Machine Intelligence Lab

Week 1

Vanshika Goel	PES1UG20CS484	Section H	Roll No 40

PES1UG20CS484.py

Q1) create_numpy_ones_array(shape)

- **Input : tuple (x,y)**
- **Output: numpy array of the shape (x,y) with 1 at all position**

A1)

```
#input: tuple (x,y)    x,y:int
def create_numpy_ones_array(shape):
    #return a numpy array with one at all index
        array = np.ones([int(shape[0]), int(shape[1])], dtype = int)
        return array
```

Q2)

create_numpy_zeros_array(shape)

- **Input : tuple (x,y)**
- **Output: numpy array of the shape (x,y) with 0 at all position**

A2)

```
#input: tuple (x,y)    x,y:int

def create_numpy_zeros_array(shape):

#return a numpy array with zeros at all index

array = np.zeros([int(shape[0]), int(shape[1])], dtype = int)

 return array
```

Q3)

create_identity_numpy_array(order)

- **Input : int**
- **Output: Identity matrix in the form of numpy array of dimension order x order**

A3)

```
#input: int
def create_identity_numpy_array(order):
#return a identity numpy array of the defined order
 array=None
array = np.identity(order, dtype = int)
 return array
```

Q4)

matrix_cofactor(array)

- **Input: numpy array**
- **Output: cofactor matrix of the input in the form of numpy array**

A4)

```
#input: numpy array
def matrix_cofactor(array):
#return cofactor matrix of the given array
    cofactor = np.linalg.inv(array).T * (np.linalg.det(array))
    return array
```

Q5)

f1(X1,coef1,X2,coef2,seed1,seed2,seed3,shape1,shape2)

- **Input: (numpy array, int ,numpy array, int , int , int , tuple,tuple)**
- **Perform W1 x (X1 ** coef1) + W2 x (X2 ** coef2) +b**
- **where W1 is random matrix of shape shape1 with seed1**
- **where W2 is random matrix of shape shape2 with seed2**
- **if dimension mismatch occur return -1**
- **Output: computed function(numpy array) or -1**

A5)

```
#Input: (numpy array, int ,numpy array, int , int , int , tuple,tuple)
#tuple (x,y)    x,y:int
def f1(X1,coef1,X2,coef2,seed1,seed2,seed3,shape1,shape2):
   #note: shape is of the forst (x1,x2)
   np.random.seed(seed1)
   W1 = np.random.randn(*shape1)
   np.random.seed(seed2)
   W2 = np.random.randn(*shape2)
   powertuple1= np.power(X1,coef1)
   powertuple2= np.power(X2,coef2)
   multuple1 = np.matmul(W1,powertuple1)
   multuple2 = np.matmul(W2,powertuple2)
   res = np.add(multuple1,multuple2)
   shape3=res.shape
   np.random.seed(seed3)
   b=np.random.randn(*shape3)
```

```
    ans = np.add(res,b)
    if (shape1==shape2):
        return ans
    else:
        return -1
```

Q6)

fill_with_mode(filename, column)

- **Input: (str, str)**
- **Fill the missing values(NaN) in a column with the mode of that column**
- **output: df: Pandas DataFrame object.(Representing entire data and where 'column' does not contain NaN values)**

A6)

```
def fill_with_mode(filename, column):
    df=pd.read_csv(filename)
    df[column] = df[column].fillna(df[column].mode()[0])
    return df
```

Q7)

fill_with_group_average(df, group, column)

- **Input: (DataFrame,str, str)**
- **Fill the missing values(NaN) in 'column' with the mean value of the group the row belongs to.**
- **output: df: Pandas DataFrame object.(Representing entire data and where 'column' does not contain NaN values)**

A7)

```
def fill_with_group_average(df, group, column):
    df[column] = df.groupby(group)[column].transform(lambda x: x.fillna(x.mean()))
    return df
```

Q8)

get_rows_greater_than_avg(df, column)

- **Input: (DataFrame, str)**
- **Return all the rows(with all columns) where the value in a certain 'column' is greater than the average value of that column.**
- **output: df: Pandas DataFrame object.**

A8)

```
def get_rows_greater_than_avg(df, column):
    mask = df[column] > df[column].mean()
    df=df[mask]
    return df
```