**CMPE462 Project 2 Report**

1. **Introduction**
   1. **Decision Trees**
   2. **Support Vector Machines**

On svm, we designed and developed quadratic programming solver for primal and dual formulation of SVM by using one-vs-all strategy. At first, we designed it for the basic 2 classifier examples as course’s svm slide. You can follow these examples from basic\_quadratic\_solver.py, and basic\_dual\_quadratic\_solver.py files. Thereafter, we develop each of the step of assignments in a different classes. We commented every step, with their logic about why we build matrices of solver as like that etc. You can check LinearQuadraticSolver.set\_QP\_matrices and NonLinearQuadraticSolver.arrange\_matrices methods for these explanations.   
At the end we merged every single classes within a one class named svm.py who has 5 classes named LinearQuadraticSolver, ScikitLinearKernel, NonLinearQuadraticSolver, ScikitNonLinearKernel and MainClass.  
You can do every step by changing the constructor parameters of the first four classes in MainClass.run\_code method. You can check the commented out lines to do that, it’s explained in a detail. When you run the svm.py class, you should select an option to which step would you want to implement. You should enter   
**1:** for SVM from scratch using a quadratic programming solver

**2:** scikit-learn’s soft margin primal SVM function with linear kernel

**3:** for dual formulation of SVM from scratch using a quadratic programming solver

**4:** for scikit-learn’s soft margin dual SVM function with a non-linear kernel

You can change the parameters easily when calling their classes from their constructors by following the detailed explanation in MainClass.run\_code method.

* 1. **Clustering**

On the clustering.py file, we used k-means clustering algorithm with MNIST data. There are three different types of outputs. First one is Euclidean distance, second one is cosine similarity and the last one is Euclidean distance after extracted features with PCA. We made normalization before the algorithm.

1. **Decision Trees (30 pts)**
   1. **a**
   2. **b**
   3. **c**
2. **Support Vector Machines (40 pts)**
   * 1. **Please train a 4-class linear SVM using one-vs-all. Please train the primal formulation of SVM from scratch using a quadratic programming solver. Please clearly write the expressions you feed to the solver. Please tune the hyperparameters and report your training and test accuracy:**We feed the cvxopt’s solvers.qp(Q, p, G, h) you can follow the more detailed explanations in LinearQuadraticSolver. set\_QP\_matrices method. Simply:  
        Q is the zero matrix where it has identity matrix \* (1+regularization\_param) in its lower right  
        p is the zero matrix with size number of features + 1  
        A, c matrices are for ensuring the y \* (w^T \* x + b) >= 1  
        matrix A’s size is n\_samples, n\_features + 1,   
        it’s first column is the train labels, the rest is y\_i \* x\_i^T   
        c matrix has number of ones as n\_samples.  
        G and h matrices are negative values of A,c because in cvxopt.solvers, it solves the equation as Gx<=h but we constructed it as Au>=c. Therefore we multiply the matrices with -1  
        We tried different C (regularization parameters and C=0.1 gives the best result) So, our hyper param is C =0.1

Test Accuracy: 0.915527950310559

Train Accuracy: 0.9220143162124828

Process completed in 34.02 seconds.

* + 1. **Please train a 4-class SVM using the scikit-learn’s soft margin primal SVM function with linear kernel. Please tune the hyperparameters and report your training and test accuracy. Compare the results with part (a) regarding classification accuracy and training time.**We found 1 as a best C (regularization param) when we try different C values 0.01, 0.1, 1, 10, 100 with scikit.GridSearchCV.Training Accuracy: 0.9545397463267613

Testing Accuracy: 0.9438509316770186

Process completed in 3.10 seconds  
It has higher accuracy regarding to part a, and it’s completed faster than that because it uses one of the most optimized algorithms unlike we designed.

* + 1. **Please train a 4-class non-linear SVM using one-vs-all. Please train the dual formulation of SVM from scratch using a quadratic programming solver. Please clearly write the expressions you feed to the solver. You may choose any kernel you like. Please tune the hyperparameters and report your training and test accuracy.***Warning*: We used smaller part of the test and dataset because it compiles to many times (some of our friends say more than 10 hourst). Therefore we used MainClass.make\_smaller\_sets methods which takes first 1000 of train images, and 100 of test images. In that way you can easily see the result within seconds. But because we don’t have to many data to train our model, its accuracy become a little lower than the others. But we think that’s not a problem comparing to training the model more than hours.  
       You can follow the more detailed explanations in NonLinearQuadraticSolver.arrange\_matrices method  
       We used solution = solvers.qp(P, q, G, h, A, b) where  
       P: yi\*yj\*K(xi,xj) (K is for the kernel)  
       q: column vector of minus ones for maximizing ∑α(i) which means minimize -∑α(i)  
       G: identity matrix for making C>=a(i)>=0  
       h: zeros for making a(i)>=0, C values for making C>=a(i), where C is the regularization parameter  
       A: row vector with labels  
       b: scalar 0 for ∑α(i)y(i) = 0 's right handside

We picked rbf kernel, because it’s the most suitable kernel for our design when we tried rbf, polynomial and sigmoid kernels.  
We also arranged gamma=0.1 and regularization\_param = 1 because they give the best score.

Train Labels: {9: 254, 2: 258, 3: 254, 8: 230}  
Test Labels: {2: 25, 9: 30, 3: 23, 8: 18}  
Testing Accuracy: 0.87

Training Accuracy: 0.998

Process completed in 24.41 seconds.

* + 1. **Please train a 4-class SVM using the scikit-learn’s soft margin dual SVM function with a non-linear kernel. You may choose any kernel you like. Please tune the hyperparameters and report your training and test accuracy. Compare the results with part (c) regarding classification accuracy and training time.**We picked the best parameters as {'C': 10, 'kernel': 'rbf', ‘gamma’:0.1} by find\_best\_params = True when we call ScikitNonLinearKernel object. But because it takes time (more than 5 min) we make it False by default

Testing Accuracy: 0.9873291925465838

Training Accuracy: 0.9933860772740591

Process completed in 51.16 seconds.

It has higher accuracy and very few training time comparing to part c because it uses the best algorithms.

* 1. **Please extract features from the images. You may try any feature extraction technique you like. However, please explain the reason behind your choice. Repeat the experiments in 1. a-d with the extracted features and compare the performance in terms of accuracy and training time.**We used sklearn.PCA technique because it generates uncorrelated features which can improve the performance and training speed of machine learning models, making it a versatile and efficient choice for general feature extraction tasks. On the other hand, it’s suitable for our dataset for part a,c because when we tried other techniques like hog, LDA, TSNE it gives an error when we implementing the cvxopt.solver. Therefore we picked PCA.  
     You need to make our object’s constructor parameter is\_feature\_extract=True to implement it as we explained in the code. When we applied PCA, the training and predicting time decreases, but the accuracy score also decreases in general because important features are lost during the transformation. However in part d it increases the accuracy it may remove noise by retaining the components that capture the most variance in the data.
     1. PCA Process completed in 1.05 seconds.

Test Accuracy: 0.8094409937888198

Train Accuracy: 0.8038846330947298

Process completed in 1.42 seconds.

* + 1. PCA Process completed in 1.26 seconds.

Training Accuracy: 0.9334840303068358

Testing Accuracy: 0.9381366459627329

Process completed in 0.27 seconds.

* + 1. PCA Process completed in 0.14 seconds.

Testing Accuracy: 0.87

Training Accuracy: 0.968

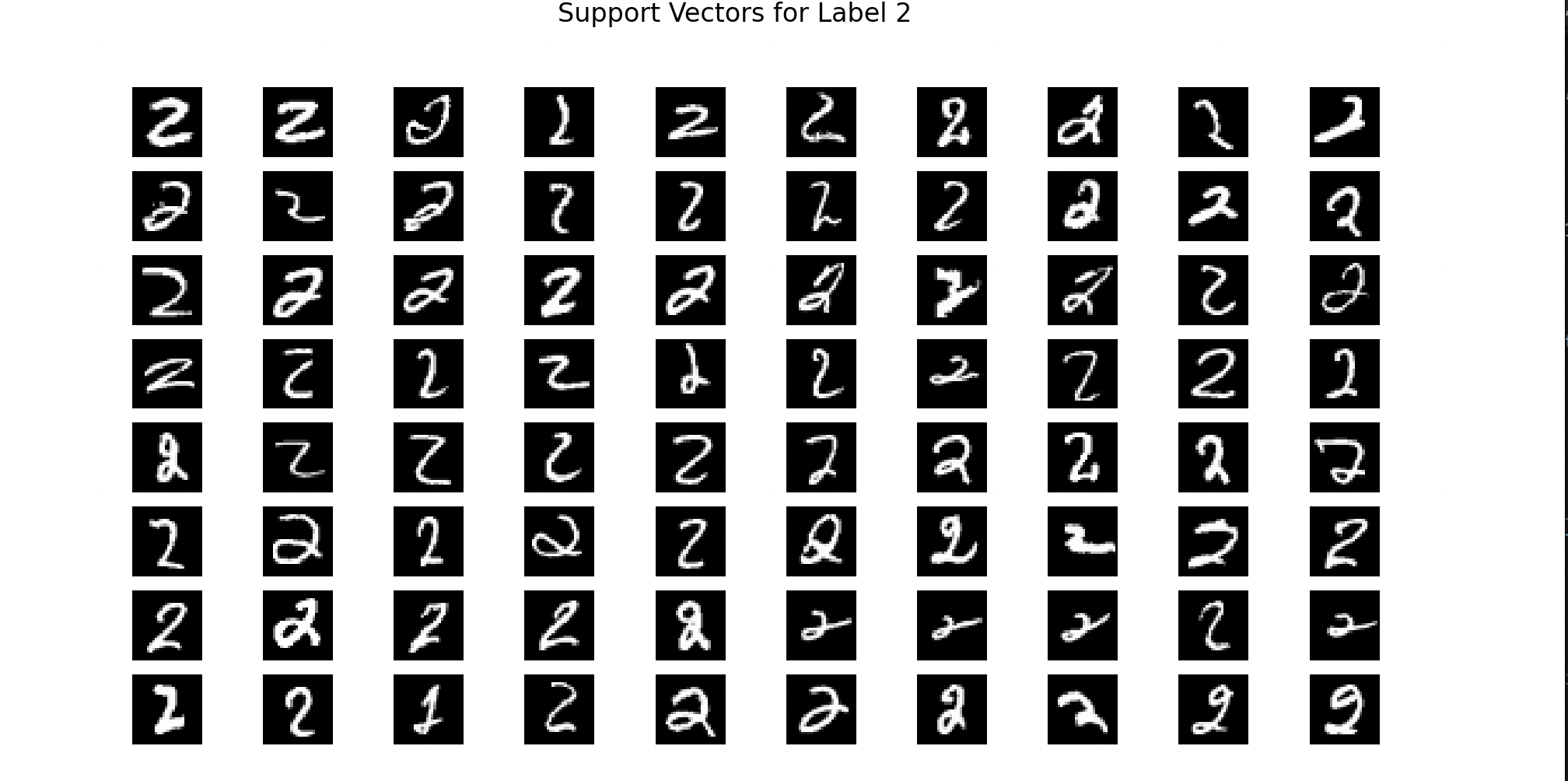
Process completed in 22.35 seconds.

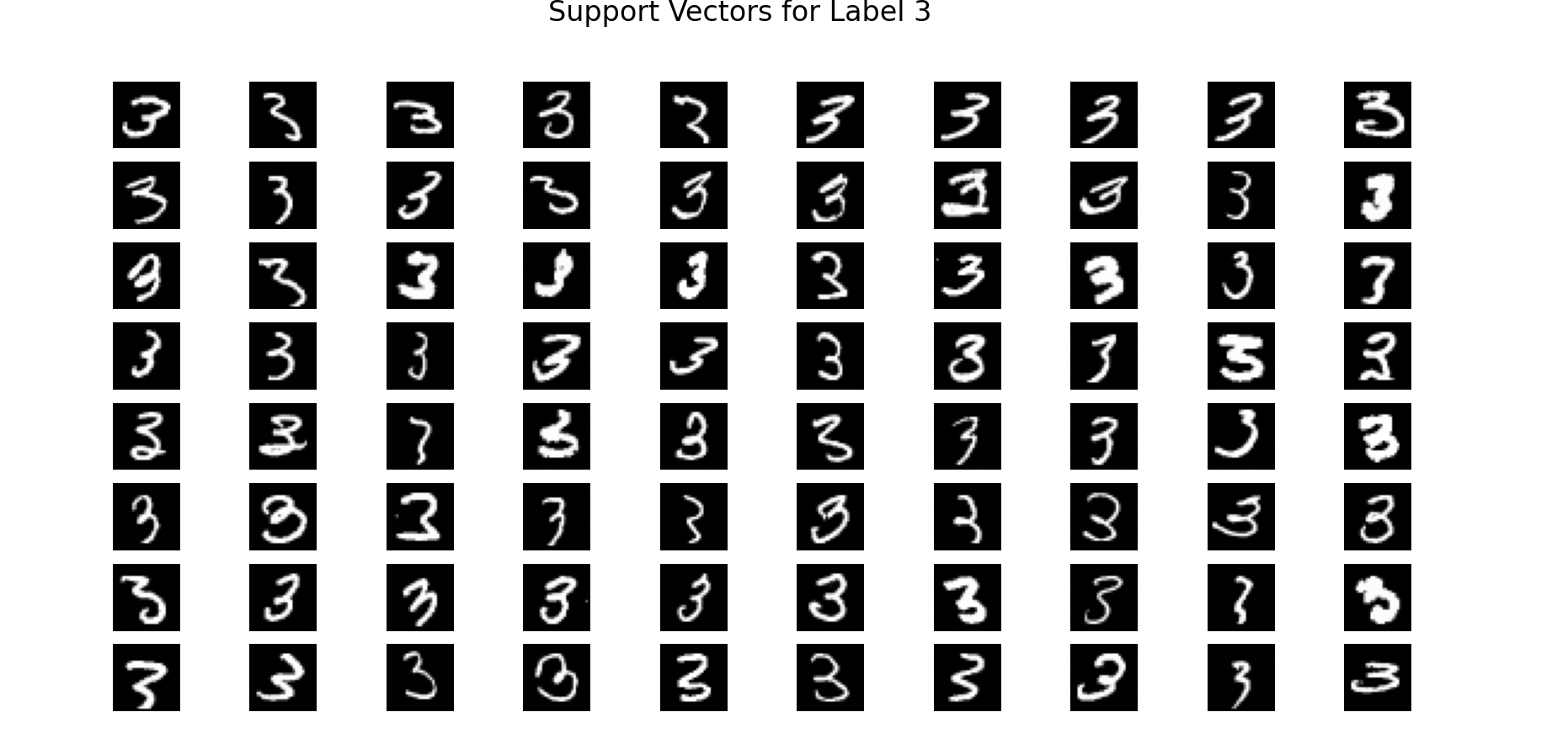
* + 1. PCA Process completed in 1.07 seconds.

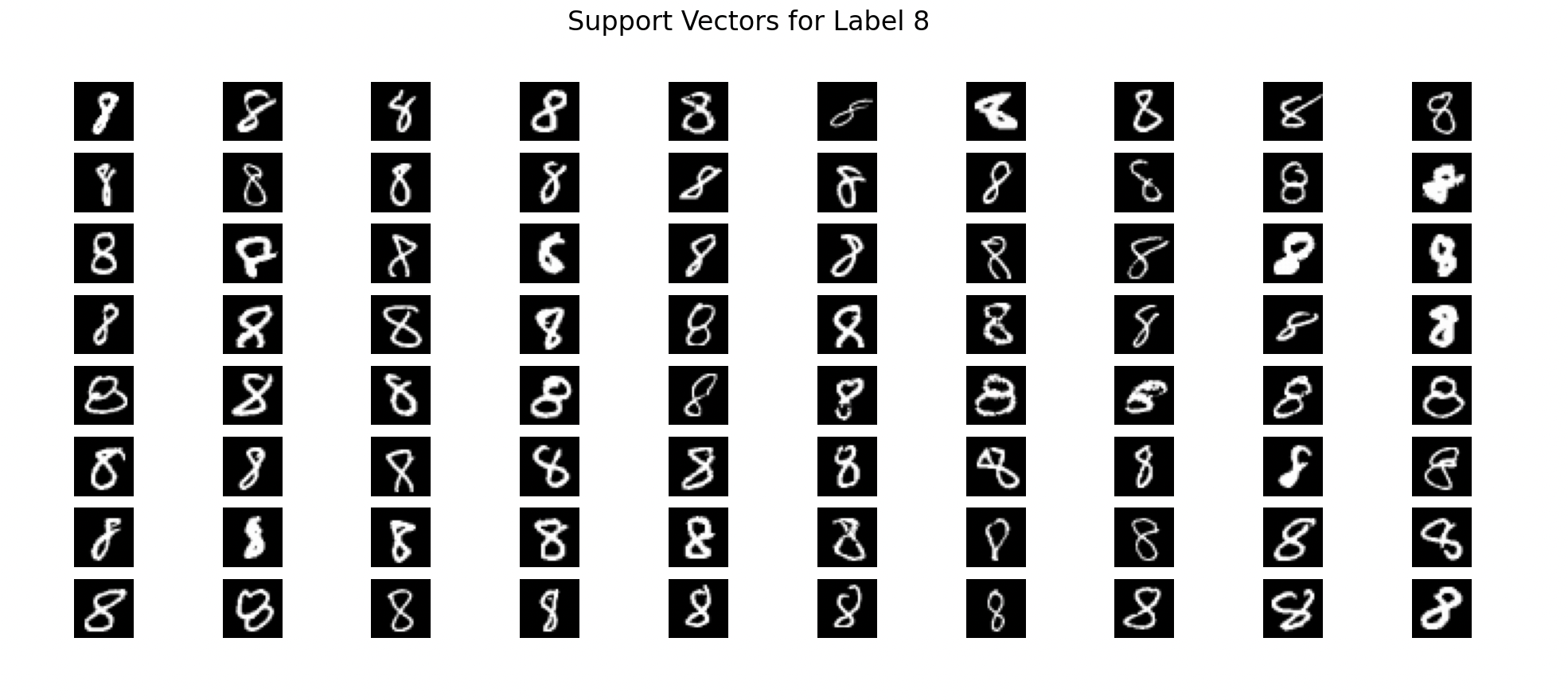
Testing Accuracy: 0.9915527950310559

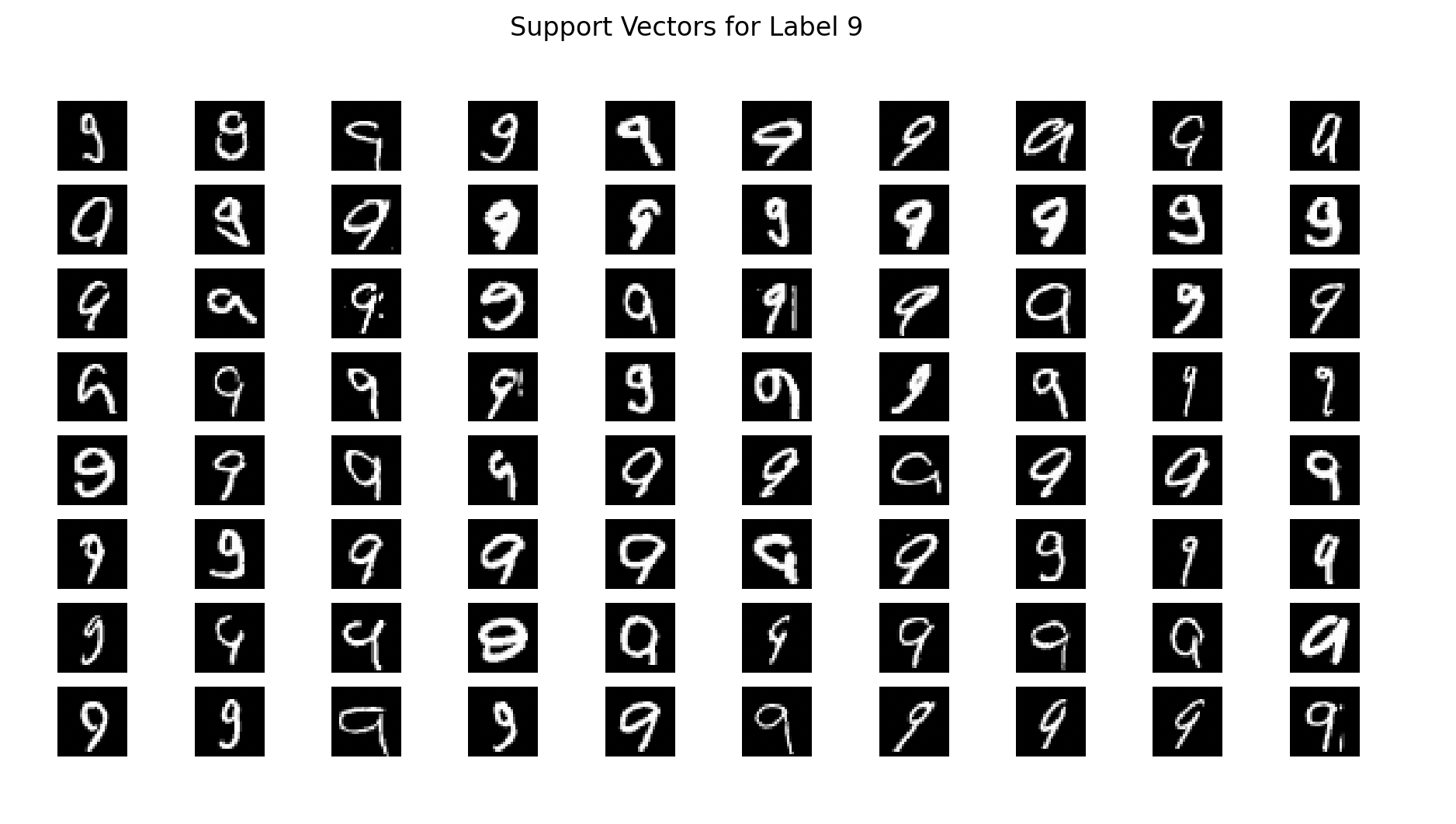
Training Accuracy: 0.9949767675499184

Process completed in 8.31 seconds.

* 1. **Please find the support vectors using one of the dual SVM models you trained and inspect the images. Please discuss whether there is any visual difference between the support vectors and other images.**You should make ScikitNonLinearKernel’s constructor is\_feature\_extract=False, display\_support\_vector=True to see the support vectors. Here are the images of our support vectors in matplotlib  
     





  
The support vectors seem more variable, noisier and less distinct compared to typical examples of each handwritten images, they reflecting the challenging cases that the model must learn to distinguish correctly. Normal handwritten images would more cleaner, consistent and uniform unlike the support vectors.

1. **Clustering (30 pts)**
   1. **a**

All the features are the pixel values from 0 to 255 and they are all the same unit. So each feature contributes equally to the distance computations. Normalizing affects only sse values not the accuracy.

* 1. **b**

Accuracy of euclidean: 0.7729917535267278

SSE of euclidean: 1042452.1736352231

Accuracy of euclidean features extracted 0.7736615178534053

SSE of euclidean features extracted 816375.7755159766

Although accuracy values are similar to each other, after features are extracted with pca, sse decreases compared to the original one. This means that clusters are more compact and well-separated after the features are extracted.

* 1. **c**

Accuracy of euclidean: 0.7685964251329064

SSE of euclidean: 1042456.6991974805

Accuracy of cosine similarity: 0.7428104985558207

SSE of cosine similarity: 1043865.8799700012

There is not much difference between accuracies, they are usually between 73-77 for both euclidean distance and cosine similarity. Also sse values are really close to each other. Therefore, there is not much difference in results between euclidean distance and cosine similarity methods.