1. Algorithmic Problems

   All scripts are built using Python 3.6.5 and the example of applying the scripts is done on Anaconda Python 3.6.5.

   a. Given A, a set of N integers $[a_1, a_2, \dots, a_n]$ and each integer ranges between $A_{low} \geq 1$ and $A_{high} \leq 100$. Now, we want to find a set of integers that has difference between the maximum and the minimum of the set is less than or equal to MAX_DIFF. Beside that, the set also needs to minimize the cost, the square of Euclidean distance to A. The script to solve this problem is a function “min_dist_100()” in file “minimum_distance.py” which is the input are the length of A, MAX_DIFF, and set A, whereas the output is an integer number that shows the cost between A and B. The example of applying this script is in file “Test_Number_1.ipynb”.
   b. Similar case with case (1.a.). However, the range of set A is too higher than before $\left(1 \leq A_{low} \leq a_i \leq A_{high} \leq 1000000000\right)$. To make the model runs faster, we loop the code use multiple of one hundred first. After find the minimum, we run it like number (1.a). The script for this problem is a function “min_dist()” in the same file “minimum_distance.py” also. The example of applying this function is also in the file “Test_Number_1.ipynb”.

2. Big Data

   All scripts are built using Python 3.6.5 and the example of applying the scripts is done on Anaconda Python 3.6.5.

   a. Given a textfile data, “name_age.txt”, contains the name and the age of the whole world (~7 billion people). Also we only have a computer on hand with 1 GB of RAM. From calculation, the textfile of ~7 billion string data will take ~56 GB of memory space (assuming it's 32 bit integer, a long string each line, so it is 7 billion x 8 byte). The problem arises when we only have 1 GB of RAM. Our RAM cannot hold the data to sort it since it is 56 GB. So, we split the big data into some chunk files, the size of each chunk file is lower than our RAM size of course, name it “chunk_memory”. Each chunk file also contains sorted data. When we finish splitting the data, we merge it into one file (doing comparation in RAM). The buffer memory size also doesn't exceed the “chunk_memory”, so if we want the total buffer memory remains the same as “chunk_memory”, the buffer memory size must be “chunk_memory” divided by total of created chunks. The name of script file to sort the data is “SortingBigFile.py”. The main function to do sort in the file is “sort_as_chunk()”, it takes four arguments, the input filename (which is “name_age.txt”), memory of chunk, flag to clean up the chunk files, and format of split chunk files and the output is a file (which is “name_age_sorted.txt”) contains sorted data. The file

    "SortingBigFile.py" depends on file "memory_chunk_module.py", the files use Python build-in library. The method above has a price, when we use chunk data stored in hard-disk memory (which is slower than RAM), the sorting process will take some time. It will be slower when the chunk memory is low (many chunk files). But there are some methods to optimize it, we can use multithreaded process for each chunk file and/or use GPU to make the process of sorting faster. The example of applying of this script is in the file named "Test_Number_2.ipynb".

    b. Assuming we want to create API for this case. We have to write two functions, "initalize(blacklist)", it takes string input which is the blacklist we have, called when the API is starting, and "check_blacklist(name, phone_number)", this function takes two arguments and return Boolean value whether the name with the phone number is in blacklist or not. The name of script file to solve it is "BlackListAPI.py". When the script starts, it will call "initialize()" function, and then do check the name and phone number. The example of applying of this script is also in the file named "Test_Number_2.ipynb".
    c. Continuing the blacklist on (2.b.), but now suppose we are being attacked by hacker. There are a lot of fake name and phone number that we want to blacklist. The blacklist grow onto 10 billion of name and phone number so it cannot longer fit in our RAM. To deal with this problem, we just need to do as what we do in (2.a.) that use sorted chunk files to reduce RAM usage (in function "initialize()"), but instead of merge it into single sorted file, we do binary search for each chunk file for the name and phone number (in function "check_blacklist()"). Both function with chunk_memory, we put in "BlackListAPI_mcm.py". The example of applying of this script is also in the file named "Test_Number_2.ipynb".

3. Machine Learning
    a. There are several methods usually used to deal with this kind problems. However, I suggest to use Principal Component Analysis (PCA) or Kernel PCA to reduce the number of features, depends of linearity among features, to get several principal components before building the model using the principal components. The number of principal components is recommended to be determined by the variance of principal components. The variance is recommended to meet minimum 80% of variance of the whole principal components.
    b. The possible problem occur on the case, in my opinion, is overfitting. It is rarely but possible to occur when we just split data to be training data to build the model and testing data to validate the model. To handle this problems, I suggest several methods:

1) Firstly, we can use Cross-Validation Method (you can join it with K-Fold method). By using Cross-Validation, the training data will be split to be several new training data and new testing data and the first testing data will be the validation data. Therefore, we can get several number of accuracy scores from the new training data and the new testing data. The final accuracy we can get by the mean of all accuracy score. However, we need also to make sure that the all accuracy scores meet the normal distribution so we can sure that the mean is not bias or overfitting. If the accuracy doesn't meet the normal distribution, it means that we need to change the method that build the model or change the parameter of the model. But if the accuracy score has met the normal distribution, we can build the final model using the first training data and validate it using the first testing data.
2) Secondly, we need to make sure that there is no imbalanced data occur in that case. If imbalanced data occur, we can modified the data using resampling data method or SMOTE method to balance the data before applying Cross-Validation. We also need to add other methods to score the result not only just use accuracy, such as: F1 Score, Precision, Recall, Confusion Metrics, ROC AUC Score, and Cohen Kappa score to see the balance and weighted of the score among all classes.

c. SVM method uses a hyperplane to separate two classes. By that, SVM only can handle data with real numbers, so the first step we need to encoder all text data to be real numbers by encoding the data to be categorical types. Especially for the target classification (gender), we need to apply a binary encoder. The second is, we need to scale (normalize or standardize) the data, so no feature that can be dominate others when model calculating the distance among samples. The scale also important to make all features to be non-dimensional, so all features have same influence on the distance metrics.

4. Statistics

a. The expected of number of die rolls required to get three same consecutive outcomes. For instance, without loss of generality, we take the case 222 as the problems and let *E* denote as the answer.

Consider the first three rolls. There are 4 cases:

1) Case 1:

The first roll is a non-two $\left(\text{probability } \frac{5}{6}\right)$. We wind up at the start after one roll.

2) Case 2:

The first roll is a two, but the second is a non-two $\left(\text{probability } \frac{1}{6} \times \frac{5}{6} = \frac{5}{36}\right)$. We wind up back at the start after two rolls.

3) Case 3:

The first two rolls are twos, but the third is a non-two $\left(\text{probability } \frac{1}{6} \times \frac{1}{6} \times \frac{5}{6} = \frac{5}{216}\right)$. We wind up back at the start after three rolls.

4) Case 4:

The first three rolls are twos $\left(\text{probability } \frac{1}{6} \times \frac{1}{6} \times \frac{1}{6} = \frac{1}{216}\right)$. We win.

Thus we have:

$$E = \frac{5}{6}(E+1) + \frac{5}{36}(E+2) + \frac{5}{216}(E+3) + \frac{1}{216}(3) \rightarrow E = 258.$$

So after 258 number of die rolls, we expect to get three same consecutive outcomes.

b. We throw 8 dice and take the sum of the highest 4 outcomes. The sum of the highest 4 outcomes equals to 24 means there are 4 dice that must get 6s and other are not equal to 6. Therefore, the probability of 4 dice get 6 are $\left(\frac{1}{6}\right)^4$ and the probability of another 4 dice get no 6 are $\left(C_4^8 \times \left(\frac{5}{6}\right)^{8-4}\right)$. So, the probability of the sum of the highest 4 outcomes of throwing 8 dice equal to 24 can be computed a:

$$\left(C_4^8 \times \left(\frac{1}{6}\right)^4 \times \left(\frac{5}{6}\right)^{8-4} = \frac{43750}{1679616} \approx 0.026.\right)$$

c. Let $I_i, i = 1,2,\ldots,6$ be random variables indicating if number *i* appears in the outcomes. More precisely, $I_i = 1$ if at least one outcome is number *i,* and $I_i = 0$ if no outcome is number *i*. The total number of distinct outcomes is therefore $N = \sum_{i=1}^{6} I_i$. For any given *i*, $I_i = 0$ only if all the 6 outcomes take other 5 possible values but not *i.* This happens with probability $\left(\frac{5}{6}\right)^6$. Hence , $E(I_i) = 1 - \left(\frac{5}{6}\right)^6$. Therefore:

$$E(N) = E\left(\sum_{i=1}^{6} I_i\right) = \sum_{i=1}^{6} E(I_i) = 6 \times \left(1 - \left(\frac{5}{6}\right)^6\right) \approx 3.99.$$

d. We throw a die 10000 times and record the sum of outcomes. Now we want to approximate the probability that the sum is in range of $[34500,35500]$.

Let $I_i, i = 1,2,\ldots,6$ be random variables indicating if number *i* appears in the outcomes. For a single roll, expected mean and variance is computed as:

$$E(I) = \frac{(1+2+3+4+5+6)}{6} = 3.5$$

$$Var(I) = E(I^2) - E^2(I) \approx 2.916667$$

So, the mean and the variance of 10000 rolls of a die is

$$\mu = 10000 \times E(I) = 35000$$

$$\sigma^2 = 10000 \times Var(I) \approx 29166.67 \rightarrow \sigma \approx 170.783$$

Therefore, the probability that the sum would be in the range of $[34500,35500]$ can be computed as:

$$\begin{aligned} P(34500 \leq W \leq 35500) &\approx P\left(\frac{34500 - 35000}{170.783} \leq Z \leq \frac{35500 - 35000}{170.783}\right) \\ &\approx P(-2.9277 \leq Z \leq 2.9277) \\ &= P(Z \leq 2.9277) - P(Z \leq -2.9277) \\ &\approx 0.9966 \end{aligned}$$