>. TechLabs Final Exam

Please enter your full name here:

This exam will test your knowledge in Artificial Intelligence.

We will test the following:

- Logistic Regressions
- Neural Networks
- Monte Carlo Simulation

Question 1:

Below is the code to import a standard dataset with breast cancer. When you run the cell, you will have it stored as the object "data", and the description of the dataset is printed out for you.

Your task is to create two models for classifying the diagnosis, and compare the accuracy metrics for the two:

1. a) Logistic Regression:

"sklearn.linear_model" offers good logit classifiers. Because the dataset is fairly small (~500 rows), we suggest that you use the "liblinear" solver for fitting the logit classifier.

Split the dataset into a test and training (30% test / 70% training) dataset and train the model on the training dataset. Then, classify the test dataset, and compute the accuracy, which you can print as an output.

P.S.: We do not expect you to regularize for type 1 or type 2 errors. Your goal is to build models that classify well the diagnosis. Also, we do not expect you to spend hours on feature engineering and/or other things that may or may not improve testing accuracy. A good answer shall just contain a sensible model and the testing accuracy.

In [1]:
```
import numpy as np
import pandas as pd

from sklearn import datasets
data = datasets.load_breast_cancer()
print(np.shape(data.data))
```

```
(569, 30)
```

In [2]:
```python
print(data.DESCR)
```

```
Breast Cancer Wisconsin (Diagnostic) Database
=============================================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
```

In [3]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from numpy import set_printoptions
set_printoptions(precision=3)

# Extracting data in array format and storing it in X
X = data.data
# The categories(Malignant=0,Benign=1) are stored in y
y = data.target
# The array is converted to a dataframe for initial examination
df = pd.DataFrame(X,columns=data.feature_names)

# The Logisitc Regression model is called
logreg = LogisticRegression(solver="liblinear")
# The dataset is split into train and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3, random_st
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

# Calculating accuracy
result = logreg.score(X_test, y_test)
print("Accuracy {}".format(round(result*100,3)))
```

```
Accuracy 96.491
```

Print the logit accuracy below:

In [4]:
```
logitacc = "[96.491]"
print("\ntesting set accuracy with logit is {}".format(logitacc))
```

```
testing set accuracy with logit is [96.491]
```

1. b) Feedforward Neural Network:

So far, so good. Let's see how a feedforward neural network does for the same task.

Take the same train/test split as in a), and construct the following neural network, which you fit to the data:

In [5]:
```
import pickle
pickle.loads(b'\x80\x03]q\x00(X\x15\x00\x00\x00Model: "sequential_2"q\x01XA\x00\x
```

Out[5]:
```
['Model: "sequential_2"',
 '_________________________________________________________________',
 'Layer (type)                 Output Shape              Param #   ',
 '=================================================================',
 'dense_5 (Dense)              (None, 30)                930       ',
 '_________________________________________________________________',
 'dense_6 (Dense)              (None, 50)                1550      ',
 '_________________________________________________________________',
 'dense_7 (Dense)              (None, 30)                1530      ',
 '_________________________________________________________________',
 'dense_8 (Dense)              (None, 1)                 31        ',
 '=================================================================',
 'Total params: 4,041',
 'Trainable params: 4,041',
 'Non-trainable params: 0',
 '_________________________________________________________________']
```

In [6]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from tensorflow import keras
from tensorflow.keras import backend
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

```
C:\Users\ajnai\Anaconda3\envs\newenvt\lib\site-packages\h5py\__init__.py:36: Fu
tureWarning: Conversion of the second argument of issubdtype from `float` to `n
p.floating` is deprecated. In future, it will be treated as `np.float64 == np.d
type(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [8]:
```
model = Sequential()
model.add(Dense(30, init='uniform', input_dim=30, activation='relu'))
model.add(Dense(50, init='uniform', activation='relu'))
model.add(Dense(30, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 30)                930       
_________________________________________________________________
dense_6 (Dense)              (None, 50)                1550      
_________________________________________________________________
dense_7 (Dense)              (None, 30)                1530      
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 31        
=================================================================
Total params: 4,041
Trainable params: 4,041
Non-trainable params: 0
_________________________________________________________________


C:\Users\ajnai\Anaconda3\envs\newenvt\lib\site-packages\ipykernel_launcher.py:
2: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(30, input_d
im=30, kernel_initializer="uniform", activation="relu")`

C:\Users\ajnai\Anaconda3\envs\newenvt\lib\site-packages\ipykernel_launcher.py:
3: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(50, kernel_
initializer="uniform", activation="relu")`
  This is separate from the ipykernel package so we can avoid doing imports unt
il
C:\Users\ajnai\Anaconda3\envs\newenvt\lib\site-packages\ipykernel_launcher.py:
4: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(30, kernel_
initializer="uniform", activation="relu")`
  after removing the cwd from sys.path.
C:\Users\ajnai\Anaconda3\envs\newenvt\lib\site-packages\ipykernel_launcher.py:
5: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(1, kernel_i
nitializer="uniform", activation="sigmoid")`
  """
```

In [9]:
```
# Compiling the ANN
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [10]:
```
model.fit(X_train, y_train, batch_size=100, nb_epoch=150)
```

```
C:\Users\ajnai\Anaconda3\envs\newenvt\lib\site-packages\ipykernel_launcher.p
y:1: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  """Entry point for launching an IPython kernel.
```

In [11]:
```
# Predicting the Test set results
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
```

In [12]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [13]:
```
print("Accuracy {}".format(round(((cm[0][0] + cm[1][1])/171)*100,3)))
```

```
Accuracy 98.246
```

You can use the library keras to contruct the ANN. Since we are dealing with a classification problem, we recommend binary_crossentropy as a loss function.

Choose a reasonable batch size and epoch count and train the model.

Here again, print the testing dataset accuracy:

In [64]:
```
'''
this looks like a nice cell to train models
'''
```

Out[64]:
```
'\nthis looks like a nice cell to train models\n'
```

In [14]:
```
ANNacc = "[98.246]"
print("\ntesting set accuracy with logit is {}".format(ANNacc))
```

```
testing set accuracy with logit is [98.246]
```

1. c) Compare the performance of the neural net with the logit. Which one seems better at the job and why might that be?

Please also comment on which model you would choose to implement in a case like this and why.

Accuracy Obtained from logistic regression: 96.491

Accuracy Obtained from Artifical neural net: 98.246

This shows that the Artificial neural net does a better job at classifying unknown data into malignant/benign types

Question 2:

Your coding skills have gained you a job as an options trader at a successful hedge fund! Congratulations!!

At the first day, your boss comes to you and asks you, whether he should buy a *call option** with a certain set of characteristics for 1€.

```
*A call option gives you the right (but not obligation) to buy a share fo
r a certain strike price. In other words, if the stock price is higher th
an the strike price, you get the difference, otherwise, you get 0:

callpayoff = max(stockprice - strikeprice, 0)
```

To price the option, you shall build a monte-carlo simulator which generatates *1 000 000* random walks, each representing the stock price in one year, which is when the option can be expired. By taking the average of these payouts, you will get the expected payout at expiry!

Luckily, your boss has also given you the characteristics and hints for how the stock price moves:

The stock price follows a student T distribution with 3 degrees of freedom (the rvs function within the t class from scipy.stats package is a great tool for creating random walks with this distribution: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.t.html) ctrl+f "rvs")

- The stock price today: 69
- Stock returns follow student T distribution with 3 degrees of freedom
- drift of 10% per year
- volatility of 20%
- strike price of the option is 96

Build a function that returns the price of the call option, and shows your boss what payout he can expect from the option in order to explain him, whether he should buy it or not.

```
P.S.: If you run into trouble on your way, you can always ask for tips fr
om your boss and since he is a nice guy, he will also give a good grade f
or all attempts!
```

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt

class Configuration:
     def __init__(self, NumberOfScenarios, NumberOfTimesteps):
         self.NumberOfScenarios=NumberOfScenarios
         self.NumberOfTimesteps = NumberOfTimesteps

class OptionTrade:
  def __init__(self, stock_price, strike_price, risk_free_rate,    volatility, ti
        self.stock_price=stock_price
        self.strike_price=strike_price
        self.risk_free_rate=risk_free_rate
        self.volatility=volatility
        self.time_to_maturity = time_to_maturity

class GBMModel:
    def __init__(self, Configuration):
        self.Configuration = Configuration

    #simulate risk factors using GBM stochastic differential equation
    def SimulateRiskFactor(self, trade):
        prices = []
        # for this example, we only are concerned with one time step as it’s an E
        timestep = 1
        for scenarioNumber in range(self.Configuration.NumberOfScenarios):
            normal_random_number = np.random.normal(0, 1)
            drift = (trade.risk_free_rate-0.5*(trade.volatility**2)) *timestep
            uncertainty =trade.volatility*np.sqrt(timestep)*normal_random_number
            price = trade.stock_price * np.exp(drift+uncertainty)
            prices.append(price)
        return prices

class OptionTradePayoffPricer:
    def CalculatePrice(self, trade, prices_per_scenario):
        pay_offs = 0
        total_scenarios = len(prices_per_scenario)
        for i in range(total_scenarios):
            price = prices_per_scenario[i]
            pay_off = price - trade.strike_price
            if(pay_off>0):
                pay_offs=pay_offs+pay_off

        discounted_price = (np.exp(-1.0*trade.risk_free_rate * trade.time_to_matu
        result = discounted_price/total_scenarios
        return result

class MonteCarloEngineSimulator:

    #instatiate with configuration and the model
    def __init__(self, configuration, model):
        self.configuration = configuration
        self.model = model

    #simulate trade and calculate price
    def Simulate(self, trade, tradePricer):
        prices_per_scenario = self.model.SimulateRiskFactor(trade)
        price = tradePricer.CalculatePrice(trade, prices_per_scenario)

        return price

def Main():
    #prepare the data
    configuration = Configuration(1000000, 1) # config
    trade = OptionTrade(69, 96, 0.1, 0.2,1) # trade
    model = GBMModel(configuration)
    tradePricer = OptionTradePayoffPricer()
    simulator = MonteCarloEngineSimulator(configuration, model)

    #simulate price
    price = simulator.Simulate(trade, tradePricer)
    print(round(price,3))

Main()
```

```
0.958
```

In [16]:

```
#This can also be calculated using Black-Scholes formula
import numpy as np
import scipy.stats as si

def call_price(S, K, T, r, sigma):

    #S: spot price
    #K: strike price
    #T: time to maturity
    #r: interest rate
    #sigma: volatility of underlying asset

    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = (np.log(S / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))

    call = (S * si.norm.cdf(d1, 0.0, 1.0) - K * np.exp(-r * T) * si.norm.cdf(d2,

    return round(call,3)

call_price(69,96,1,0.1,0.2)
```

Out[16]:

```
0.956
```

As the results show, the value of the call option is 0.95 which means the investment is not going to generate a great deal of value that the boss would've expected. The payoff is greater than 0 but not huge. Therefore, I suggest not to proceed with the deal.

Good luck!

Don't forget: google, github and stack overflow are your best friends!

In [ ]: