Machine Learning

Innovative Assignment

Developed By:

- Gahan Saraiya (18MCEC10)
- Priyanka Bhati (18MCEC02)

In [1]:
```
"""
Implementing Classifier with Neural Network and Regression for given dataset
"""
__author__ = ["Gahan Saraiya", "Priyanka Bhati"]
```

In [2]:
```
# Import built-in modules
import os
import numpy as np # linear algebra
import itertools
from subprocess import check_output
from collections import Counter
```

In [3]:
```
# Import 3rd party Python packages
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #for plotting
from sklearn import linear_model
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
# dividing into train and test
from sklearn.model_selection import train_test_split
import seaborn as sns
# print(check_output(["ls", "input"]).decode("utf8"))
# %matplotlib inline
```

In [4]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
batch_size = 64
num_classes = 10
epochs = 20
input_shape = (28, 28, 1)
```

```
Using TensorFlow backend.
```

In [5]:
```
WORKING_DIR = os.getcwd()
DATASET_DIR = os.path.join(WORKING_DIR, "dataset")
DATA_PATH_B = os.path.join(DATASET_DIR, "B.csv")
DATA_PATH_navy = os.path.join(DATASET_DIR, "navy_svm.csv")
```

In [6]:
```
# read data frame
# bdf = pd.read_csv(DATA_PATH_B)
navyData = pd.read_csv(DATA_PATH_navy)
navyData.head()
```

Out[6]:

	class	a1	a2	a3	a4	a5	a6
0	0	235.47	12.22	0.271	25.2	45.6	25.28
1	0	235.47	12.22	0.271	25.2	45.6	25.28
2	0	235.66	12.22	1.061	25.2	45.6	25.28
3	0	235.66	12.11	1.061	25.2	45.6	25.28
4	0	235.66	12.06	1.341	25.2	45.6	25.28

In [7]:
```
navyData.describe()
```

Out[7]:

	class	a1	a2	a3	a4	a5	a6
count	582905.000000	582905.000000	582905.000000	582905.000000	582905.000000	582905.000000	582905.000000
mean	0.283895	232.478022	11.897178	0.784452	21.182873	38.854022	27.888130
std	0.450887	4.017430	0.372777	0.410430	3.781182	4.108154	1.400513
min	0.000000	217.590000	0.160000	0.031000	16.400000	27.900000	-11.550000
25%	0.000000	229.670000	11.640000	0.451000	18.400000	36.800000	26.880000
50%	0.000000	233.740000	11.860000	0.681000	20.100000	39.100000	27.540000
75%	1.000000	235.700000	12.080000	1.051000	22.700000	40.600000	28.670000
max	1.000000	244.600000	16.330000	5.781000	39.300000	52.700000	37.130000

In [8]:
```
navyData.groupby('class').mean()
```

Out[8]:

	a1	a2	a3	a4	a5	a6
class						
0	232.752346	11.800671	0.747009	19.535520	39.021003	27.171677
1	231.786060	12.140608	0.878898	25.338199	38.432825	29.695329

In [9]:
```
##Saperating attributes and labels
navy_data_attrib = navyData.iloc[:, 1:]
# preserves column header rather than converting to series navyData['class']
navy_data_class = navyData.iloc[:, [0]]

attrib_train, attrib_test, class_train, class_test = train_test_split(
                    navy_data_attrib,
                    navy_data_class,
                #     train_size=0.085778,  # Exact 100000 for train size 0.171556
                #     test_size=0.4,
                    shuffle=True, # boolean, optional
                    stratify=None # array-like or None: If not None, data is split
                                #in a stratified fashion, using this as the class labels.
)
```

Define Model in Keras

1. Models in Keras are defined as a sequence of layers
2. We create a Sequential model and add layers one at a time until we are happy with our network topology.
3. The first thing to get right is to ensure the input layer has the right number of inputs. This can be specified when creating the first layer with the input_dim argument and setting it to 6 for the 6 input variables.

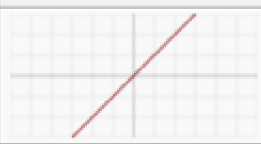
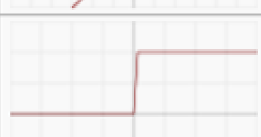
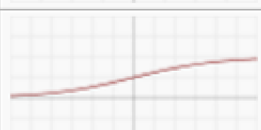
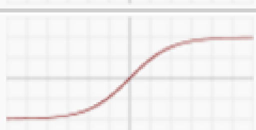
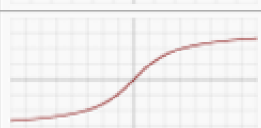
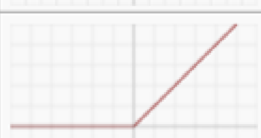
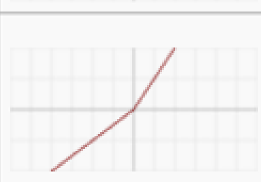
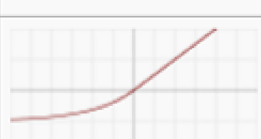
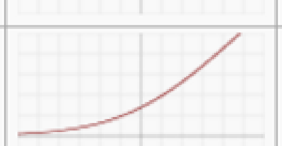
Deciding the number of layers and their types

There are heuristics that we can use and often the best network structure is found through a process of trial and error experimentation. Generally, you need a network large enough to capture the structure of the problem if that helps at all.

> Here We'll first use fully connected net.

Fully connected layers are defined using the `Dense` class. We can specify the number of neurons in the layer as the first argument, the initialization method as the second argument as init and specify the activation function using the activation argument.

Below displayed a list of comparions for activation functions:

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

In [10]:
```
# create model
model = Sequential()
first_layer_neurons = 12
input_dimension = 6
model.add(Dense(first_layer_neurons, input_dim=input_dimension, activation='relu'))
second_layer_neurons = 8
model.add(Dense(second_layer_neurons, activation='relu'))
num_classes = 1
model.add(Dense(num_classes, activation='sigmoid'))
```

```
WARNING:tensorflow:From /home/jarvis/.local/lib/python3.5/site-packages/tensorflow/python/framewor
k/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and wi
ll be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

Compiling the model

Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware, such as CPU or GPU or even distributed.

In this case, we will use logarithmic loss, which for a binary classification problem is defined in Keras as `binary_crossentropy` . We will also use the efficient gradient descent algorithm `adam` for no other reason that it is an efficient default. Learn more about the Adam optimization algorithm in the paper [Adam: A Method for Stochastic Optimization (http://arxiv.org/abs/1412.6980)](http://arxiv.org/abs/1412.6980).

In [11]:
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fit Model

time to execute the model on some data!

We can train or fit our model on our loaded data by calling the `fit()` function on the model.

The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the `nepochs` argument. We can also set the number of instances that are evaluated before a weight update in the network is performed, called the batch size and set using the `batch_size` argument.

Evaluate Model

This will only give us an idea of how well we have modeled the dataset (e.g. train accuracy), but no idea of how well the algorithm might perform on new data. We have done this for simplicity, but ideally, you could separate your data into train and test datasets for training and evaluation of your model.

`evaluate()` - function on model to be used to evaluate. pass it the same input and output used to train the model.

In [12]:
```
for iteration in [20, 50]:
    # Fit the model
    model.fit(attrib_train, class_train, epochs=iteration, batch_size=100)
    # evaluate the model
    scores = model.evaluate(attrib_train, class_train)
    print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
WARNING:tensorflow:From /home/jarvis/.local/lib/python3.5/site-packages/tensorflow/python/ops/math
_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in
a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/20
437178/437178 [==============================] - 8s 19us/step - loss: 0.2820 - acc: 0.8960
Epoch 2/20
437178/437178 [==============================] - 7s 17us/step - loss: 0.0972 - acc: 0.9610
Epoch 3/20
437178/437178 [==============================] - 7s 17us/step - loss: 0.0777 - acc: 0.9676
Epoch 4/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0735 - acc: 0.9690
Epoch 5/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0718 - acc: 0.9700
Epoch 6/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0706 - acc: 0.9705
Epoch 7/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0695 - acc: 0.9712
Epoch 8/20
437178/437178 [==============================] - 6s 13us/step - loss: 0.0693 - acc: 0.9712
Epoch 9/20
437178/437178 [==============================] - 8s 19us/step - loss: 0.0688 - acc: 0.9716
Epoch 10/20
437178/437178 [==============================] - 8s 18us/step - loss: 0.0681 - acc: 0.9715
Epoch 11/20
437178/437178 [==============================] - 6s 14us/step - loss: 0.0671 - acc: 0.9721
Epoch 12/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0657 - acc: 0.9728
Epoch 13/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0658 - acc: 0.9727
Epoch 14/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0654 - acc: 0.9727
Epoch 15/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0637 - acc: 0.9736
Epoch 16/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0631 - acc: 0.9738
Epoch 17/20
437178/437178 [==============================] - 5s 11us/step - loss: 0.0636 - acc: 0.9735
Epoch 18/20
437178/437178 [==============================] - 6s 14us/step - loss: 0.0632 - acc: 0.9735
Epoch 19/20
437178/437178 [==============================] - 8s 18us/step - loss: 0.0627 - acc: 0.9739
Epoch 20/20
437178/437178 [==============================] - 8s 18us/step - loss: 0.0622 - acc: 0.9741
437178/437178 [==============================] - 6s 14us/step

acc: 96.58%
Epoch 1/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0626 - acc: 0.9740
Epoch 2/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0636 - acc: 0.9736
Epoch 3/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0614 - acc: 0.9746
Epoch 4/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0615 - acc: 0.9747
Epoch 5/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0612 - acc: 0.9749
Epoch 6/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0599 - acc: 0.9753
Epoch 7/50
437178/437178 [==============================] - 6s 13us/step - loss: 0.0603 - acc: 0.9751
Epoch 8/50
437178/437178 [==============================] - 8s 18us/step - loss: 0.0603 - acc: 0.9750
Epoch 9/50
437178/437178 [==============================] - 9s 20us/step - loss: 0.0600 - acc: 0.9750
Epoch 10/50
437178/437178 [==============================] - 5s 12us/step - loss: 0.0595 - acc: 0.9754
Epoch 11/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0601 - acc: 0.9753
Epoch 12/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0582 - acc: 0.9758
Epoch 13/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0597 - acc: 0.9753
Epoch 14/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0586 - acc: 0.9757
Epoch 15/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0588 - acc: 0.9757
Epoch 16/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0575 - acc: 0.9763
Epoch 17/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0584 - acc: 0.9759
Epoch 18/50
437178/437178 [==============================] - 7s 17us/step - loss: 0.0577 - acc: 0.9760
Epoch 19/50
437178/437178 [==============================] - 8s 18us/step - loss: 0.0570 - acc: 0.9764
Epoch 20/50
437178/437178 [==============================] - 7s 16us/step - loss: 0.0573 - acc: 0.9765
Epoch 21/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0558 - acc: 0.9771
Epoch 22/50
```

```
437178/437178 [==============================] - 5s 11us/step - loss: 0.0537 - acc: 0.9783
Epoch 23/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0539 - acc: 0.9780
Epoch 24/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0523 - acc: 0.9788
Epoch 25/50
437178/437178 [==============================] - 7s 16us/step - loss: 0.0514 - acc: 0.9794
Epoch 26/50
437178/437178 [==============================] - 8s 18us/step - loss: 0.0503 - acc: 0.9800
Epoch 27/50
437178/437178 [==============================] - 8s 17us/step - loss: 0.0498 - acc: 0.9801
Epoch 28/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0497 - acc: 0.9801
Epoch 29/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0487 - acc: 0.9806
Epoch 30/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0490 - acc: 0.9806
Epoch 31/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0482 - acc: 0.9808
Epoch 32/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0487 - acc: 0.9804
Epoch 33/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0475 - acc: 0.9811
Epoch 34/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0472 - acc: 0.9814
Epoch 35/50
437178/437178 [==============================] - 8s 18us/step - loss: 0.0466 - acc: 0.9814
Epoch 36/50
437178/437178 [==============================] - 7s 17us/step - loss: 0.0468 - acc: 0.9813
Epoch 37/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0475 - acc: 0.9809
Epoch 38/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0466 - acc: 0.9814
Epoch 39/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0455 - acc: 0.9820
Epoch 40/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0457 - acc: 0.9816
Epoch 41/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0455 - acc: 0.9816
Epoch 42/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0460 - acc: 0.9814
Epoch 43/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0511 - acc: 0.9786
Epoch 44/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0620 - acc: 0.9726
Epoch 45/50
437178/437178 [==============================] - 8s 19us/step - loss: 0.0560 - acc: 0.9758
Epoch 46/50
437178/437178 [==============================] - 8s 17us/step - loss: 0.0516 - acc: 0.9783
Epoch 47/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0500 - acc: 0.9791
Epoch 48/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0479 - acc: 0.9804
Epoch 49/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0464 - acc: 0.9809
Epoch 50/50
437178/437178 [==============================] - 5s 11us/step - loss: 0.0465 - acc: 0.9810
437178/437178 [==============================] - 5s 12us/step

acc: 98.54%
```

In [13]:
```python
# create model
model = Sequential()
first_layer_neurons = 128
input_dimension = 6
model.add(Dense(first_layer_neurons, input_dim=input_dimension, activation='relu'))
# model.add(Dropout(0.20))
second_layer_neurons = 64
model.add(Dense(second_layer_neurons, activation='relu'))
num_classes = 1
model.add(Dense(num_classes, activation='relu'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
learning_rate_reduction = ReduceLROnPlateau(monitor='acc',
                                            patience=5,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)

for iteration in [20]:
    # Fit the model
    model.fit(attrib_train,
              class_train,
              epochs=iteration,
              batch_size=100,
              callbacks=[learning_rate_reduction]
             )
    # evaluate the model
    scores = model.evaluate(attrib_train, class_train)
    print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
Epoch 1/20
437178/437178 [==============================] - 7s 15us/step - loss: 4.5807 - acc: 0.7158
Epoch 2/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 3/20
437178/437178 [==============================] - 10s 22us/step - loss: 4.5807 - acc: 0.7158
Epoch 4/20
437178/437178 [==============================] - 8s 19us/step - loss: 4.5807 - acc: 0.7158
Epoch 5/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 6/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 7/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 8/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 9/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 10/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 11/20
437178/437178 [==============================] - 8s 18us/step - loss: 4.5807 - acc: 0.7158

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 12/20
437178/437178 [==============================] - 10s 22us/step - loss: 4.5807 - acc: 0.7158
Epoch 13/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 14/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 15/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 16/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158

Epoch 00016: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 17/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 18/20
437178/437178 [==============================] - 6s 14us/step - loss: 4.5807 - acc: 0.7158
Epoch 19/20
437178/437178 [==============================] - 7s 17us/step - loss: 4.5807 - acc: 0.7158
Epoch 20/20
437178/437178 [==============================] - 11s 24us/step - loss: 4.5807 - acc: 0.7158
437178/437178 [==============================] - 6s 14us/step

acc: 71.58%
```