Hong Seh Looi
frank@minerva.kgi.edu

1. **Moore's Law**
   a. **Extract the date and base speed for a benchmark of your choice**

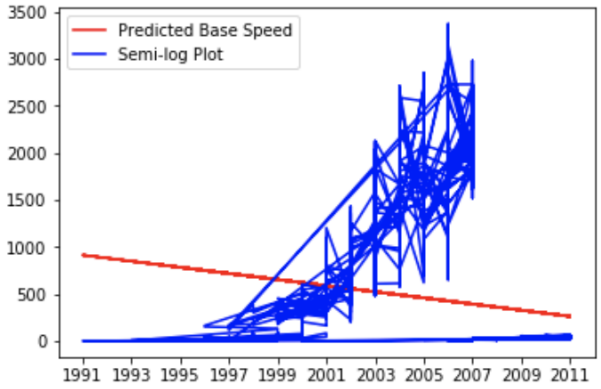
      Refer to Appendix A for Code

   b. **Plot the data in a semi-log plot**

      Refer to Appendix A for Code

   c. **Now train a linear model to fit your plot.**

      Refer to Appendix A for Code



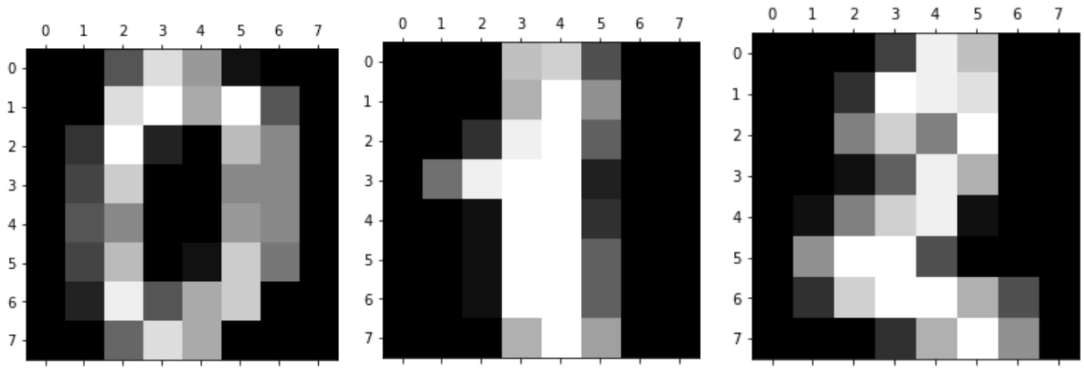

```
The R^2 score for the linear regression model is: -5590.80
The Mean Squared Error is: 472232.07
```

   d. **How well is Moore's law holding up?**

      In general there is a positive linear-semi logarithmic trend over the years which shows that there is an exponential growth in base speed every year. However, from 2006 onwards the base speed suffered a dip, which could be explained by (i) Wirth's Law, which is basically slowdown caused by software bloat or (ii) different threading operations of next-generation processors

introduced in 2006. Our predicted base speed seems to disprove Moore's Law with its negative linear trend, however, our linear regression model is not very reliable because of its low R^2 score and high Mean Squared Error.

2. **MNIST Digits**
   a. **Using Scikit.learn, load the MNIST digits.**
   b. **Plot some of the examples.**



   c. **Choose two digit classes (e.g 7s and 3s) , and train a k-nearest neighbor classifier.**

```
mnist = load_digits(2) # Load digit classes of 0 and 1

X = np.array(mnist.data)
Y = mnist.target

def KNN(X,Y,initial_k,k_increment):
    # k-nearest neighbor where X and Y = Training data and labels.
    # Initial_k = Starting k-value, k-increment = increment in k during test

    # Splitting data into training and testing sets
    (trainData, testData, trainLabels, testLabels) = train_test_split(X, Y,
test_size=0.25, random_state=42)

    # Further splitting training set into validation set to find optimal value of k
    (trainData, valData, trainLabels, valLabels) = train_test_split(trainData,
trainLabels,test_size=0.25, random_state=84)

    # Declare array to store all possible k-values and accuracy values of each
k-value
    kVals = range(initial_k, len(valData)+1, k_increment)
    accuracies = []

    print("Testing the best value of k on validation dataset of size %d" %
len(valLabels))

    # Runs KNN for all values of k within range and select k with the highest
accuracy
    for k in kVals:
        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(trainData, trainLabels)

        score = model.score(valData, valLabels)
```

```
        accuracies.append(score)

    index = np.argmax(accuracies)

    print("The best value of k is %d with accuracy score of %.2f" % (kVals[index],
accuracies[index] * 100))

    # Run KNN with the best value of k and predict the test data
    model = KNeighborsClassifier(n_neighbors=kVals[index])
    model.fit(trainData, trainLabels)
    predictions = model.predict(testData)

    # Report the accuracy for predicted values of test
    print ("")
    print("Evaluation Report")
    print(classification_report(testLabels, predictions))

KNN(X,Y,1,1)

# Model accuracy improved with codes adopted from
https://gurus.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/
```

d. Report your error rates on a held out part of the data.

```
Testing the best value of k on validation dataset of size 68
The best value of k is 1 with accuracy score of 100.00

Evaluation Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      1.00      1.00        44

avg / total       1.00      1.00      1.00        90
```

e. (Optional) Test your model on the full dataset

See Appendix B for code

```
Testing the best value of k on validation dataset of size 1875
The best value of k is 1 with accuracy score of 94.99

Evaluation Report
             precision    recall  f1-score   support

          0       0.95      0.98      0.96       247
          1       0.97      1.00      0.98       285
          2       0.94      0.91      0.92       266
          3       0.92      0.93      0.92       256
          4       0.93      0.92      0.92       261
          5       0.94      0.92      0.93       204
          6       0.95      0.96      0.96       248
          7       0.91      0.95      0.93       238
          8       0.97      0.88      0.92       239
          9       0.90      0.90      0.90       256

avg / total       0.94      0.94      0.94      2500
```

Hong Seh Looi
frank@minerva.kgi.edu

Exercise 1.14

A redistribution lottery involves picking the correct four numbers from 1 to 9 (without replacement, so 3,4,4,1 for example is not possible). The order of the picked numbers is irrelevant.

Every week a million people play this game, each paying £1 to enter, with the numbers 3,5,7,9 being the most popular (1 in every 100 people chooses these numbers). Given that the million pounds prize money is split equally between winners, and that any four (different) numbers come up at random, what is the expected amount of money each of the players choosing 3,5,7,9 will win each week?

The least popular set of numbers is 1,2,3,4 with only 1 in 10,000 people choosing this. How much do they profit each week, on average? Do you think there is any 'skill' involved in playing this lottery?

Total possible numbers = ${}^{9}C_{4}$ = 126

Probability of [3,5,7,9] = 0.01
Number of people with [3,5,7,9] = 1,000,000 * 0.01 = 10,000
Probability of [3,5,7,9] being the winning number this week = 1/126

Expected amount of money each player win = 1,000,000 / (10,000 * 126) = £0.79

Probability of [1,2,3,4] = 0.0001
Number of people with [1,2,3,4] = 1,000,000 * 0.0001 = 100
Probability of [1,2,3,4] being the winning number this week = 1/126

Expected amount of money each player win = 1,000,000 / (100 * 126) = £79.37

Since the 4 numbers and the winning combination are both picked randomly, the only thing we can somewhat control is the commonality of our number-at-hand. The people who bet against the odds by picking the least popular numbers (eg: 1,2,3,4) will split their winnings with less people. At any given week, we should attempt the lottery as many time as possible to choose a non-popular combination. This will ensure we win a larger amount of money (if and only if we win). If we got a common combination, we could keep it and continue drawing numbers to get an uncommon combination.

Exercise 13.5

WowCo.com is a new startup prediction company. After years of failures, they eventually find a neural network with a trillion hidden units that achieves zero test error on every learning problem posted on the internet up to last week.

Each learning problem included a train and test set. Proud of their achievement, they market their product aggressively with the claim that it 'predicts perfectly on all known problems'. Discuss whether or not these claims would justify buying this product.

If I had no current dataset or bad training/test dataset about the problem I am facing (eg: a purely qualitative, unquantifiable problem), the claim would not justify purchasing the product. This is because neural networks are trained and tested on existing data. If there was no way of representing the problem for computers to understand, neural network would not work. This is unlike human intuition or expert diagnosis, which is what the neural network is somewhat based on.

I would buy the product if I merely need to be able to make good classification or regression predictions without understanding *why* the predictions were good. This is because there are too many hidden units (a trillion) that goes behind the prediction that makes explaining the prediction nearly impossible. We can ask why the product predicted things a certain way but would not get an isolated answer as to why, and therefore cannot act to change the predicted outcome if they were unfavorable to us (eg: predicting a cyber attack without knowing the exact source and nature). Without knowing the mechanics behind the predictions, this neural network model are hence not considered very generalizable. This means that beyond the test data, we always run the risk of having a model that is overfitted or under-fitted when making predictions. The hardest thing is that we won't be able to know which exact parameter to tweak during supervised learning, such as bias. I would justifiably purchase the product to the extent that the predicted outcome is more important than the explanation process.

Appendix A - Code for Moore's Law

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.model_selection import train_test_split
import sklearn.metrics
import numpy as np

data = pd.read_csv('summaries.txt')
data = data.drop(['testID','tester','machine','cpu','mhz','os','compiler',
                  'autoParallel','benchType','peak'],axis=1)

data['hwAvail'] =
data['hwAvail'].str.strip('AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz-')

X = data['hwAvail']
Y = data['base'] # Base speed as dependent variable

YEARS = [1991, 1993, 1995, 1997, 1999, 2001, 2003, 2005, 2007, 2009, 2011]

plt.semilogy(X, Y)
plt.xlabel("Year")
plt.ylabel("Base Speed")
plt.xticks(YEARS)
plt.title('Moore\'s Law')
plt.show()
```

```
X_train, X_test, Y_train, Y_test = train_test_split(data['hwAvail'].values.reshape(-1,1),
                                                    data['base'].values.reshape(-1,1),
                                                    test_size=0.33, random_state=42)

reg = linear_model.LinearRegression()
reg = reg.fit(X_train, Y_train)

pred = reg.predict(X_test)
score = reg.score(pred, Y_test)
MSE = sklearn.metrics.mean_squared_error(Y_test,pred)

plt.plot(X_test,pred,color='r',label='Predicted Base Speed')
plt.plot(X,Y,color='b',label="Semi-log Plot")
plt.legend()
plt.xticks(YEARS)
plt.show()

print "The R^2 score for the linear regression model is: %.2f" % score
print "The Mean Squared Error is: %.2f" % MSE
```

Appendix B - Code for MNIST Full Database

```
from mnist import MNIST

# Test the full MNIST data
mndata = MNIST('')
images, labels = mndata.load_training()
images, labels = mndata.load_testing()

KNN(images,labels,1,3000)
```