MACHINE LEARNING WORKSHEET

1.(D)

2.(B)

3.(B)

4.(C)

5.(A)

6.(B)

7.(A)

8.(B)

9.A,C,D

10.A,C

11.C,D

12.Which Linear Regression training algorithm can we use if we have a training set with millions of features?

If you have a training set with millions of features you can use Stochastic Gradient Descent or Mini-batch Gradient Descent, and perhaps Batch Gradient Descent if the training set fits in memory. But you cannot use the Normal Equation because the computational complexity grows quickly (more than quadratically) with the number of features.

13.Which algorithms will not suffer or might suffer, if the features in training set have very different scales?

### If the features in your training set have very different scales, the cost function will have the shape of an elongated bowl, so the Gradient Descent algorithms will take a long time to converge. To solve this you should scale the data before training the model. Note that the Normal Equation will work just fine without scaling.

PYTHON WORKSHEET

1.(C)%

2.(A)0.666

3.(C)24

4.(A)2

5.(D)6

6.(D)

7.(A)

8.(C)

9.A,B,C

10.A,B

12. Write a python program to find whether a number is prime or composite.

num = int(input("Enter a number: "))

**if** num > 1:

**for** i **in** range(2,num):

**if** (num % i) == 0:

**print**(num,"is a composite number")

**print**(i,"times",num//i,"is",num)

**break**

**else**:

**print**(num,"is a prime number")

**else**:

**print**(num,"is a composite number")

13. Write a python program to check whether a given string is palindrome or not.

def isPalindrome(s):

    return s == s[::-1]

s = "malayalam"

ans = isPalindrome(s)

if ans:

    print("Yes")

else:

    print("No")

14. Write a Python program to get the third side of right-angled triangle from two given sides.

def pythagoras(opposite\_side,adjacent\_side,hypotenuse):

if opposite\_side == str("x"):

return ("Opposite = " + str(((hypotenuse\*\*2) - (adjacent\_side\*\*2))\*\*0.5))

elif adjacent\_side == str("x"):

return ("Adjacent = " + str(((hypotenuse\*\*2) - (opposite\_side\*\*2))\*\*0.5))

elif hypotenuse == str("x"):

return ("Hypotenuse = " + str(((opposite\_side\*\*2) + (adjacent\_side\*\*2))\*\*0.5))

else:

return "You know the answer!"

print(pythagoras(3,4,'x'))

print(pythagoras(3,'x',5))

print(pythagoras('x',4,5))

print(pythagoras(3,4,5))