Name: Han Nguyen

NetID: TXN200004

Homework 4

Problem 1:

Each row of the Gisette data has 5001 entries, first entry is class label $y_i \in \{-1, +1\}$, the remaining 5000 entries are the features $x_i \in \mathbb{R}^{5000}$. $X = \begin{bmatrix} x_1 \\ \vdots \\ x_{5000} \end{bmatrix}$

$n = 5000$

Data point: M

a) SVMs and PCA:

Problem1_a.py

- Perform PCA on the training data

Step 1: Center the data:

- $\mu = \text{mean}(X)$
- $\overline{X} = X - \mu$

Step 2: Compute covariant matrix

$$\text{Cov\_matrix} = \frac{1}{M-1} \cdot \overline{X}^T \cdot X$$

Step 3 : Compute eigenvalue, eigenvector

$$Cov\_Matrix = Q D Q^T$$

Simply use np.linalg.eigh(cov_matrix) to calculate

$$D = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix} \qquad Q = \begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix}$$

Step4 : Find best 6 λ

=> Sort D, Q by descending order to get 6 biggest λ

```
Eigenvalue 1: 16440913.57
Eigenvalue 2: 12009807.65
Eigenvalue 3: 9726589.20
Eigenvalue 4: 7483337.87
Eigenvalue 5: 6784373.74
Eigenvalue 6: 5485503.60
```

- Build a set of $K = \{k_{99}, k_{95}, k_{90}, k_{80}, k_{75}\}$

- Calculate total variance

$$V_{total} = \sum_{j=1}^{n} \lambda_j$$

- Calculat cummulative variance

$$V(m) = \sum_{j=1}^{m} \lambda_j$$

- Explained variance

$$Explained(m) = \frac{V(m)}{V_{total}}$$

Then pick : $k_z = \min\{m : \text{Explained}(m) \geq z\}$

Result: $k = \{2695, 1761, 1320, 855, 700\}$

```
k99 (explains 99.0% variance):
  k = 2695 components

k95 (explains 95.0% variance):
  k = 1761 components

k90 (explains 90.0% variance):
  k = 1320 components

k80 (explains 80.0% variance):
  k = 855 components

k75 (explains 75.0% variance):
  k = 700 components

Set K = {'k99': 2695, 'k95': 1761, 'k90': 1320, 'k80': 855, 'k75': 700}
```

- Train SVM for each k in K

  For each k value = number of components

  - Take the first k best eigenvector in PCA part

  - $X_{train} = (X_{train} - \mu) \cdot Q_k$

    $X_{validate} = (X_{validate} - \mu) \cdot Q_k$

    $X_{test} = (X_{test} - \mu) \cdot Q_k$

  - Then train under SVM with Gaussian kernel

    Use validation data to tune slack and variance $c = [0.1, 1, 10]$

    $\sigma = [0.1, 1, 10]$

- Report k, c, and σ

```
----------------------------------------------------------------
Training SVM with k99 = 2695 components
----------------------------------------------------------------
Projected training data shape: (6000, 2695)
Tuning hyperparameters on validation set...
Best hyperparameters: C=1, sigma=10000
Validation accuracy: 97.80%
Test accuracy: 98.00%
Test error: 2.00%
```

K_{99} : 2695 components

$C = 1$ Test acc = 98%

$\sigma = 10000$

```
----------------------------------------------------------------
Training SVM with k95 = 1761 components
----------------------------------------------------------------
Projected training data shape: (6000, 1761)
Tuning hyperparameters on validation set...
Best hyperparameters: C=1, sigma=10000
Validation accuracy: 97.80%
Test accuracy: 98.00%
Test error: 2.00%
```

K_{95} : 1761 components

$C = 1$ Test acc = 98%

$\sigma = 10000$

```
----------------------------------------------------------------
Training SVM with k90 = 1320 components
----------------------------------------------------------------
Projected training data shape: (6000, 1320)
Tuning hyperparameters on validation set...
Best hyperparameters: C=1, sigma=10000
Validation accuracy: 97.80%
Test accuracy: 98.00%
Test error: 2.00%
```

K_{90} : 1320 component

$C = 1$ Test acc = 98%

$\sigma = 10000$

```
----------------------------------------------------------------
Training SVM with k80 = 855 components
----------------------------------------------------------------
Projected training data shape: (6000, 855)
Tuning hyperparameters on validation set...
Best hyperparameters: C=1, sigma=10000
Validation accuracy: 97.80%
Test accuracy: 98.20%
Test error: 1.80%
```

K_{80} : 855 component

$C = 1$ Test acc = 98.2%

$\sigma = 10000$

```
----------------------------------------------------------------
Training SVM with k75 = 700 components
----------------------------------------------------------------
Projected training data shape: (6000, 700)
Tuning hyperparameters on validation set...
Best hyperparameters: C=10, sigma=10000
Validation accuracy: 98.00%
Test accuracy: 97.80%
Test error: 2.20%
```

K_{75} : 700 component

$C = 10$ Test acc = 97.8%

$\sigma = 10000$

→ Best model : K_{80}, $C = 1$, $\sigma = 10000$

- Comparision to best SVM without PCA

```
============================================================
BASELINE: SVM WITHOUT PCA (All Original Features)
============================================================

Training SVM with ALL 5000 original features (no PCA)
Tuning hyperparameters on validation set...
  Trying C=0.1, sigma=3000... Val Acc: 69.60%
  Trying C=0.1, sigma=10000... Val Acc: 92.80%
  Trying C=0.1, sigma=30000... Val Acc: 93.20%
  Trying C=1, sigma=3000... Val Acc: 69.60%
  Trying C=1, sigma=10000... Val Acc: 97.80%
  Trying C=1, sigma=30000... Val Acc: 97.20%
  Trying C=10, sigma=3000... Val Acc: 69.60%
  Trying C=10, sigma=10000... Val Acc: 97.80%
  Trying C=10, sigma=30000... Val Acc: 97.40%

Best hyperparameters: C=1, sigma=10000
Validation accuracy: 97.80%
Training final model with best hyperparameters...

============================================================
BASELINE RESULTS
============================================================
Features used: ALL 5000 original features (no PCA)
Best C: 1
Best sigma: 10000
Validation accuracy: 97.80%
Test accuracy: 98.00%
Test error: 2.00%
============================================================
```

SVM without PCA

$C = 1$

$\sigma = 10000$

Test acc = 98%

- PCM best model (k_{80}) performs slightly better than SVM without PCA (98.2% vs 98%)

- Observation:
  + PCA achives better performance using only 855 features compared to 5000 of full SVM
  + Massive dimensionality reduction → reduce computational resources
  + PCA filter out noise without reducing accuracy

- Choosing k before evaluating on validation set
  - I will choose $k = 90$, if computation resource is limited I can choose $k = 80$ or 85
  - It massively reduce the dimension but still not overly simplifies the data set

b) PCA for feature selection

Problem 1-b.py

- Why π define propapility distribution?

$$\pi_j = \frac{1}{k} \sum_{i=1}^{k} v_j^{(i)^2}$$

I use scikit_learn for this one since using cvxopt take a very long time to get result.

- Each eigenvector has unit form

$$\sum_j \left(v_j^{(i)}\right)^2 = 1$$

$$\rightarrow \sum_j \pi_j = \sum_j \frac{1}{k} \sum_i^k v_j^{(i)^2} = \frac{1}{k} \sum_i^k \left(\sum_j v_j^{(i)^2} \right) = \frac{1}{k} \cdot k = 1$$

π is non-negative and sums to 1, making it a valid probablity distribution

- Result of experiments

```
================================================================================
RESULTS SUMMARY (Test Error %)
================================================================================

k value     s=10        s=20        s=30        s=40        s=50        s=60        s=70        s=80        s=90        s=100
--------------------------------------------------------------------------------------------------------------------------------
k99 (k=2695)39.84±6.55  33.86±7.10  30.90±5.06  29.64±5.05  25.06±3.92  22.58±4.81  22.70±4.77  20.82±3.18  19.16±4.16  17.40±5.01
k95 (k=1761)46.04±5.11  36.34±4.98  29.78±8.16  28.20±5.93  26.98±4.85  27.10±5.02  25.26±3.32  24.20±5.01  21.40±5.04  20.46±4.02
k90 (k=1320)46.06±4.77  39.46±5.05  34.44±5.38  32.76±8.02  30.96±5.14  28.38±4.65  24.72±6.42  23.96±4.25  26.42±6.71  23.42±5.45
k80 (k=855 )41.00±6.77  35.56±5.22  30.78±7.68  30.14±8.08  26.08±6.57  26.98±6.20  21.14±3.39  19.96±3.76  20.38±4.27  20.90±6.40
k75 (k=700 )37.78±6.04  33.84±5.74  34.48±5.73  29.90±7.09  29.84±6.30  25.50±4.63  22.58±3.86  23.26±3.95  21.30±3.76  19.26±4.01
--------------------------------------------------------------------------------------------------------------------------------
```

- Based on the results, this method we can compare with the baseline model from 1a

```
================================================
BEST CONFIGURATIONS
================================================

Best for k99 (k=2695): s=100, error=17.40%

Best for k95 (k=1761): s=100, error=20.46%

Best for k90 (k=1320): s=100, error=23.42%

Best for k80 (k=855): s=80, error=19.96%

Best for k75 (k=700): s=100, error=19.26%

------------------------------------------------
BEST OVERALL: k99 (k=2695), s=100
Mean test error: 17.40%
------------------------------------------------
```

$S = 100$

→ k_{99} selected

Test error : 17.40%

```
============================================================
COMPARISON WITH BASELINES
============================================================

Feature Selection Best: 17.40% (k=2695, s=100)
PCA Projection Best (from 1a): 1.80% (k=855)
No Feature Selection (from 1a): 2.00% (all 5000 features)
============================================================
```

Compared to best model in 1_a (k_{80}) this method produce a higher error (17.4% vs 1.8%)

- Does this a reasonable alternative ?
  - It depends on the goal, if we can accept higher error then it could work since we use significantly less feature (100 vs 855)

- But in this particular case, I don't think it's a good alternative, since in 1-a going from 5000 -> 855 components already a big reduction
- Unless we face a slow down then we can try to use feature selection

* Pros:
  - Dimensionality reduction
  - Noise filtering
* Cons:
  - Unsupervised selection
  - Higher error

Problem 2: $X \sim \lambda e^{-\lambda x}$ $(\lambda > 0, x > 0)$

(1.) Maximum likelihood estimate for λ

The likelihood:

$$P(D|\lambda) = \lambda^m \exp\left(-\lambda \sum_{i=1}^{m} x^{(i)}\right)$$

Log-likelihood:

$$\ell(\lambda) = m \log\lambda - \lambda \sum_{i=1}^{m} x^{(i)}$$

$$\frac{dl}{d\lambda} = 0 \Leftrightarrow \frac{m}{\lambda} - \sum_{i=1}^{m} x^{(i)} = 0$$

$\Rightarrow$ Maximum likelihood

$$\hat{\lambda}_{MLE} = \frac{m}{\sum_{i=1}^{m} x^{(i)}} = \frac{1}{\bar{x}}$$

2.

MAP depends on the prior through

$$p(\lambda|D) \propto p(D|\lambda)p(\lambda)$$

If the prior assign zero probability to the true parameter value, the the posterior can never place mass there, regardless of how much data is observed

Example : Suppose the true parameter is $\lambda_0 = 7$

Choose a prior

$$p(\lambda) = \begin{cases} 1 & (0 \le \lambda \le 5) \\ 0 & \text{otherwise} \end{cases}$$

This will fail to converge because

- $P(D|\lambda)$ will peak around $\lambda = 7$
- the prior $P(\lambda)$ is 0 at $\lambda = 7$
- The posterior $= 7 \times 0 = 0$

3. Gamma prior over λ:

$$\lambda \sim \text{Gamma}(\alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda} \quad (\alpha, \beta > 0)$$

- MAP estimate

$$P(\lambda|D) \propto P(D|\lambda) \cdot P(\lambda)$$

- $P(D|\lambda) = \prod_{i=1}^{m} \lambda e^{-\lambda x^{(i)}} = \lambda^m e^{-\lambda \sum_{i=1}^{m} x^{(i)}}$

- $P(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda}$ We can ignore $\frac{\beta^\alpha}{\Gamma(\alpha)}$ constant during maximization

Posterior $\propto \left(\lambda^m e^{-\lambda \sum_{i=1}^{m} x^{(i)}}\right) \cdot \left(\lambda^{\alpha-1} e^{-\beta\lambda}\right)$

$$= \lambda^{m+\alpha-1} e^{-\lambda\left(\sum_{i=1}^{m} x^{(i)} + \beta\right)}$$

-

$$\ln(\text{Posterior}) = (m+\alpha-1)\ln(\lambda) - \lambda\left(\sum_{i=1}^{m} x^{(i)} + \beta\right)$$

- $\frac{dL}{d\lambda} = 0 \iff \frac{m+\alpha-1}{\lambda} - \left(\sum_{i=1}^{m} x^{(i)} + \beta\right) = 0$

$\iff$

$$\hat{\lambda}_{MAP} = \frac{m+\alpha-1}{\sum_{i=1}^{m} x^{(i)} + \beta}$$

-

we have:

- $\hat{\lambda}_{MLE} = \frac{m}{\sum_{i=1}^{m} x^{(i)}}$

- $\lim_{m\to\infty} \hat{\lambda}_{MAP} = \lim_{m\to\infty} \frac{1 + \frac{\alpha-1}{m}}{\frac{\sum_{i=1}^{m} x^{(i)}}{m} + \frac{\beta}{m}} = \frac{m}{\sum_{i=1}^{m} x^{(i)}} = \hat{\lambda}_{MLE}$

the MAP estimate does converge to MLE as $m \to \infty$

Problem 3: Problem3.py

(1). Given set $k \in \{10, 20, 30, 36, 40\}$

a) For each k value

. Initialization:

$$\mu_j^{(0)} = x_{rand(j)}, \quad j = 1, \ldots, k$$

$$rand(j) \sim Uniform(\{1, \ldots, n\})$$

. Assignment step:

$$c_i = \arg\min_j \|x_i - \mu_j\|^2$$

. Update step:

$$\mu_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

. Objective

$$J = \sum_i \|x_i - \mu_{c_i}\|^2$$

Repeat 100 runs, record mean & std of J, and purity

Result:

```
--------------------------------------------------------------------------------
Part 1(a): Standard K-means (random initialization)
--------------------------------------------------------------------------------
k          Mean Objective          Std Objective           Mean Purity (%)
--------------------------------------------------------------------------------
10         1076.93                 74.17                   29.19
20         649.52                  57.95                   44.65
30         490.46                  33.43                   53.31
36         428.53                  36.13                   57.84
40         393.58                  21.35                   60.25
```

(b) For each k value

Initialization

$$\mu_j^{(0)} = X_{rand}$$

For each new center:

Compute:

$$d_i = \min_{j<t} \|x_i - \mu_j\|^2$$

Sample new center with

$$P(x_i) = \frac{d_i}{\sum_k d_k}$$

Run the k-mean with the same update steps in 1_a

Run 100 times, record mean, std of J and purity

Result:

```
--------------------------------------------------------------------------------
Part 1(b): K-means++ initialization
--------------------------------------------------------------------------------
k          Mean Objective          Std Objective           Mean Purity (%)
--------------------------------------------------------------------------------
10         1025.05                 58.02                   28.38
20         614.89                  22.65                   42.82
30         457.10                  17.11                   52.42
36         392.05                  14.20                   56.25
40         358.22                  12.65                   58.54
```

② Comparision

Comparison: K-means vs K-means++

k	KM Obj	KM++ Obj	Improv (%)	KM Pur (%)	KM++ Pur (%)
10	1076.93	1025.05	4.82	29.19	28.38
20	649.52	614.89	5.33	44.65	42.82
30	490.46	457.10	6.80	53.31	52.42
36	428.53	392.05	8.51	57.84	56.25
40	393.58	358.22	8.99	60.25	58.54

- k-means++ shows lower objective
- k-means++ shows improvement increase
- Purity is higher in k-means but the different is low

→ I think k-means++ perforns better