377, 0.44408623],
[0.45756342, 0.54243658],
[0.72698012, 0.27301988],
[0.44365491, 0.55634509],
[0.55170566, 0.44829434],
[0.6213535 , 0.3786465 ],
[0.68123294, 0.31876706],
[0.45002797, 0.54997203],
```

```
[0.54032726, 0.45967274],
[0.44924928, 0.55075072],
[0.66820182, 0.33179818],
[0.48959175, 0.51040825],
[0.49230443, 0.50769557],
[0.52414484, 0.47585516],
[0.56284075, 0.43715925],
[0.66387016, 0.33612984],
[0.51296674, 0.48703326],
[0.59633032, 0.40366968],
[0.62170365, 0.37829635],
[0.40210139, 0.59789861],
[0.45440166, 0.54559834],
[0.42723941, 0.57276059],
[0.34930697, 0.65069303],
[0.39813789, 0.60186211],
[0.5641771 , 0.4358229 ],
[0.41920002, 0.58079998],
[0.38862606, 0.61137394],
[0.44392726, 0.55607274],
[0.56848521, 0.43151479],
[0.57863448, 0.42136552],
[0.5020308 , 0.4979692 ],
[0.31188146, 0.68811854],
[0.63255461, 0.36744539],
[0.72471362, 0.27528638],
[0.42686396, 0.57313604],
[0.52437948, 0.47562052],
[0.53698358, 0.46301642],
[0.41757379, 0.58242621],
[0.70285366, 0.29714634],
[0.44801443, 0.55198557],
[0.51109076, 0.48890924],
[0.4257558 , 0.5742442 ],
[0.45115943, 0.54884057],
[0.28007947, 0.71992053],
[0.40079309, 0.59920691],
[0.56188611, 0.43811389],
[0.44780884, 0.55219116],
[0.43188322, 0.56811678],
[0.72202295, 0.27797705],
[0.44457073, 0.55542927],
[0.36651068, 0.63348932],
[0.46202617, 0.53797383],
[0.56135759, 0.43864241],
[0.60311466, 0.39688534],
[0.31623003, 0.68376997],
[0.73026279, 0.26973721],
[0.55875005, 0.44124995],
[0.31903294, 0.68096706],
[0.42099121, 0.57900879],
[0.47683823, 0.52316177],
[0.49060407, 0.50939593],
[0.57223474, 0.42776526],
[0.5920744 , 0.4079256 ],
[0.55734288, 0.44265712],
[0.5616754 , 0.4383246 ],
[0.42762831, 0.57237169],
[0.47585426, 0.52414574],
[0.59216958, 0.40783042],
[0.27617226, 0.72382774],
```

```
[0.62406787, 0.37593213],
[0.59006085, 0.40993915],
[0.43899994, 0.56100006],
[0.57272495, 0.42727505],
[0.55601016, 0.44398984],
[0.56276181, 0.43723819],
[0.44685005, 0.55314995],
[0.70043166, 0.29956834],
[0.42451388, 0.57548612],
[0.55364032, 0.44635968],
[0.39255803, 0.60744197],
[0.33340012, 0.66659988],
[0.45675619, 0.54324381],
[0.4879185 , 0.5120815 ],
[0.4373746 , 0.5626254 ],
[0.370574  , 0.629426  ],
[0.56344844, 0.43655156],
[0.71038579, 0.28961421],
[0.53606558, 0.46393442],
[0.44062785, 0.55937215],
[0.56712429, 0.43287571],
[0.58371718, 0.41628282],
[0.28527045, 0.71472955],
[0.57332497, 0.42667503],
[0.45732688, 0.54267312],
[0.49558131, 0.50441869],
[0.54487378, 0.45512622],
[0.53242977, 0.46757023],
[0.45241193, 0.54758807],
[0.42780146, 0.57219854],
[0.2828757 , 0.7171243 ],
[0.49450483, 0.50549517],
[0.44413549, 0.55586451],
[0.44777244, 0.55222756],
[0.41949334, 0.58050666],
[0.46087877, 0.53912123],
[0.44233869, 0.55766131],
[0.71251194, 0.28748806],
[0.43960796, 0.56039204],
[0.49500147, 0.50499853],
[0.41882393, 0.58117607],
[0.43055182, 0.56944818],
[0.56318551, 0.43681449],
[0.57862154, 0.42137846],
[0.61548824, 0.38451176],
[0.38128707, 0.61871293],
[0.55928571, 0.44071429],
[0.72167142, 0.27832858],
[0.28851053, 0.71148947],
[0.56064123, 0.43935877],
[0.31115107, 0.68884893],
[0.41388389, 0.58611611],
[0.38556228, 0.61443772],
[0.35229821, 0.64770179],
[0.32368072, 0.67631928],
[0.55471573, 0.44528427],
[0.38649835, 0.61350165],
[0.52626581, 0.47373419],
[0.52020475, 0.47979525],
[0.56384971, 0.43615029],
[0.58649748, 0.41350252],
```

```
       [0.57379025, 0.42620975],
       [0.44193695, 0.55806305],
       [0.55677894, 0.44322106],
       [0.58599535, 0.41400465],
       [0.56406193, 0.43593807],
       [0.45045036, 0.54954964],
       [0.44190791, 0.55809209],
       [0.44544382, 0.55455618],
       [0.66260058, 0.33739942],
       [0.63961419, 0.36038581],
       [0.55038091, 0.44961909],
       [0.71426502, 0.28573498],
       [0.48853807, 0.51146193],
       [0.40482485, 0.59517515],
       [0.57223469, 0.42776531],
       [0.49864832, 0.50135168],
       [0.56313726, 0.43686274],
       [0.54510974, 0.45489026],
       [0.43709061, 0.56290939],
       [0.55920916, 0.44079084],
       [0.57229783, 0.42770217],
       [0.44208347, 0.55791653],
       [0.29082716, 0.70917284],
       [0.50919693, 0.49080307],
       [0.56442044, 0.43557956],
       [0.6132054 , 0.3867946 ],
       [0.40327004, 0.59672996],
       [0.54654775, 0.45345225],
       [0.4594372 , 0.5405628 ],
       [0.51249158, 0.48750842],
       [0.44553399, 0.55446601],
       [0.62378736, 0.37621264],
       [0.48951755, 0.51048245],
       [0.59564205, 0.40435795],
       [0.42637161, 0.57362839],
       [0.47562394, 0.52437606],
       [0.43990445, 0.56009555],
       [0.33181008, 0.66818992],
       [0.42029777, 0.57970223],
       [0.39580255, 0.60419745],
       [0.46748259, 0.53251741],
       [0.56851961, 0.43148039],
       [0.48633496, 0.51366504],
       [0.63608743, 0.36391257],
       [0.4463488 , 0.5536512 ],
       [0.43329275, 0.56670725],
       [0.6181356 , 0.3818644 ],
       [0.396852  , 0.603148  ],
       [0.43234738, 0.56765262],
       [0.71176096, 0.28823904],
       [0.37243677, 0.62756323],
       [0.59305146, 0.40694854],
       [0.43534634, 0.56465366],
       [0.51067345, 0.48932655],
       [0.5736724 , 0.4263276 ],
       [0.5642692 , 0.4357308 ]])
```

In [ ]:

```
model.summary()
```

```
W1  (2, 10)
```

```
A1  (500, 10)
W2  (10, 5)
A2  (500, 5)
W3  (5, 2)
Y_  (500, 2)
```

In [ ]:

```
# y_oht = one_hot(y,2)
# print(y_oht)
```

In [ ]:

```
def train(X,Y,model,epochs,learning_rate,logs=True):
    training_loss = []

    classes = 2
    Y_OHT = one_hot(Y,classes)

    for ix in range(epochs):

        Y_ = model.forward(X)
        l = loss(Y_OHT,Y_)
        training_loss.append(l)
        model.backward(X,Y_OHT,learning_rate)

        if(logs):
            print("Epoch %d Loss %.4f"%(ix,l))


    return training_loss
```

In [ ]:

```
losses = train(x,y,model,500,0.001)
```

```
Epoch 0 Loss 0.3571
Epoch 1 Loss 0.3554
Epoch 2 Loss 0.2593
Epoch 3 Loss 0.2407
Epoch 4 Loss 0.2258
Epoch 5 Loss 0.2132
Epoch 6 Loss 0.2020
Epoch 7 Loss 0.1919
Epoch 8 Loss 0.1827
Epoch 9 Loss 0.1742
Epoch 10 Loss 0.1664
Epoch 11 Loss 0.1593
Epoch 12 Loss 0.1527
Epoch 13 Loss 0.1467
Epoch 14 Loss 0.1411
Epoch 15 Loss 0.1360
Epoch 16 Loss 0.1313
Epoch 17 Loss 0.1270
Epoch 18 Loss 0.1230
Epoch 19 Loss 0.1193
Epoch 20 Loss 0.1159
Epoch 21 Loss 0.1127
Epoch 22 Loss 0.1098
Epoch 23 Loss 0.1070
Epoch 24 Loss 0.1045
Epoch 25 Loss 0.1021
Epoch 26 Loss 0.0999
```

```
Epoch 27 Loss 0.0978
Epoch 28 Loss 0.0958
Epoch 29 Loss 0.0940
Epoch 30 Loss 0.0922
Epoch 31 Loss 0.0906
Epoch 32 Loss 0.0891
Epoch 33 Loss 0.0876
Epoch 34 Loss 0.0862
Epoch 35 Loss 0.0849
Epoch 36 Loss 0.0837
Epoch 37 Loss 0.0825
Epoch 38 Loss 0.0814
Epoch 39 Loss 0.0803
Epoch 40 Loss 0.0793
Epoch 41 Loss 0.0783
Epoch 42 Loss 0.0774
Epoch 43 Loss 0.0765
Epoch 44 Loss 0.0756
Epoch 45 Loss 0.0748
Epoch 46 Loss 0.0740
Epoch 47 Loss 0.0732
Epoch 48 Loss 0.0725
Epoch 49 Loss 0.0718
Epoch 50 Loss 0.0711
Epoch 51 Loss 0.0705
Epoch 52 Loss 0.0699
Epoch 53 Loss 0.0693
Epoch 54 Loss 0.0687
Epoch 55 Loss 0.0681
Epoch 56 Loss 0.0676
Epoch 57 Loss 0.0671
Epoch 58 Loss 0.0666
Epoch 59 Loss 0.0661
Epoch 60 Loss 0.0656
Epoch 61 Loss 0.0651
Epoch 62 Loss 0.0647
Epoch 63 Loss 0.0643
Epoch 64 Loss 0.0638
Epoch 65 Loss 0.0634
Epoch 66 Loss 0.0630
Epoch 67 Loss 0.0627
Epoch 68 Loss 0.0623
Epoch 69 Loss 0.0619
Epoch 70 Loss 0.0616
Epoch 71 Loss 0.0612
Epoch 72 Loss 0.0609
Epoch 73 Loss 0.0606
Epoch 74 Loss 0.0602
Epoch 75 Loss 0.0599
Epoch 76 Loss 0.0596
Epoch 77 Loss 0.0593
Epoch 78 Loss 0.0591
Epoch 79 Loss 0.0588
Epoch 80 Loss 0.0585
Epoch 81 Loss 0.0582
Epoch 82 Loss 0.0580
Epoch 83 Loss 0.0577
Epoch 84 Loss 0.0575
Epoch 85 Loss 0.0572
Epoch 86 Loss 0.0570
Epoch 87 Loss 0.0568
```

```
Epoch 88 Loss 0.0565
Epoch 89 Loss 0.0563
Epoch 90 Loss 0.0561
Epoch 91 Loss 0.0559
Epoch 92 Loss 0.0557
Epoch 93 Loss 0.0555
Epoch 94 Loss 0.0553
Epoch 95 Loss 0.0551
Epoch 96 Loss 0.0549
Epoch 97 Loss 0.0547
Epoch 98 Loss 0.0545
Epoch 99 Loss 0.0544
Epoch 100 Loss 0.0542
Epoch 101 Loss 0.0540
Epoch 102 Loss 0.0538
Epoch 103 Loss 0.0537
Epoch 104 Loss 0.0535
Epoch 105 Loss 0.0534
Epoch 106 Loss 0.0532
Epoch 107 Loss 0.0530
Epoch 108 Loss 0.0529
Epoch 109 Loss 0.0527
Epoch 110 Loss 0.0526
Epoch 111 Loss 0.0525
Epoch 112 Loss 0.0523
Epoch 113 Loss 0.0522
Epoch 114 Loss 0.0520
Epoch 115 Loss 0.0519
Epoch 116 Loss 0.0518
Epoch 117 Loss 0.0517
Epoch 118 Loss 0.0515
Epoch 119 Loss 0.0514
Epoch 120 Loss 0.0513
Epoch 121 Loss 0.0512
Epoch 122 Loss 0.0510
Epoch 123 Loss 0.0509
Epoch 124 Loss 0.0508
Epoch 125 Loss 0.0507
Epoch 126 Loss 0.0506
Epoch 127 Loss 0.0505
Epoch 128 Loss 0.0504
Epoch 129 Loss 0.0503
Epoch 130 Loss 0.0502
Epoch 131 Loss 0.0501
Epoch 132 Loss 0.0500
Epoch 133 Loss 0.0499
Epoch 134 Loss 0.0498
Epoch 135 Loss 0.0497
Epoch 136 Loss 0.0496
Epoch 137 Loss 0.0495
Epoch 138 Loss 0.0494
Epoch 139 Loss 0.0493
Epoch 140 Loss 0.0492
Epoch 141 Loss 0.0491
Epoch 142 Loss 0.0490
Epoch 143 Loss 0.0489
Epoch 144 Loss 0.0489
Epoch 145 Loss 0.0488
Epoch 146 Loss 0.0487
Epoch 147 Loss 0.0486
Epoch 148 Loss 0.0485
```

```
Epoch 149 Loss 0.0484
Epoch 150 Loss 0.0484
Epoch 151 Loss 0.0483
Epoch 152 Loss 0.0482
Epoch 153 Loss 0.0481
Epoch 154 Loss 0.0481
Epoch 155 Loss 0.0480
Epoch 156 Loss 0.0479
Epoch 157 Loss 0.0478
Epoch 158 Loss 0.0478
Epoch 159 Loss 0.0477
Epoch 160 Loss 0.0476
Epoch 161 Loss 0.0476
Epoch 162 Loss 0.0475
Epoch 163 Loss 0.0474
Epoch 164 Loss 0.0474
Epoch 165 Loss 0.0473
Epoch 166 Loss 0.0472
Epoch 167 Loss 0.0472
Epoch 168 Loss 0.0471
Epoch 169 Loss 0.0471
Epoch 170 Loss 0.0470
Epoch 171 Loss 0.0469
Epoch 172 Loss 0.0469
Epoch 173 Loss 0.0468
Epoch 174 Loss 0.0468
Epoch 175 Loss 0.0467
Epoch 176 Loss 0.0466
Epoch 177 Loss 0.0466
Epoch 178 Loss 0.0465
Epoch 179 Loss 0.0465
Epoch 180 Loss 0.0464
Epoch 181 Loss 0.0464
Epoch 182 Loss 0.0463
Epoch 183 Loss 0.0463
Epoch 184 Loss 0.0462
Epoch 185 Loss 0.0462
Epoch 186 Loss 0.0461
Epoch 187 Loss 0.0461
Epoch 188 Loss 0.0460
Epoch 189 Loss 0.0460
Epoch 190 Loss 0.0459
Epoch 191 Loss 0.0459
Epoch 192 Loss 0.0458
Epoch 193 Loss 0.0458
Epoch 194 Loss 0.0457
Epoch 195 Loss 0.0457
Epoch 196 Loss 0.0456
Epoch 197 Loss 0.0456
Epoch 198 Loss 0.0455
Epoch 199 Loss 0.0455
Epoch 200 Loss 0.0454
Epoch 201 Loss 0.0454
Epoch 202 Loss 0.0453
Epoch 203 Loss 0.0453
Epoch 204 Loss 0.0453
Epoch 205 Loss 0.0452
Epoch 206 Loss 0.0452
Epoch 207 Loss 0.0451
Epoch 208 Loss 0.0451
Epoch 209 Loss 0.0450
```

```
Epoch 210 Loss 0.0450
Epoch 211 Loss 0.0450
Epoch 212 Loss 0.0449
Epoch 213 Loss 0.0449
Epoch 214 Loss 0.0448
Epoch 215 Loss 0.0448
Epoch 216 Loss 0.0448
Epoch 217 Loss 0.0447
Epoch 218 Loss 0.0447
Epoch 219 Loss 0.0447
Epoch 220 Loss 0.0446
Epoch 221 Loss 0.0446
Epoch 222 Loss 0.0445
Epoch 223 Loss 0.0445
Epoch 224 Loss 0.0445
Epoch 225 Loss 0.0444
Epoch 226 Loss 0.0444
Epoch 227 Loss 0.0444
Epoch 228 Loss 0.0443
Epoch 229 Loss 0.0443
Epoch 230 Loss 0.0443
Epoch 231 Loss 0.0442
Epoch 232 Loss 0.0442
Epoch 233 Loss 0.0442
Epoch 234 Loss 0.0441
Epoch 235 Loss 0.0441
Epoch 236 Loss 0.0441
Epoch 237 Loss 0.0440
Epoch 238 Loss 0.0440
Epoch 239 Loss 0.0440
Epoch 240 Loss 0.0439
Epoch 241 Loss 0.0439
Epoch 242 Loss 0.0439
Epoch 243 Loss 0.0438
Epoch 244 Loss 0.0438
Epoch 245 Loss 0.0438
Epoch 246 Loss 0.0437
Epoch 247 Loss 0.0437
Epoch 248 Loss 0.0437
Epoch 249 Loss 0.0436
Epoch 250 Loss 0.0436
Epoch 251 Loss 0.0436
Epoch 252 Loss 0.0436
Epoch 253 Loss 0.0435
Epoch 254 Loss 0.0435
Epoch 255 Loss 0.0435
Epoch 256 Loss 0.0434
Epoch 257 Loss 0.0434
Epoch 258 Loss 0.0434
Epoch 259 Loss 0.0434
Epoch 260 Loss 0.0433
Epoch 261 Loss 0.0433
Epoch 262 Loss 0.0433
Epoch 263 Loss 0.0432
Epoch 264 Loss 0.0432
Epoch 265 Loss 0.0432
Epoch 266 Loss 0.0432
Epoch 267 Loss 0.0431
Epoch 268 Loss 0.0431
Epoch 269 Loss 0.0431
Epoch 270 Loss 0.0431
```

```
Epoch 271 Loss 0.0430
Epoch 272 Loss 0.0430
Epoch 273 Loss 0.0430
Epoch 274 Loss 0.0429
Epoch 275 Loss 0.0429
Epoch 276 Loss 0.0429
Epoch 277 Loss 0.0429
Epoch 278 Loss 0.0428
Epoch 279 Loss 0.0428
Epoch 280 Loss 0.0428
Epoch 281 Loss 0.0428
Epoch 282 Loss 0.0427
Epoch 283 Loss 0.0427
Epoch 284 Loss 0.0427
Epoch 285 Loss 0.0427
Epoch 286 Loss 0.0426
Epoch 287 Loss 0.0426
Epoch 288 Loss 0.0426
Epoch 289 Loss 0.0426
Epoch 290 Loss 0.0426
Epoch 291 Loss 0.0425
Epoch 292 Loss 0.0425
Epoch 293 Loss 0.0425
Epoch 294 Loss 0.0425
Epoch 295 Loss 0.0424
Epoch 296 Loss 0.0424
Epoch 297 Loss 0.0424
Epoch 298 Loss 0.0424
Epoch 299 Loss 0.0423
Epoch 300 Loss 0.0423
Epoch 301 Loss 0.0423
Epoch 302 Loss 0.0423
Epoch 303 Loss 0.0423
Epoch 304 Loss 0.0422
Epoch 305 Loss 0.0422
Epoch 306 Loss 0.0422
Epoch 307 Loss 0.0422
Epoch 308 Loss 0.0422
Epoch 309 Loss 0.0421
Epoch 310 Loss 0.0421
Epoch 311 Loss 0.0421
Epoch 312 Loss 0.0421
Epoch 313 Loss 0.0420
Epoch 314 Loss 0.0420
Epoch 315 Loss 0.0420
Epoch 316 Loss 0.0420
Epoch 317 Loss 0.0420
Epoch 318 Loss 0.0419
Epoch 319 Loss 0.0419
Epoch 320 Loss 0.0419
Epoch 321 Loss 0.0419
Epoch 322 Loss 0.0419
Epoch 323 Loss 0.0418
Epoch 324 Loss 0.0418
Epoch 325 Loss 0.0418
Epoch 326 Loss 0.0418
Epoch 327 Loss 0.0418
Epoch 328 Loss 0.0417
Epoch 329 Loss 0.0417
Epoch 330 Loss 0.0417
Epoch 331 Loss 0.0417
```

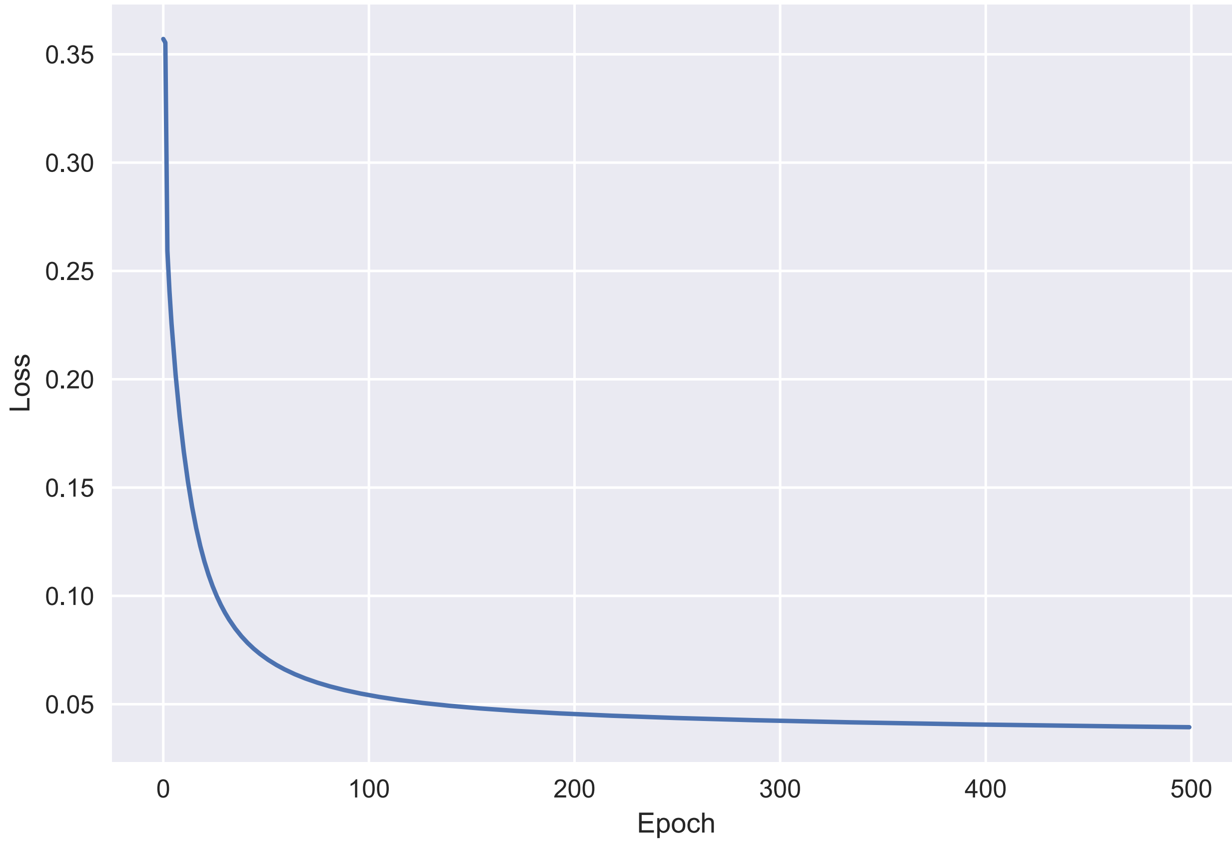
```
Epoch 332 Loss 0.0417
Epoch 333 Loss 0.0417
Epoch 334 Loss 0.0416
Epoch 335 Loss 0.0416
Epoch 336 Loss 0.0416
Epoch 337 Loss 0.0416
Epoch 338 Loss 0.0416
Epoch 339 Loss 0.0415
Epoch 340 Loss 0.0415
Epoch 341 Loss 0.0415
Epoch 342 Loss 0.0415
Epoch 343 Loss 0.0415
Epoch 344 Loss 0.0415
Epoch 345 Loss 0.0414
Epoch 346 Loss 0.0414
Epoch 347 Loss 0.0414
Epoch 348 Loss 0.0414
Epoch 349 Loss 0.0414
Epoch 350 Loss 0.0413
Epoch 351 Loss 0.0413
Epoch 352 Loss 0.0413
Epoch 353 Loss 0.0413
Epoch 354 Loss 0.0413
Epoch 355 Loss 0.0413
Epoch 356 Loss 0.0412
Epoch 357 Loss 0.0412
Epoch 358 Loss 0.0412
Epoch 359 Loss 0.0412
Epoch 360 Loss 0.0412
Epoch 361 Loss 0.0412
Epoch 362 Loss 0.0411
Epoch 363 Loss 0.0411
Epoch 364 Loss 0.0411
Epoch 365 Loss 0.0411
Epoch 366 Loss 0.0411
Epoch 367 Loss 0.0411
Epoch 368 Loss 0.0410
Epoch 369 Loss 0.0410
Epoch 370 Loss 0.0410
Epoch 371 Loss 0.0410
Epoch 372 Loss 0.0410
Epoch 373 Loss 0.0410
Epoch 374 Loss 0.0410
Epoch 375 Loss 0.0409
Epoch 376 Loss 0.0409
Epoch 377 Loss 0.0409
Epoch 378 Loss 0.0409
Epoch 379 Loss 0.0409
Epoch 380 Loss 0.0409
Epoch 381 Loss 0.0408
Epoch 382 Loss 0.0408
Epoch 383 Loss 0.0408
Epoch 384 Loss 0.0408
Epoch 385 Loss 0.0408
Epoch 386 Loss 0.0408
Epoch 387 Loss 0.0408
Epoch 388 Loss 0.0407
Epoch 389 Loss 0.0407
Epoch 390 Loss 0.0407
Epoch 391 Loss 0.0407
Epoch 392 Loss 0.0407
```

```
Epoch 393 Loss 0.0407
Epoch 394 Loss 0.0407
Epoch 395 Loss 0.0406
Epoch 396 Loss 0.0406
Epoch 397 Loss 0.0406
Epoch 398 Loss 0.0406
Epoch 399 Loss 0.0406
Epoch 400 Loss 0.0406
Epoch 401 Loss 0.0406
Epoch 402 Loss 0.0405
Epoch 403 Loss 0.0405
Epoch 404 Loss 0.0405
Epoch 405 Loss 0.0405
Epoch 406 Loss 0.0405
Epoch 407 Loss 0.0405
Epoch 408 Loss 0.0405
Epoch 409 Loss 0.0404
Epoch 410 Loss 0.0404
Epoch 411 Loss 0.0404
Epoch 412 Loss 0.0404
Epoch 413 Loss 0.0404
Epoch 414 Loss 0.0404
Epoch 415 Loss 0.0404
Epoch 416 Loss 0.0403
Epoch 417 Loss 0.0403
Epoch 418 Loss 0.0403
Epoch 419 Loss 0.0403
Epoch 420 Loss 0.0403
Epoch 421 Loss 0.0403
Epoch 422 Loss 0.0403
Epoch 423 Loss 0.0403
Epoch 424 Loss 0.0402
Epoch 425 Loss 0.0402
Epoch 426 Loss 0.0402
Epoch 427 Loss 0.0402
Epoch 428 Loss 0.0402
Epoch 429 Loss 0.0402
Epoch 430 Loss 0.0402
Epoch 431 Loss 0.0402
Epoch 432 Loss 0.0401
Epoch 433 Loss 0.0401
Epoch 434 Loss 0.0401
Epoch 435 Loss 0.0401
Epoch 436 Loss 0.0401
Epoch 437 Loss 0.0401
Epoch 438 Loss 0.0401
Epoch 439 Loss 0.0401
Epoch 440 Loss 0.0400
Epoch 441 Loss 0.0400
Epoch 442 Loss 0.0400
Epoch 443 Loss 0.0400
Epoch 444 Loss 0.0400
Epoch 445 Loss 0.0400
Epoch 446 Loss 0.0400
Epoch 447 Loss 0.0400
Epoch 448 Loss 0.0399
Epoch 449 Loss 0.0399
Epoch 450 Loss 0.0399
Epoch 451 Loss 0.0399
Epoch 452 Loss 0.0399
Epoch 453 Loss 0.0399
```

```
Epoch 454 Loss 0.0399
Epoch 455 Loss 0.0399
Epoch 456 Loss 0.0399
Epoch 457 Loss 0.0398
Epoch 458 Loss 0.0398
Epoch 459 Loss 0.0398
Epoch 460 Loss 0.0398
Epoch 461 Loss 0.0398
Epoch 462 Loss 0.0398
Epoch 463 Loss 0.0398
Epoch 464 Loss 0.0398
Epoch 465 Loss 0.0397
Epoch 466 Loss 0.0397
Epoch 467 Loss 0.0397
Epoch 468 Loss 0.0397
Epoch 469 Loss 0.0397
Epoch 470 Loss 0.0397
Epoch 471 Loss 0.0397
Epoch 472 Loss 0.0397
Epoch 473 Loss 0.0397
Epoch 474 Loss 0.0396
Epoch 475 Loss 0.0396
Epoch 476 Loss 0.0396
Epoch 477 Loss 0.0396
Epoch 478 Loss 0.0396
Epoch 479 Loss 0.0396
Epoch 480 Loss 0.0396
Epoch 481 Loss 0.0396
Epoch 482 Loss 0.0396
Epoch 483 Loss 0.0396
Epoch 484 Loss 0.0395
Epoch 485 Loss 0.0395
Epoch 486 Loss 0.0395
Epoch 487 Loss 0.0395
Epoch 488 Loss 0.0395
Epoch 489 Loss 0.0395
Epoch 490 Loss 0.0395
Epoch 491 Loss 0.0395
Epoch 492 Loss 0.0395
Epoch 493 Loss 0.0394
Epoch 494 Loss 0.0394
Epoch 495 Loss 0.0394
Epoch 496 Loss 0.0394
Epoch 497 Loss 0.0394
Epoch 498 Loss 0.0394
Epoch 499 Loss 0.0394
```

In [ ]:

```
plt.plot(losses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```

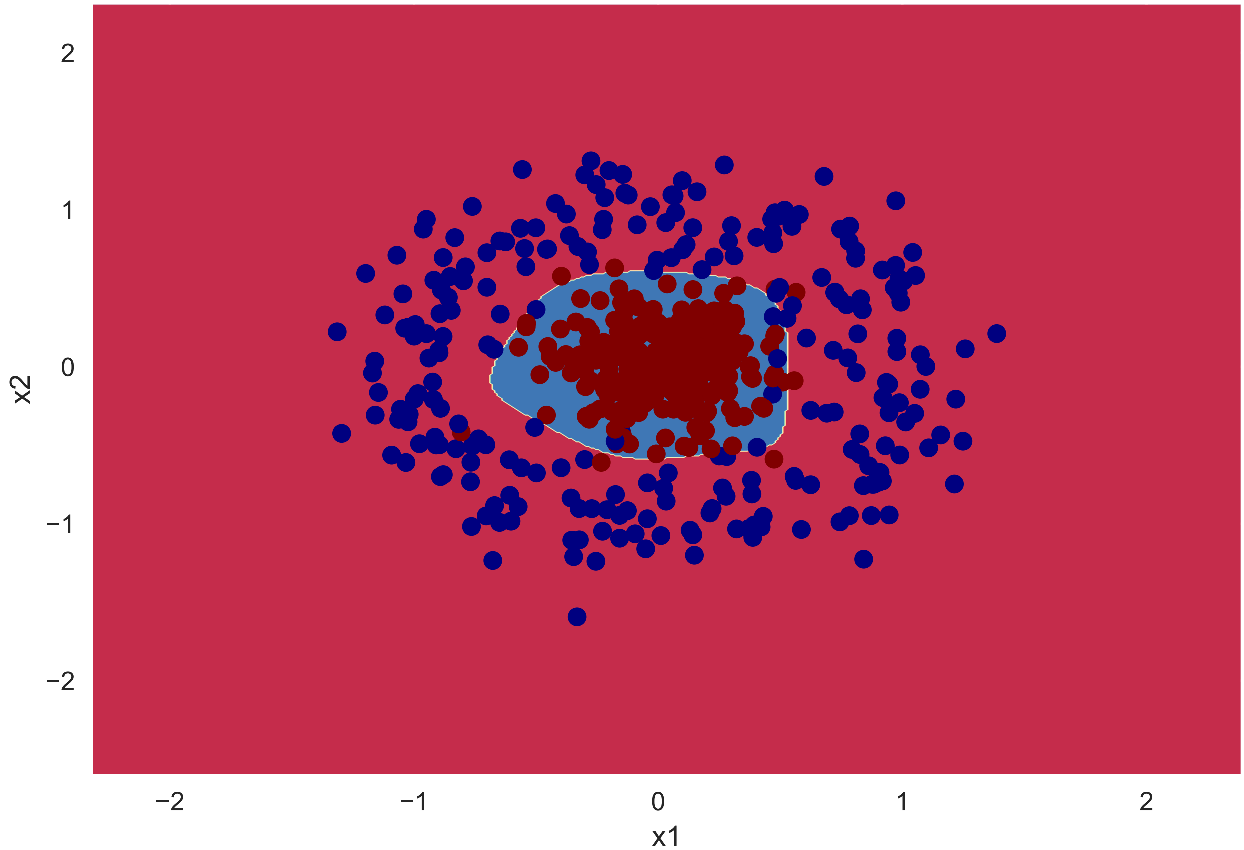


In [ ]:

```
## Find Accuracy
from visualize import plot_decision_boundary
```

In [ ]:

```
plot_decision_boundary(lambda X:model.predict(X),x,y)
```



In [ ]:

```
outputs = model.predict(x)
```

In [ ]:

```
training_accuracy = np.sum(outputs==y)/y.shape[0]
print("Training Accuracy %.4f"%(training_accuracy*100))
```

```
Training Accuracy 97.0000
```

Other Data Sets

In [ ]:

```
model  = NeuralNetwork(input_size=2,layers=[10,5],output_size=2)
```

XOR Data Set

In [ ]:

```
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]
             ])

Y = np.array([0,1,1,0])
```

In [ ]:

```
losses = train(X,Y,model,300,0.1)
```

```
Epoch 0 Loss 0.3427
```

```
Epoch 1 Loss 0.2543
Epoch 2 Loss 0.2126
Epoch 3 Loss 0.1926
Epoch 4 Loss 0.1778
Epoch 5 Loss 0.1639
Epoch 6 Loss 0.1493
Epoch 7 Loss 0.1346
Epoch 8 Loss 0.1207
Epoch 9 Loss 0.1089
Epoch 10 Loss 0.1008
Epoch 11 Loss 0.1004
Epoch 12 Loss 0.1244
Epoch 13 Loss 0.1949
Epoch 14 Loss 0.3900
Epoch 15 Loss 0.1355
Epoch 16 Loss 0.0958
Epoch 17 Loss 0.0762
Epoch 18 Loss 0.0636
Epoch 19 Loss 0.0552
Epoch 20 Loss 0.0492
Epoch 21 Loss 0.0447
Epoch 22 Loss 0.0412
Epoch 23 Loss 0.0384
Epoch 24 Loss 0.0361
Epoch 25 Loss 0.0341
Epoch 26 Loss 0.0323
Epoch 27 Loss 0.0307
Epoch 28 Loss 0.0293
Epoch 29 Loss 0.0281
Epoch 30 Loss 0.0269
Epoch 31 Loss 0.0258
Epoch 32 Loss 0.0248
Epoch 33 Loss 0.0239
Epoch 34 Loss 0.0231
Epoch 35 Loss 0.0223
Epoch 36 Loss 0.0215
Epoch 37 Loss 0.0208
Epoch 38 Loss 0.0202
Epoch 39 Loss 0.0196
Epoch 40 Loss 0.0190
Epoch 41 Loss 0.0184
Epoch 42 Loss 0.0179
Epoch 43 Loss 0.0174
Epoch 44 Loss 0.0170
Epoch 45 Loss 0.0165
Epoch 46 Loss 0.0161
Epoch 47 Loss 0.0157
Epoch 48 Loss 0.0153
Epoch 49 Loss 0.0149
Epoch 50 Loss 0.0146
Epoch 51 Loss 0.0143
Epoch 52 Loss 0.0139
Epoch 53 Loss 0.0136
Epoch 54 Loss 0.0133
Epoch 55 Loss 0.0131
Epoch 56 Loss 0.0128
Epoch 57 Loss 0.0125
Epoch 58 Loss 0.0123
Epoch 59 Loss 0.0120
Epoch 60 Loss 0.0118
Epoch 61 Loss 0.0116
```

```
Epoch 62 Loss 0.0114
Epoch 63 Loss 0.0112
Epoch 64 Loss 0.0110
Epoch 65 Loss 0.0108
Epoch 66 Loss 0.0106
Epoch 67 Loss 0.0104
Epoch 68 Loss 0.0102
Epoch 69 Loss 0.0100
Epoch 70 Loss 0.0099
Epoch 71 Loss 0.0097
Epoch 72 Loss 0.0096
Epoch 73 Loss 0.0094
Epoch 74 Loss 0.0093
Epoch 75 Loss 0.0091
Epoch 76 Loss 0.0090
Epoch 77 Loss 0.0089
Epoch 78 Loss 0.0087
Epoch 79 Loss 0.0086
Epoch 80 Loss 0.0085
Epoch 81 Loss 0.0084
Epoch 82 Loss 0.0082
Epoch 83 Loss 0.0081
Epoch 84 Loss 0.0080
Epoch 85 Loss 0.0079
Epoch 86 Loss 0.0078
Epoch 87 Loss 0.0077
Epoch 88 Loss 0.0076
Epoch 89 Loss 0.0075
Epoch 90 Loss 0.0074
Epoch 91 Loss 0.0073
Epoch 92 Loss 0.0072
Epoch 93 Loss 0.0071
Epoch 94 Loss 0.0070
Epoch 95 Loss 0.0070
Epoch 96 Loss 0.0069
Epoch 97 Loss 0.0068
Epoch 98 Loss 0.0067
Epoch 99 Loss 0.0066
Epoch 100 Loss 0.0066
Epoch 101 Loss 0.0065
Epoch 102 Loss 0.0064
Epoch 103 Loss 0.0064
Epoch 104 Loss 0.0063
Epoch 105 Loss 0.0062
Epoch 106 Loss 0.0061
Epoch 107 Loss 0.0061
Epoch 108 Loss 0.0060
Epoch 109 Loss 0.0060
Epoch 110 Loss 0.0059
Epoch 111 Loss 0.0058
Epoch 112 Loss 0.0058
Epoch 113 Loss 0.0057
Epoch 114 Loss 0.0057
Epoch 115 Loss 0.0056
Epoch 116 Loss 0.0055
Epoch 117 Loss 0.0055
Epoch 118 Loss 0.0054
Epoch 119 Loss 0.0054
Epoch 120 Loss 0.0053
Epoch 121 Loss 0.0053
Epoch 122 Loss 0.0052
```

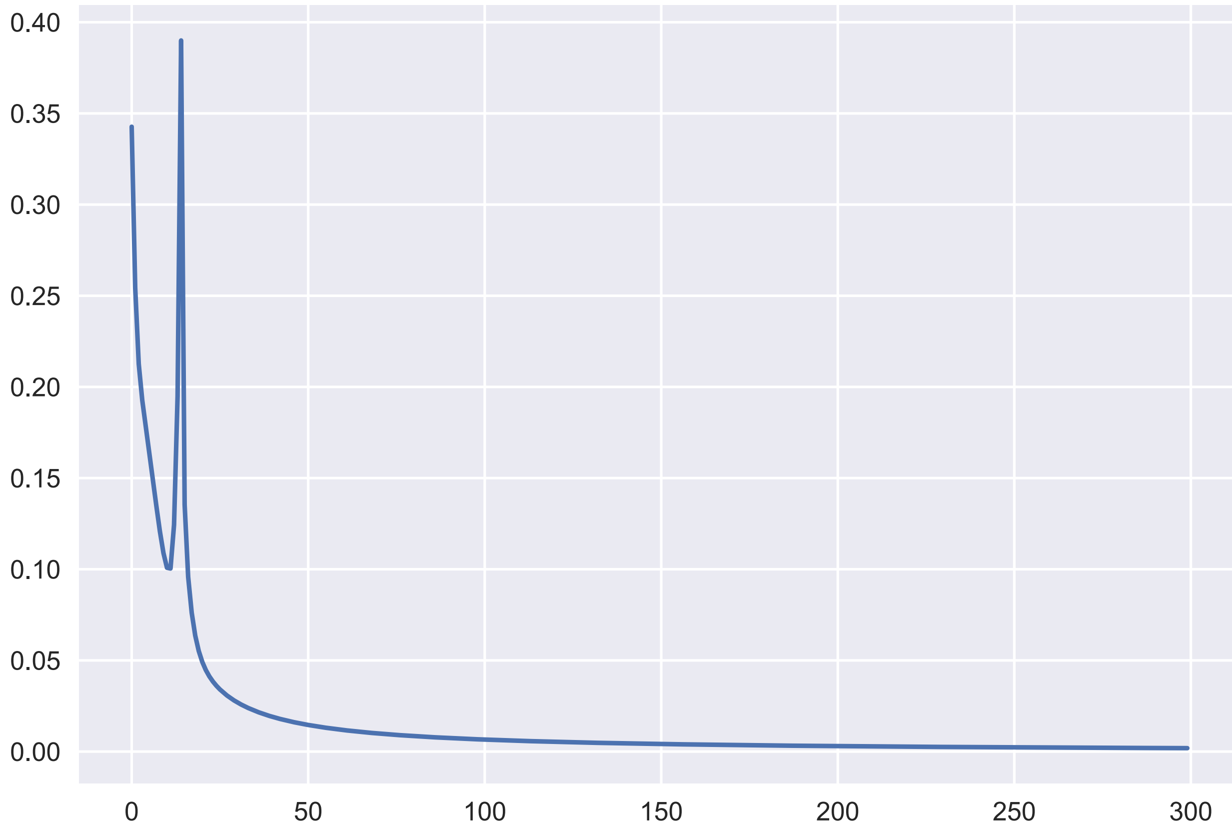
```
Epoch 123 Loss 0.0052
Epoch 124 Loss 0.0051
Epoch 125 Loss 0.0051
Epoch 126 Loss 0.0050
Epoch 127 Loss 0.0050
Epoch 128 Loss 0.0050
Epoch 129 Loss 0.0049
Epoch 130 Loss 0.0049
Epoch 131 Loss 0.0048
Epoch 132 Loss 0.0048
Epoch 133 Loss 0.0047
Epoch 134 Loss 0.0047
Epoch 135 Loss 0.0047
Epoch 136 Loss 0.0046
Epoch 137 Loss 0.0046
Epoch 138 Loss 0.0046
Epoch 139 Loss 0.0045
Epoch 140 Loss 0.0045
Epoch 141 Loss 0.0044
Epoch 142 Loss 0.0044
Epoch 143 Loss 0.0044
Epoch 144 Loss 0.0043
Epoch 145 Loss 0.0043
Epoch 146 Loss 0.0043
Epoch 147 Loss 0.0042
Epoch 148 Loss 0.0042
Epoch 149 Loss 0.0042
Epoch 150 Loss 0.0041
Epoch 151 Loss 0.0041
Epoch 152 Loss 0.0041
Epoch 153 Loss 0.0040
Epoch 154 Loss 0.0040
Epoch 155 Loss 0.0040
Epoch 156 Loss 0.0040
Epoch 157 Loss 0.0039
Epoch 158 Loss 0.0039
Epoch 159 Loss 0.0039
Epoch 160 Loss 0.0038
Epoch 161 Loss 0.0038
Epoch 162 Loss 0.0038
Epoch 163 Loss 0.0038
Epoch 164 Loss 0.0037
Epoch 165 Loss 0.0037
Epoch 166 Loss 0.0037
Epoch 167 Loss 0.0037
Epoch 168 Loss 0.0036
Epoch 169 Loss 0.0036
Epoch 170 Loss 0.0036
Epoch 171 Loss 0.0036
Epoch 172 Loss 0.0035
Epoch 173 Loss 0.0035
Epoch 174 Loss 0.0035
Epoch 175 Loss 0.0035
Epoch 176 Loss 0.0035
Epoch 177 Loss 0.0034
Epoch 178 Loss 0.0034
Epoch 179 Loss 0.0034
Epoch 180 Loss 0.0034
Epoch 181 Loss 0.0033
Epoch 182 Loss 0.0033
Epoch 183 Loss 0.0033
```

```
Epoch 184 Loss 0.0033
Epoch 185 Loss 0.0033
Epoch 186 Loss 0.0032
Epoch 187 Loss 0.0032
Epoch 188 Loss 0.0032
Epoch 189 Loss 0.0032
Epoch 190 Loss 0.0032
Epoch 191 Loss 0.0031
Epoch 192 Loss 0.0031
Epoch 193 Loss 0.0031
Epoch 194 Loss 0.0031
Epoch 195 Loss 0.0031
Epoch 196 Loss 0.0031
Epoch 197 Loss 0.0030
Epoch 198 Loss 0.0030
Epoch 199 Loss 0.0030
Epoch 200 Loss 0.0030
Epoch 201 Loss 0.0030
Epoch 202 Loss 0.0029
Epoch 203 Loss 0.0029
Epoch 204 Loss 0.0029
Epoch 205 Loss 0.0029
Epoch 206 Loss 0.0029
Epoch 207 Loss 0.0029
Epoch 208 Loss 0.0028
Epoch 209 Loss 0.0028
Epoch 210 Loss 0.0028
Epoch 211 Loss 0.0028
Epoch 212 Loss 0.0028
Epoch 213 Loss 0.0028
Epoch 214 Loss 0.0028
Epoch 215 Loss 0.0027
Epoch 216 Loss 0.0027
Epoch 217 Loss 0.0027
Epoch 218 Loss 0.0027
Epoch 219 Loss 0.0027
Epoch 220 Loss 0.0027
Epoch 221 Loss 0.0027
Epoch 222 Loss 0.0026
Epoch 223 Loss 0.0026
Epoch 224 Loss 0.0026
Epoch 225 Loss 0.0026
Epoch 226 Loss 0.0026
Epoch 227 Loss 0.0026
Epoch 228 Loss 0.0026
Epoch 229 Loss 0.0026
Epoch 230 Loss 0.0025
Epoch 231 Loss 0.0025
Epoch 232 Loss 0.0025
Epoch 233 Loss 0.0025
Epoch 234 Loss 0.0025
Epoch 235 Loss 0.0025
Epoch 236 Loss 0.0025
Epoch 237 Loss 0.0025
Epoch 238 Loss 0.0024
Epoch 239 Loss 0.0024
Epoch 240 Loss 0.0024
Epoch 241 Loss 0.0024
Epoch 242 Loss 0.0024
Epoch 243 Loss 0.0024
Epoch 244 Loss 0.0024
```

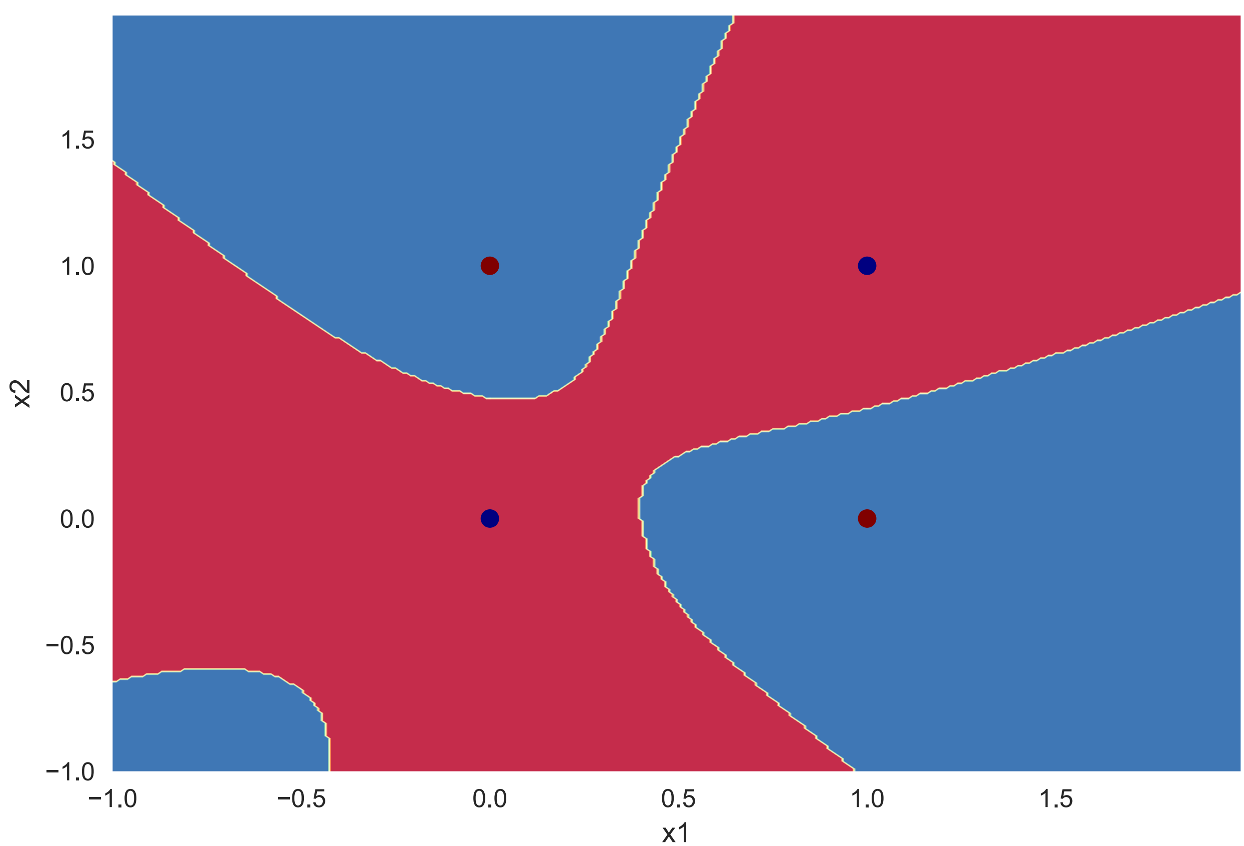
```
Epoch 245 Loss 0.0024
Epoch 246 Loss 0.0023
Epoch 247 Loss 0.0023
Epoch 248 Loss 0.0023
Epoch 249 Loss 0.0023
Epoch 250 Loss 0.0023
Epoch 251 Loss 0.0023
Epoch 252 Loss 0.0023
Epoch 253 Loss 0.0023
Epoch 254 Loss 0.0023
Epoch 255 Loss 0.0023
Epoch 256 Loss 0.0022
Epoch 257 Loss 0.0022
Epoch 258 Loss 0.0022
Epoch 259 Loss 0.0022
Epoch 260 Loss 0.0022
Epoch 261 Loss 0.0022
Epoch 262 Loss 0.0022
Epoch 263 Loss 0.0022
Epoch 264 Loss 0.0022
Epoch 265 Loss 0.0022
Epoch 266 Loss 0.0021
Epoch 267 Loss 0.0021
Epoch 268 Loss 0.0021
Epoch 269 Loss 0.0021
Epoch 270 Loss 0.0021
Epoch 271 Loss 0.0021
Epoch 272 Loss 0.0021
Epoch 273 Loss 0.0021
Epoch 274 Loss 0.0021
Epoch 275 Loss 0.0021
Epoch 276 Loss 0.0021
Epoch 277 Loss 0.0020
Epoch 278 Loss 0.0020
Epoch 279 Loss 0.0020
Epoch 280 Loss 0.0020
Epoch 281 Loss 0.0020
Epoch 282 Loss 0.0020
Epoch 283 Loss 0.0020
Epoch 284 Loss 0.0020
Epoch 285 Loss 0.0020
Epoch 286 Loss 0.0020
Epoch 287 Loss 0.0020
Epoch 288 Loss 0.0020
Epoch 289 Loss 0.0019
Epoch 290 Loss 0.0019
Epoch 291 Loss 0.0019
Epoch 292 Loss 0.0019
Epoch 293 Loss 0.0019
Epoch 294 Loss 0.0019
Epoch 295 Loss 0.0019
Epoch 296 Loss 0.0019
Epoch 297 Loss 0.0019
Epoch 298 Loss 0.0019
Epoch 299 Loss 0.0019
```

In [ ]:

```
plt.plot(losses)
plt.show()
```



In [ ]:
```
plot_decision_boundary(lambda x:model.predict(x),X,Y)
```



In [ ]:
```
from sklearn.datasets import make_moons,make_circles,make_classification
```

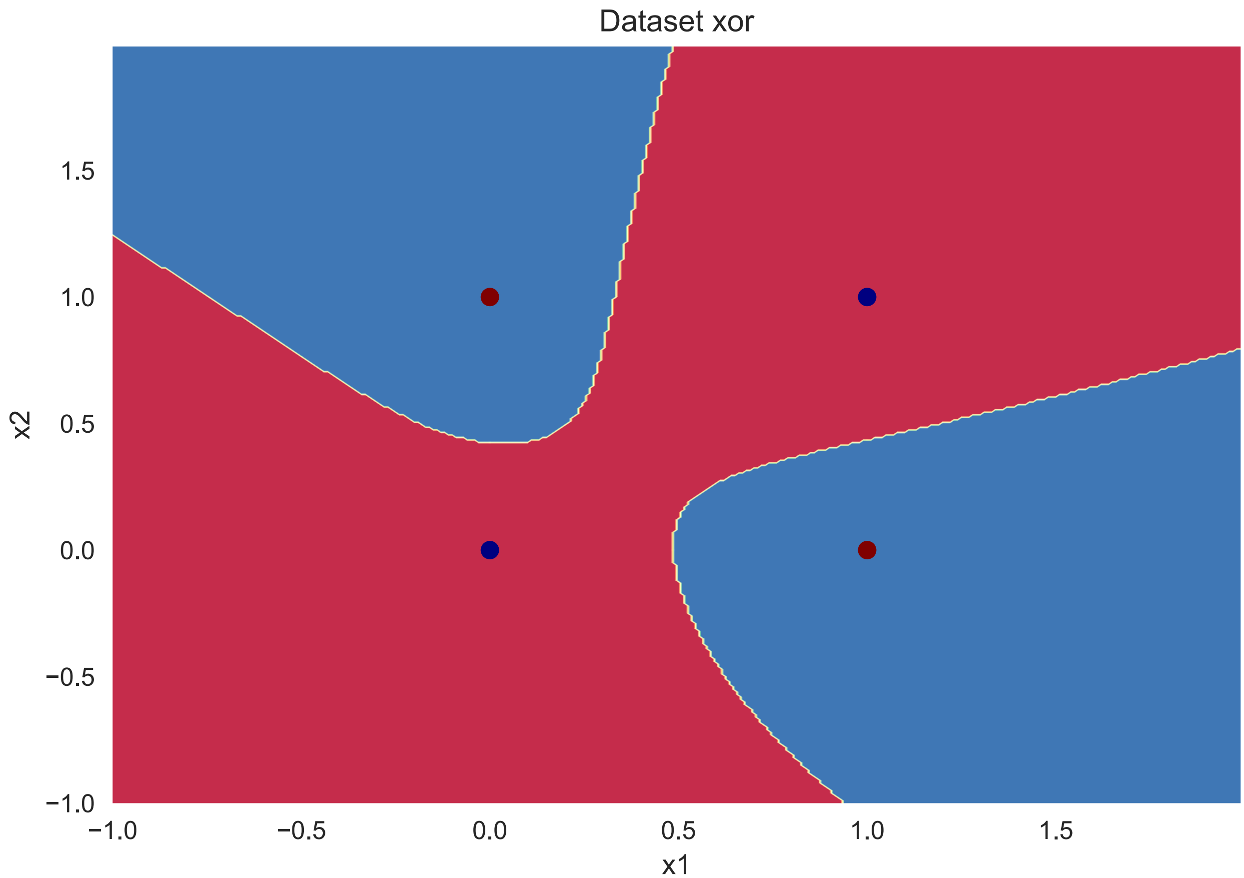
In [ ]:

```python
def load_dataset(dataset):
    if dataset=='moons':
        X,Y = make_moons(n_samples=500,noise=0.2,random_state=1) #Perceptron
    elif dataset=='circles':
        X,Y = make_circles(n_samples=500, shuffle=True, noise=0.2, random_state=1, fact
    elif dataset=='classification':
        X,Y = make_classification(n_samples=500,n_classes=2,n_features=2,n_informative=
    else:
        #Create XOR Dataset
        X = np.array([[0,0],
                    [0,1],
                    [1,0],
                    [1,1]])
        Y = np.array([0,1,1,0])

    return X,Y
```

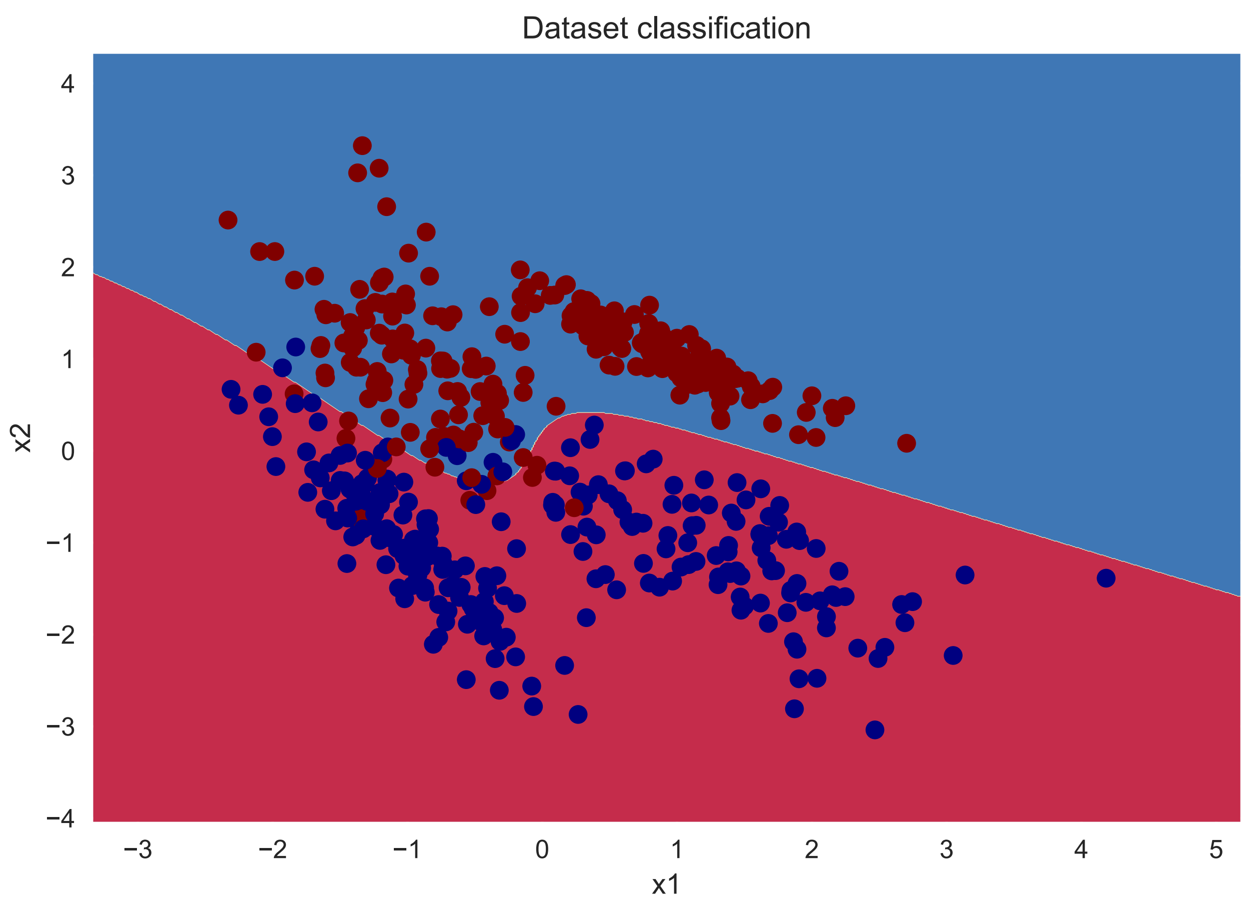
In [ ]:

```python
datasets = ["xor","classification","moons","circles"]

for d in datasets:
    model = NeuralNetwork(input_size=2,layers=[4,3],output_size=2)
    X,Y  = load_dataset(d)
    train(X,Y,model,1000,0.001,logs=False)
    outputs = model.predict(X)

    training_accuracy = np.sum(outputs==Y)/Y.shape[0]
    print("Training Acc %.4f"%training_accuracy)

    plt.title("Dataset "+d)
    plot_decision_boundary(lambda x:model.predict(x),X,Y)
    plt.show()
```

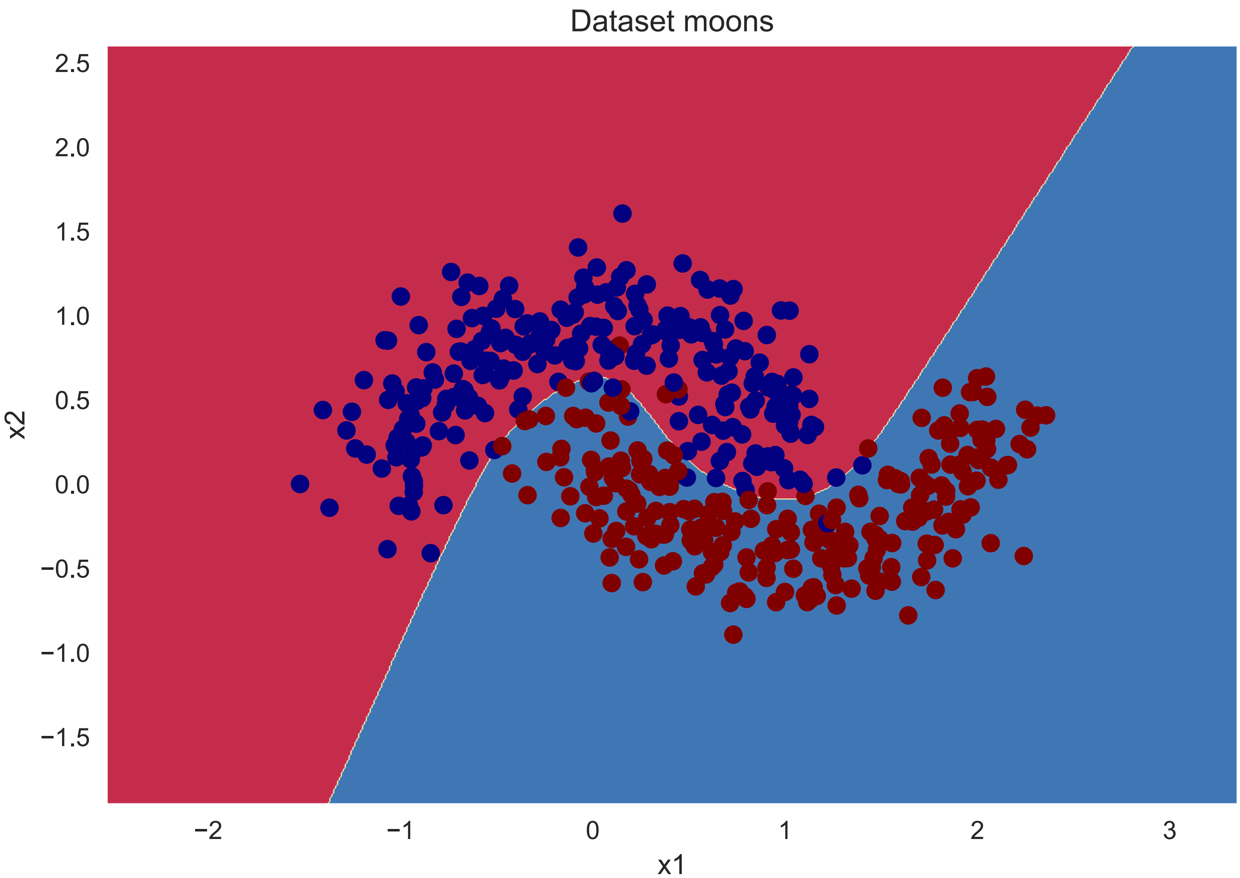
```
Training Acc 1.0000
```



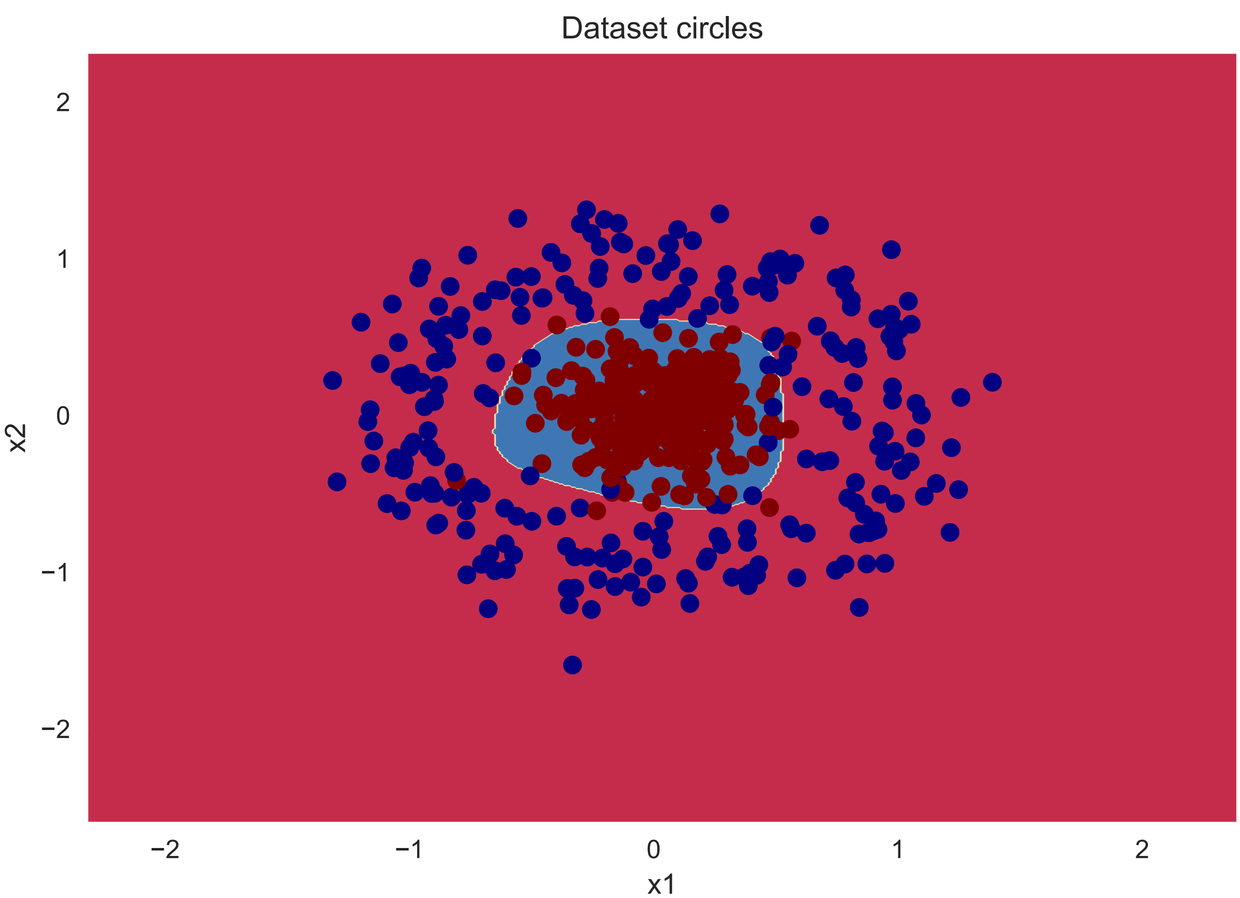

```
Training Acc 0.9600
```




```
Training Acc 0.9740
```




```
Training Acc 0.9640
```




```
In [ ]:
```