Aman 2019014

Q1: Data preprocessing: I have used one hot encoding on a categorical column and have removed all the rows with NA values. Also dropped the “No” column from the dataset.

a.

Model Training

Using Gini Index

```
from sklearn.tree import DecisionTreeClassifier
dtclf_gini = DecisionTreeClassifier(criterion='gini')
dtclf_gini.fit(X_train,y_train)
y_pred_gini = dtclf_gini.predict(X_test)
dtclf_gini.score(X_test, y_test)
```

```
0.873358348968105
```

Using Entropy

```
[ ] from sklearn.tree import DecisionTreeClassifier
    dtclf_ent = DecisionTreeClassifier(criterion='entropy')
    dtclf_ent.fit(X_train,y_train)
    y_predict_gini = dtclf_ent.predict(X_test)
    dtclf_ent.score(X_test, y_test)
```

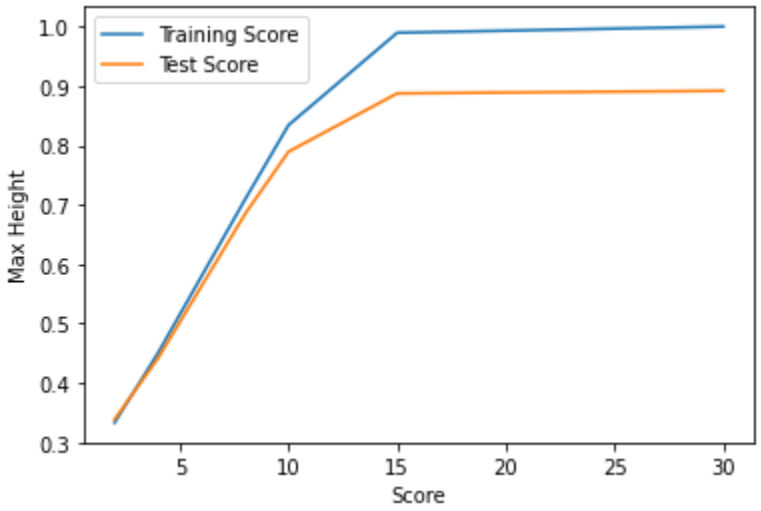
```
0.8958724202626641
```

Acti

Score on entropy came out better than gini

b.

Training trees on different max depth:




```
([0.3383364602876798,
  0.44152595372107567,
  0.6841776110068793,
  0.7892432770481551,
  0.8874296435272045,
  0.8918073796122576],
 [0.3329536317341195,
  0.45034843205574915,
  0.7087242026266416,
  0.8340257303671937,
  0.9895470383275261,
  1.0])
```

Best accuracy came out at max **depth of 30**

c.

Ensembling:

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test, y_pred_ens)
```

```
0.39305816135084426
```

Performance of the model has reduced compared to 1a and 2b. Max depth of trees we have given is 3, and number of features are comparatively greater hence our model is not learning properly.

D.

```
max_heights = [4,8,10,15,20,30]
number_trees = [10,20,30,50,100,200]
```

```
[[0.4530956848030019,
  0.7529706066291432,
  0.858036272670419,
  0.926829268292683,
  0.9312070043777361,
  0.9343339587242027],--> No. of tree=10 and Max depth=30
 [0.45997498436522827,
  0.7451532207629769,
  0.8605378361475923,
  0.9452782989368356,
  0.9393370856785491,
  0.9477798624140088],--> No. of tree=20 and Max depth=30
 [0.4462163852407755,
  0.743277048155097,
  0.8702313946216386,
  0.9424640400250156,
  0.9471544715447154,
  0.9455909943714822],-->No. of tree=30 and Max depth=30
 [0.4462163852407755,
  0.756722951844903,
  0.873358348968105,
  0.9474671669793621,
  0.9505941213258287,--> Best accuracy
  0.948405253283302],--> No. of tree=50 and Max depth=30
 [0.4462163852407755,
  0.7589118198874296,
  0.8708567854909318,
```

```
  0.9534083802376485,
  0.9515322076297686,
  0.9515322076297686],
 [0.4509068167604753,
  0.7636022514071295,
  0.8764853033145716,
  0.9505941213258287,
  0.9530956848030019,
  0.9512195121951219]]
```

e.
Adaboost:

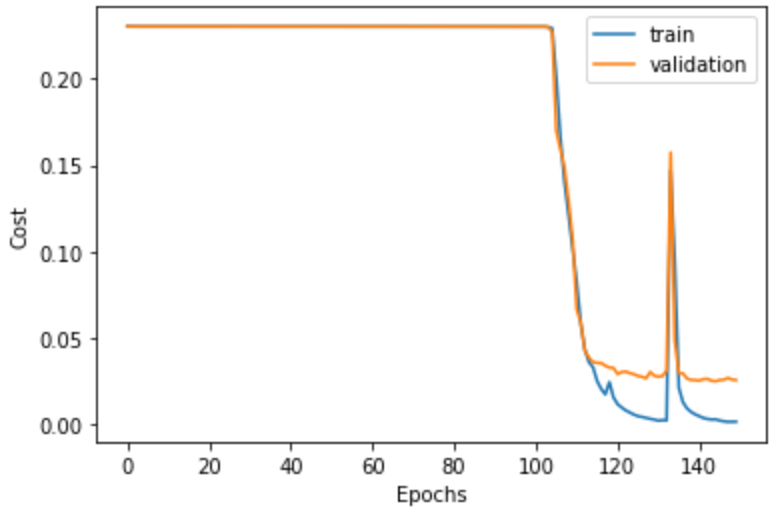
[0.8879131406674118,
0.9041992655277024,
0.9247964234392464,
0.937250518920645,
0.9457129171323647]

Both random forest and adaboost have given almost the same accuracy but adaboost performed quite faster than random forest

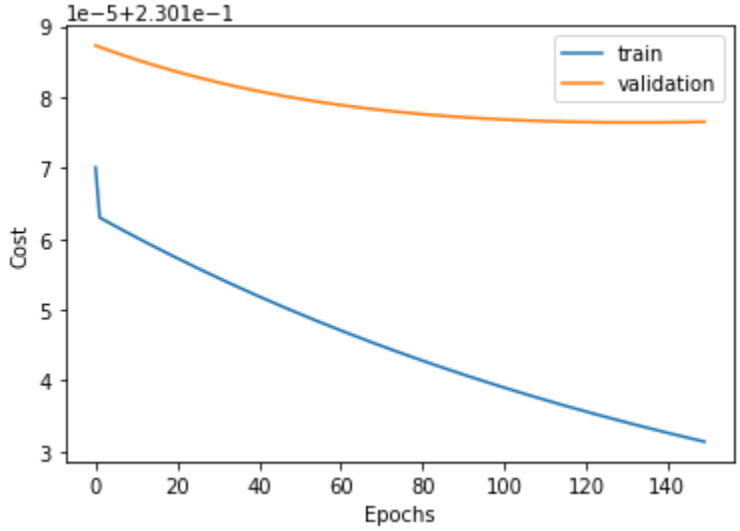
Q2:

Test accuracy:
Relu score: 0.9637142857142857
Linear score: 0.9092857142857143
Sigmoid score:0.11357142857142857
tanh score:0.967142857142857
LeakyRelu score:0.959
Soft max:0.9664285714285714

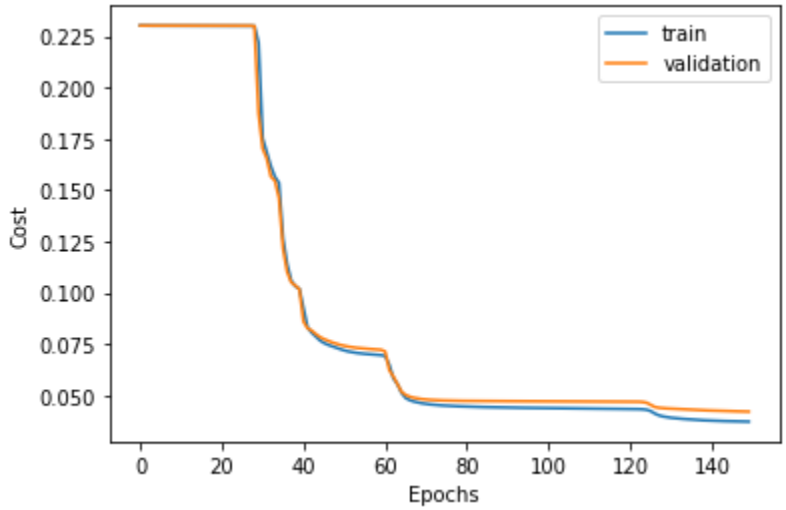
Relu:



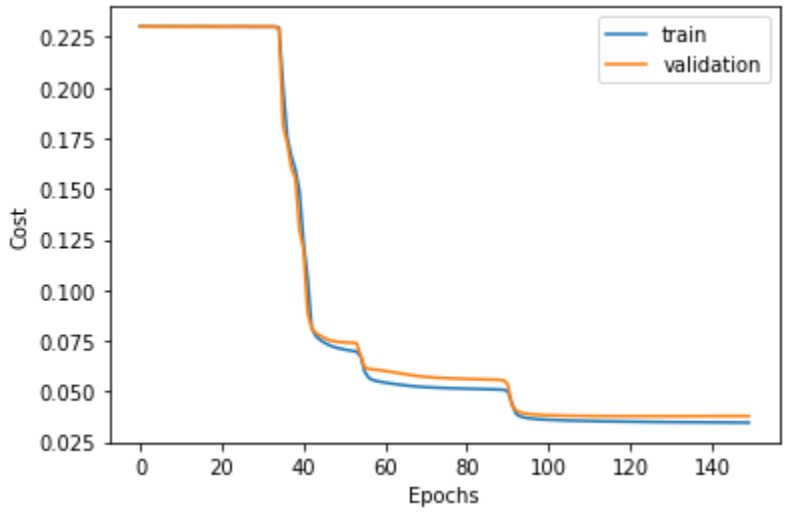
Sigmoid



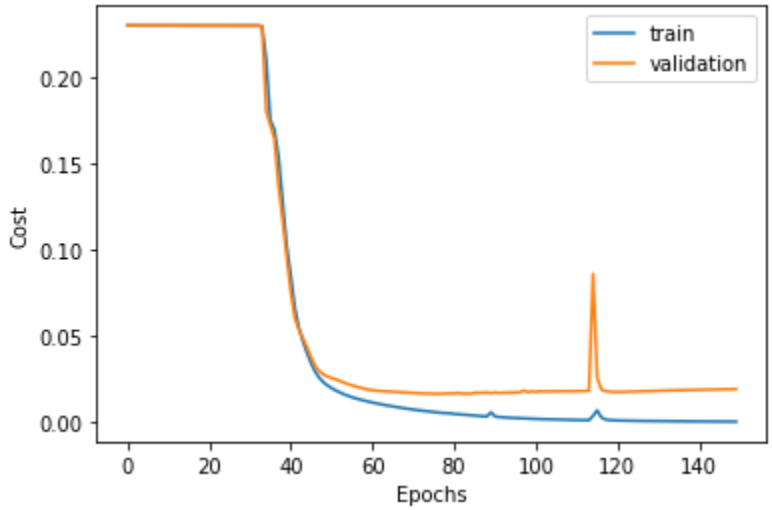
Softmax:



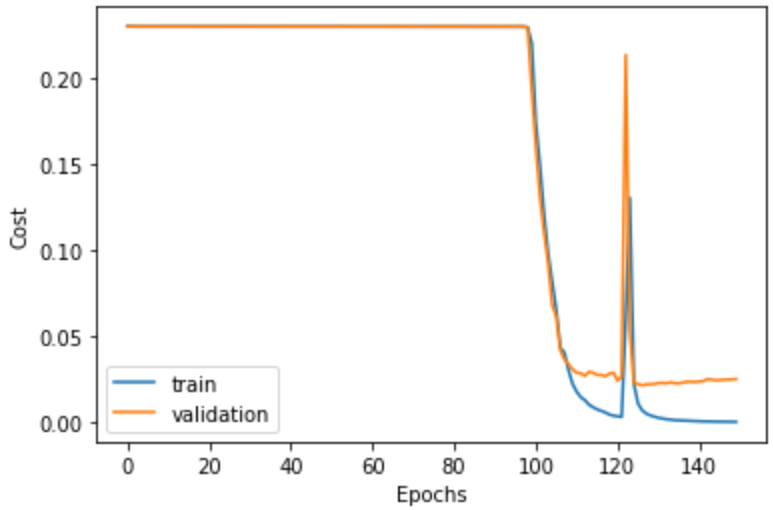
Linear:



tanh :



Leakyrelu



2.3: We use the softmax activation function at the output layer. The reason for using this function is that the softmax activation function forces the values of output neurons to be between zero and one, so they can represent probability scores for multiclass classification.

2.4:

```
    print("Test accuracy: ", model_sklearn.score(x_test, y_test))

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:934: DataConversionWarning: A
  y = column_or_1d(y, warn=True)
Activation Function:  relu  :
Train accuracy:  1.0
Val accuracy:  0.9814285714285714
Test accuracy:  0.987
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:934: DataConversionWarning: A
  y = column_or_1d(y, warn=True)
Activation Function:  tanh  :
Train accuracy:  1.0
Val accuracy:  0.9800714285714286
Test accuracy:  0.9831428571428571
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:934: DataConversionWarning: A
  y = column_or_1d(y, warn=True)
Activation Function:  logistic  :
Train accuracy:  0.9999795918367347
Val accuracy:  0.9735714285714285
Test accuracy:  0.9771428571428571
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:934: DataConversionWarning: A
  y = column_or_1d(y, warn=True)
Activation Function:  identity  :
Train accuracy:  0.9207551020408163
Val accuracy:  0.9033571428571429
Test accuracy:  0.9178571428571428
```

Activation Function: relu :
`Test accuracy: 0.9862857142857143`

Activation Function: tanh
`Test accuracy: 0.9845714285714285`

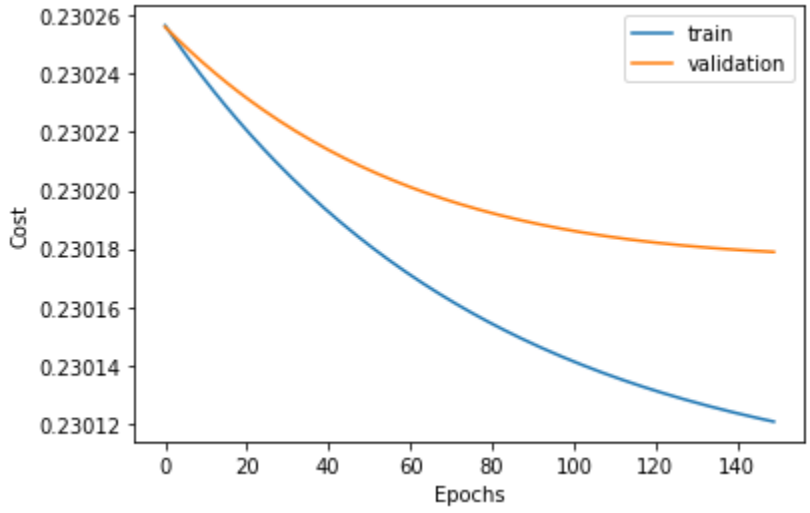
Activation Function: logistic :
`Test accuracy: 0.973`

Activation Function: identity :
`Test accuracy: 0.10128571428571428`

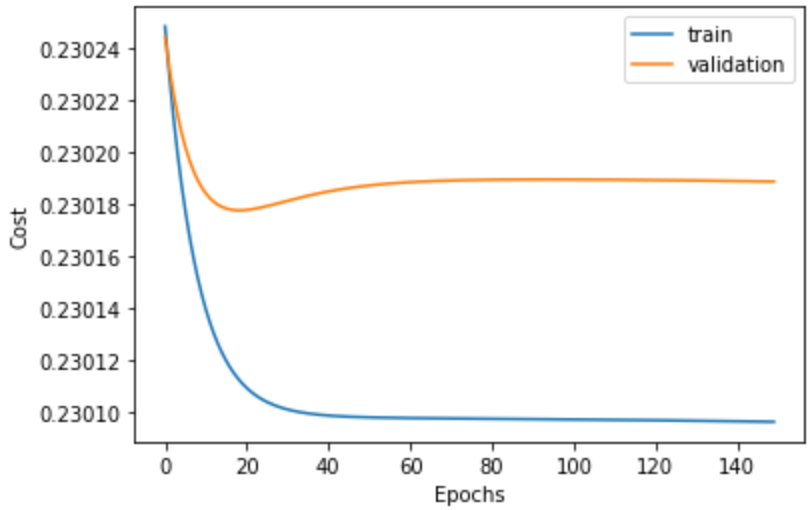
Except logistic/sigmoid and linear all other accuracies are coming out to be nearly the same. Accuracy of sigmoid using sklearn MLP got higher than the MLP I implemented and in case of linear MyNeuralNetwork gave better accuracy than sklearn MLP.

2.5: Learning rates 0.001 and 0.01 are too small to converge as even after 150 epochs our training loss did not reduce much. Learning rate 0.1 and 1 converged very well and gave a very good accuracy but learning rate 1 came out the best among all.

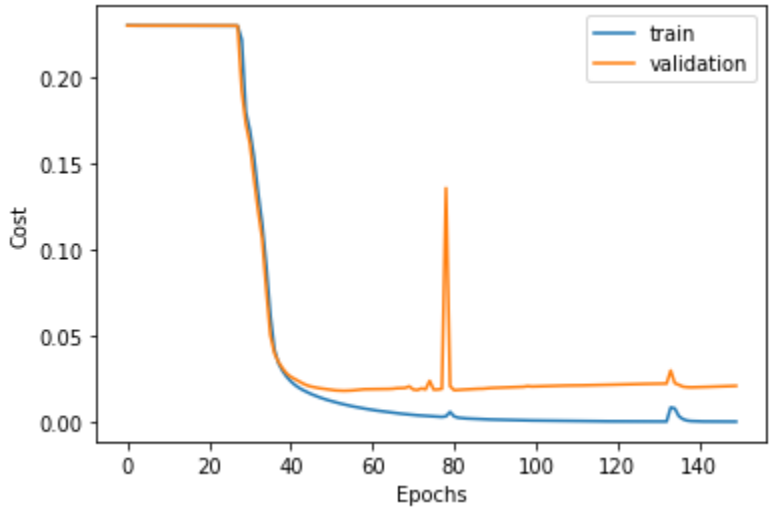
Learning rate = 0.001:



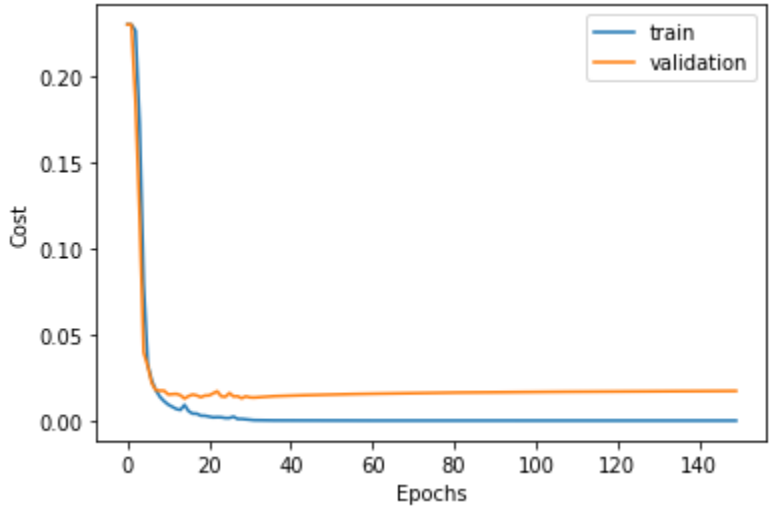
Learning rate = 0.01



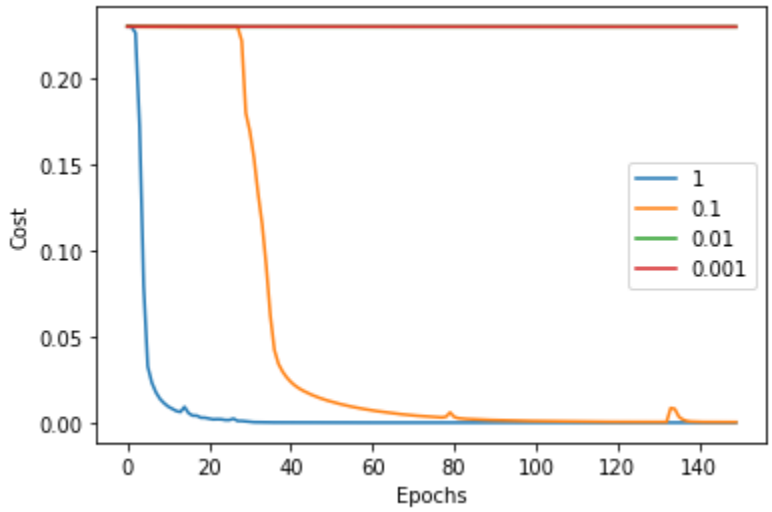
Learning rate = 0.1



Learning rate = 1



Legend:

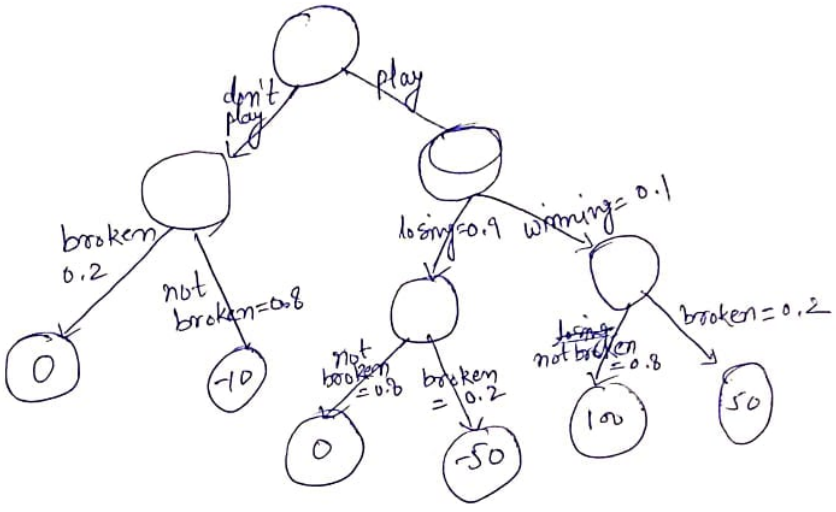

0.20
0.15
0.10
0.05
0.00
Cost
0
20
40
60
80
100
120
140
Epochs
1
0.1
0.01
0.001

Theory

Theory

Q1:

1. Decision tree diagram



2. Expected utility of not playing = $0.2 \times (-10) + 0.8 \times 0 = -2$

Expected utility of playing = $0.8 \times 0.9 \times 0 + 0.9 \times 0.2 \times (-50) + 0.1 \times 0.8 \times 100 + 0.1 \times 0.2 \times 50$

$= 0$

As the expected utility of playing is more than not playing, hence player should.

Expected utility if player plays and wins =

$0.9 \times 0.8 \times 0 + 0.1 \times 0.8 \times 100 + 0.1 \times 0.2 \times 50 = 9$

Expected utility if player plays and losses =

$0.9 \times 0.8 \times 0 + 0.9 \times 0.2 \times (-50) = -9$

Player losses or wins with equal possibility

3.

Expected utility of player's leg is broken and not playing

$= -10$

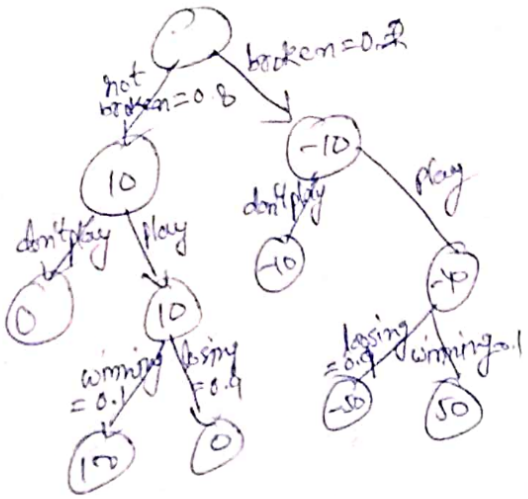
Expected utility of leg broken and play $= 0.9 \times (-50) + 0.1 \times 50$

$= -40$

Expected utility of leg broken

$= \max(-10, -40) = -10$

Decision tree based on perfect information of ankle



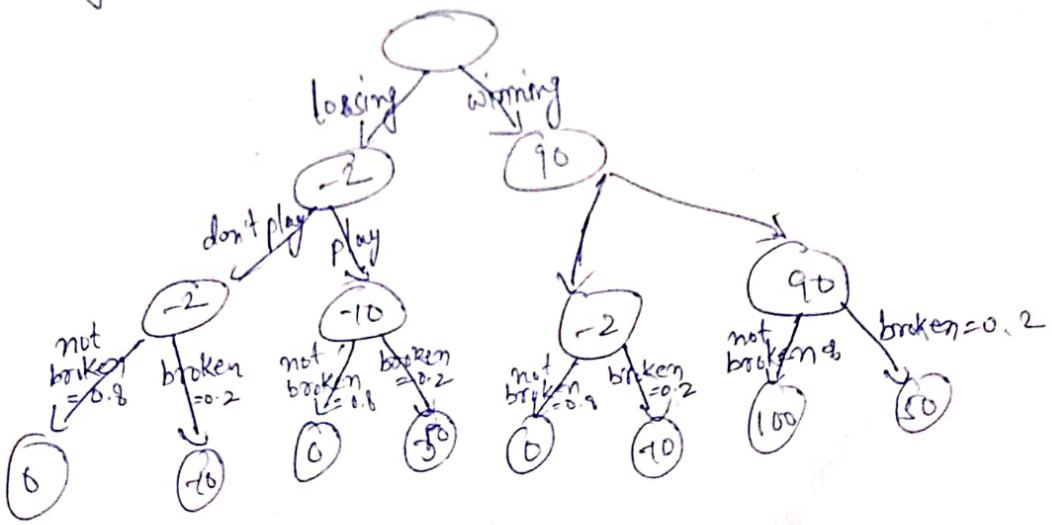
Expected utility of leg not broken and not playing $= 0$

Expected utility of leg not broken and playing $= 0.1 \times 100 + 0.9 \times 0$

$= 10$

4.

Expected utility of perfect information about the leg $= 0.8 \times 10 + 0.2 \times (-10) = 6$



Expected value of perfect information of winning =

$= 0.1 \times 90 + 0.9 \times (-2)$

$= 7.2$

S: Yes, it is possible to use a decision tree in the given case. We can make decision tree of probabilities and by putting the probabilities of winning and losing above broken leg branch.

Ans 3 Let $x = \beta_1 x_1 + \beta_2 x_2$

and output $y = \text{argmax}_y P(Y=y|x)$

So, there are two cases possible

$P(Y=1|x) > P(Y=-1|x)$ then $Y = P(Y=1|x)$

$P(Y=1|x) < P(Y=-1|x)$ then $Y = P(Y=-1|x)$

for the 1st case, when

$$P(Y=1|x) > P(Y=-1|x)$$

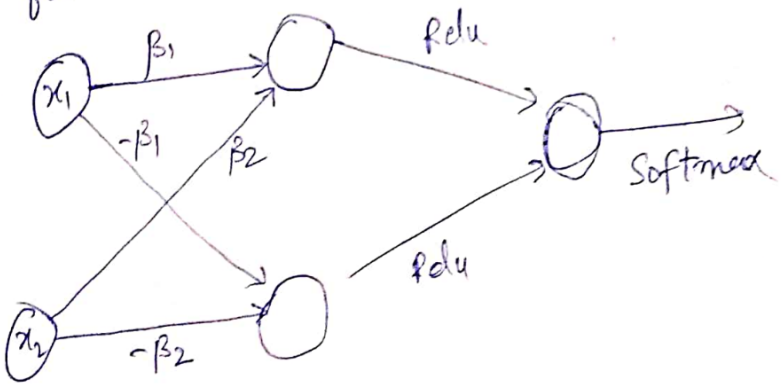
$$\Rightarrow \frac{e^x}{1+e^x} > \frac{1}{1+e^x} \Rightarrow e^x > 1$$

Solving the above we get

$$x > 0 \Rightarrow \beta_1 x_1 + \beta_2 x_2 > 0$$

hence we have to predict $\frac{\exp(\beta_1 x_1 + \beta_2 x_2)}{1+\exp(\beta_1 x_1 + \beta_2 x_2)}$

Diag for NN



References:

https://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/
https://www.nbshare.io/notebook/751082217/Activation-Functions-In-Python/
https://www.quora.com/Artificial-Neural-Networks-Why-do-we-use-softmax-function-for-output-layer