STATS 204 - Final Report

Jiayu Wu - 905054229

Problem 1. Spectral Clustering

(a, b) As displayed in Fig. 1-1, the bigger the σ, the more "overlap" among groups. The reason is that a bigger σ gives a higher measure of similarity, so different group are more likely to overlap.

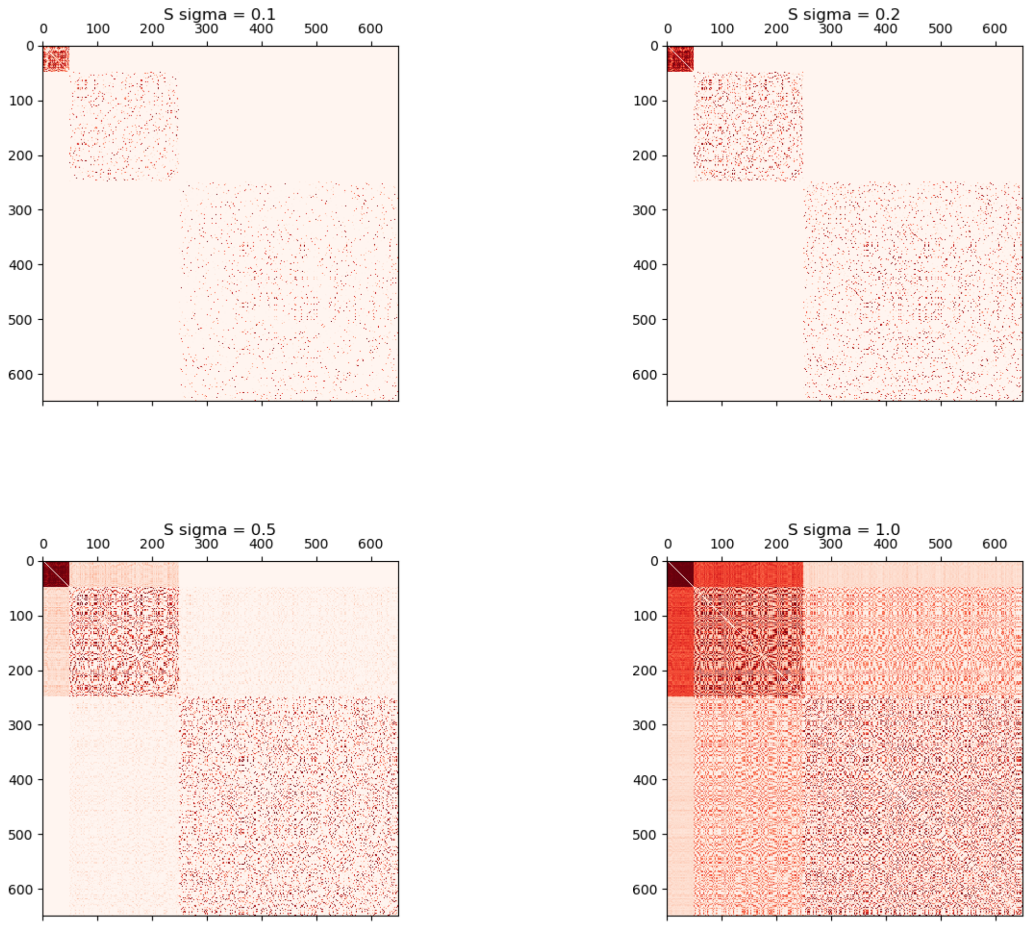

Fig. 1-1

(b) In Fig. 1-2, it can be observed that with smaller σ the eigenvectors are distinctly structured into three groups, while with bigger σ even points far away are measured as in proximity, so the eigenvectors does not plays a distinct structure.

(d, e) The choice of scaling $\sigma = 0.2$ and eig$= 2$ gives the optimal classification with 100& accuracy.

Problem 2. t-SNE

(a) In the t-SNE algorithm, we map the high-dimensional data X to a d=2 dimensional Y. There are the following steps:

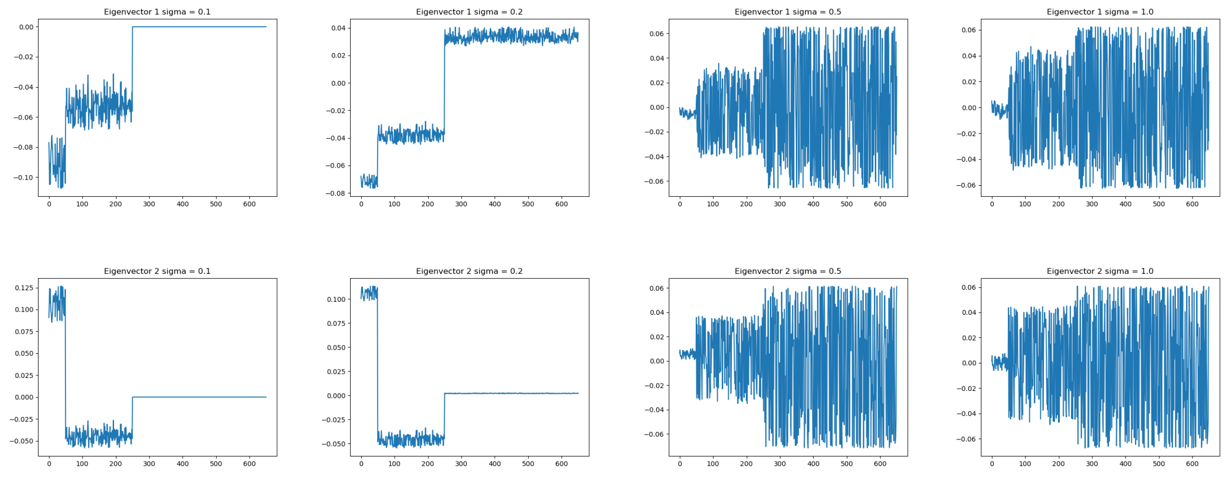
Eigenvector 1 sigma = 0.1
Eigenvector 1 sigma = 0.2
Eigenvector 1 sigma = 0.5
Eigenvector 1 sigma = 1.0
Eigenvector 2 sigma = 0.1
Eigenvector 2 sigma = 0.2
Eigenvector 2 sigma = 0.5
Eigenvector 2 sigma = 1.0

Fig. 1-2

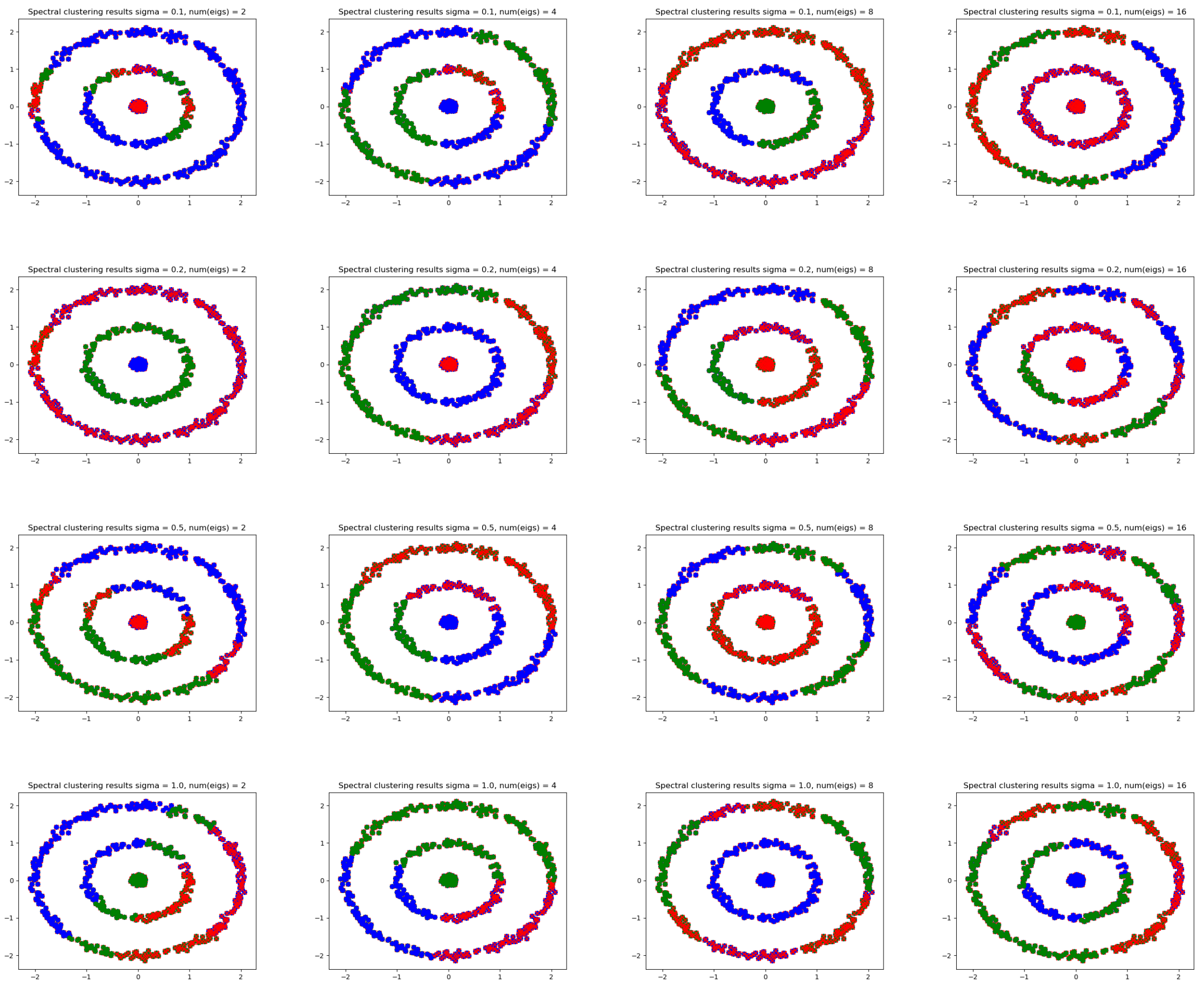
Spectral clustering results sigma = 0.1, num(eigs) = 2
Spectral clustering results sigma = 0.1, num(eigs) = 4
Spectral clustering results sigma = 0.1, num(eigs) = 8
Spectral clustering results sigma = 0.1, num(eigs) = 16
Spectral clustering results sigma = 0.2, num(eigs) = 2
Spectral clustering results sigma = 0.2, num(eigs) = 4
Spectral clustering results sigma = 0.2, num(eigs) = 8
Spectral clustering results sigma = 0.2, num(eigs) = 16
Spectral clustering results sigma = 0.5, num(eigs) = 2
Spectral clustering results sigma = 0.5, num(eigs) = 4
Spectral clustering results sigma = 0.5, num(eigs) = 8
Spectral clustering results sigma = 0.5, num(eigs) = 16
Spectral clustering results sigma = 1.0, num(eigs) = 2
Spectral clustering results sigma = 1.0, num(eigs) = 4
Spectral clustering results sigma = 1.0, num(eigs) = 8
Spectral clustering results sigma = 1.0, num(eigs) = 16

Fig. 1-3

Step 1: PCA is implemented with the function pca() on the training data by eigen decomposition of the centralized data to obtain an orthogonal X.

Step 2: Initialization P with: Computes conditional probability with the function x2p().

$$p(j|i) = \frac{A_{ij}}{\sum_{k\neq i} A_{ij}}, A_{ij} = exp(-\frac{|X_j - X_i|}{2\sigma_i^2})$$

then symmetrize it by $p(ij) = (p(j|i) + p(i|j)/2$.

In the above function, an optimal scale σ_i for each training example is required so that it reaches the ideal perplexity log(30). It is accomplished by binary search for each observation, the function Hbeta() is called to calculate the perplexity score.

Step 3: Initialize $Y^{(0)}$. Then, iteratively in each training step:

-Calculate Q according to $Y^{(t-1)}$

-Update$Y^{(t)}$ by minimizing the KL divergence between P and Q

During the training, momentum is set to be 0.8 in the first 20 steps and 0.5 later. Learning rate is adaptive according to the descending direction, if the new gradient is in a different direction from the last one, the learning rate affected by 'gain' is bigger.

(b)

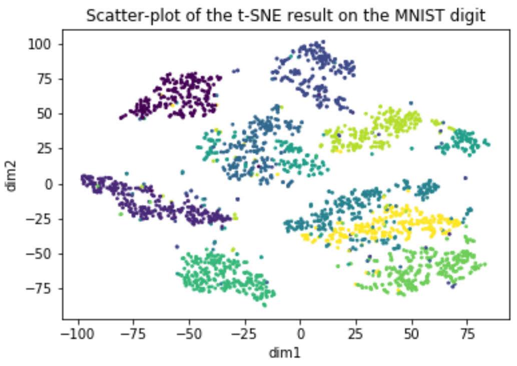

Fig. 3 - Scatter plot of the t-SNE result on the MNIST digits

Problem 3. Neural Network

(a, b, c, d)

Details are included in the .py file. The errors are smaller than $10^(- 7)$, detailed are included in the .py file. Also, it can be observed that posing regularization of 0.7 does not make much differences.

(e, f)

```
(Epoch 1 / 9) train acc: 0.936000; val_acc: 0.951000
(Epoch 2 / 9) train acc: 0.943000; val_acc: 0.958000
(Epoch 3 / 9) train acc: 0.958000; val_acc: 0.966000
(Epoch 4 / 9) train acc: 0.971000; val_acc: 0.968000
```

```
(Epoch 5 / 9) train acc: 0.960000; val_acc: 0.972000
(Epoch 6 / 9) train acc: 0.972000; val_acc: 0.976000
(Epoch 7 / 9) train acc: 0.974000; val_acc: 0.978000
(Epoch 8 / 9) train acc: 0.971000; val_acc: 0.979000
(Epoch 9 / 9) train acc: 0.967000; val_acc: 0.981000
```

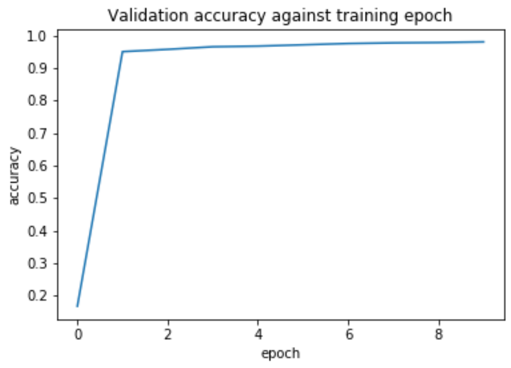

Fig. 3 - Validation accuracy against training epoch

(g)

The structure of the two-layer model is a fully-connected layer linearly mapping image input X intensities to a 100-dimensional hidden layer S, followed by a relu layer of element-wise nonlinear transformation to hidden layer H, then a fully-connected layer maps H to a 9-dimensional score layer, which is then used for classification decision by a softmax layer acting as the loss.

$$S = W_1 X + b_1, H = f_{relu}(S)$$

$$Score = W_2 H + b_2, P(Y = j) = \frac{e^{Score^\top}}{\sum_{k=1}^{K} e^{Score^\top}}$$

The learning algorithm employees gradient descent through back-propagation. The building blocks of the program utilize forward and backward functions for different type of layers, the former is the designed model and the later calculates corresponding gradients. Back-propogation based on chain rule is used in the gradient calculation. Especially, softmax may encounter logarithm over 0 which produces overflow, so the log-sum-exp trick can be used. After building the models a separate class "solver" is used to define the training process, the training is powered by update rules, the most fundamental is SGD. In the training process, loss and accuracy can be printed to monitor the process If convergence is slow or can not reach the gloabal minimum, the learning late and weight scale can be tuned.

Problem 4. XGB

(a, b)

The grid scores are:

```
mean: 0.86725, std: 0.00381, params: {'max_depth': 3, 'min_child_weight': 1},
mean: 0.86771, std: 0.00339, params: {'max_depth': 3, 'min_child_weight': 3},
mean: 0.86781, std: 0.00397, params: {'max_depth': 3, 'min_child_weight': 5},
mean: 0.86224, std: 0.00261, params: {'max_depth': 5, 'min_child_weight': 1},
mean: 0.86118, std: 0.00277, params: {'max_depth': 5, 'min_child_weight': 3},
mean: 0.86228, std: 0.00249, params: {'max_depth': 5, 'min_child_weight': 5},
mean: 0.85522, std: 0.00323, params: {'max_depth': 7, 'min_child_weight': 1},
mean: 0.85664, std: 0.00356, params: {'max_depth': 7, 'min_child_weight': 3},
mean: 0.85621, std: 0.00293, params: {'max_depth': 7, 'min_child_weight': 5}
```

(c) The feature importance plot is displayed in Figure 4.

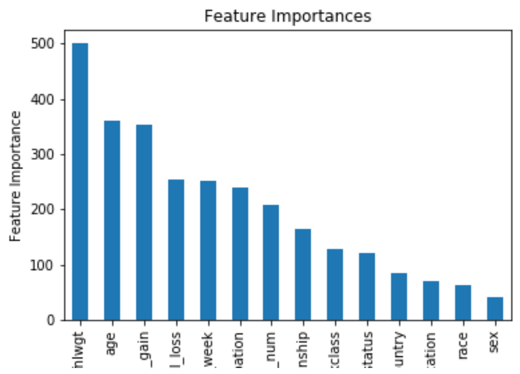

Fig. 4 - Feature importances in descending order

(d, e) Briefly discuss the method of extreme gradient boosting and the steps performed in the provided code.

XGBoost improves on gradient boosting algorithm by using an adaptive learning rate, derived by second order Taylor expansion.

The codes is implemented with an open-source software library ”XGBoost” for Python. Firstly, the data is cleaned by dropping NAs and transforming all categorical variables with binary dummy variables.

Then parameters sets are specified to perform cross-validation on a grid of parameters. The parameters to be tuned are booster parameters ”max depth” and ”min child weight”. The former is the maximum depth of a tree, XGB splits up to the specified value and then start pruning the tree backwards and remove splits with positive gain, which enables the learning to go deeper than ordinary GB that stops as it reach a negative gain. The later defines the minimum sum of weights of all observations required in a child to perform regularization, as a higher value prevents a model from learning relations highly specific to the particular sample.

Other booster parameters fixed in cv are ”eta” the learning rate to shrink the weights on each step, ”subsample” the proportion of observations to be randomly samples for each tree, and ”colsample bytree” the proportion of columns to be randomly samples for each tree. The ”objective” parameter defines the loss function as logistic regression for binary classification, returning predicted probabilities.

Cross-validation is implemented using sklearn's Grid Search over a grid of the cv parameters while fixing other parameter, to choose a set of parameters with the highest accuracy. The grid scores printed above suggests that the optimal parameters found by cross-validation is max-depth=3, min-child-weight=5, which achieves a cv accuracy as 0.868. Finally, the XGB model is trained with the chosen parameters. The function take as input the specified booster parameters, training data organized in a .Dmatrix format, and a specified number of boosting rounds.