CMPE 462 ASSIGNMENT 2 REPORT

Osman Yasin Baştuğ/ 2021400021

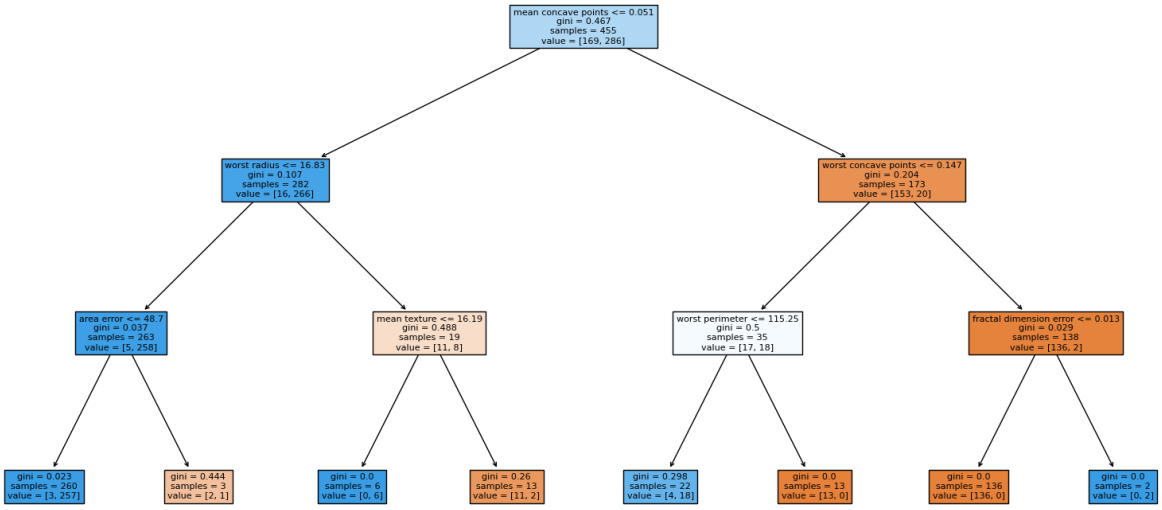
Github link of the project repo: https://github.com/yasinbastug/cmpe462_p2

There is one notebook for SVMs and another one for the others.

1. Decision Trees

1.1

max depth: 3



Accuracy: 0.9473684210526315

1.2 Compare

Naive Bayes yields to a 1.0 accuracy :0, I didn't implement this Naive Bayes my former teammate did, but no errors was in my sight. This maybe caused because the Naive Bayes takes all the features into account when trees only uses a few.

1.3 Feature Selection

Accuracy with 5 features: 0.956140350877193
Accuracy with 10 features: 0.956140350877193
Accuracy with 15 features: 0.9824561403508771
Accuracy with 20 features: 0.9912280701754386

•

Feature Importance: The initial set of features provides a solid baseline accuracy, but additional features continue to add value, improving the model's performance significantly.

1.4 Random Forest

- **Effectiveness of Increasing Trees**:
  - Increasing the number of trees in a Random Forest improves accuracy up to a certain point. Accuracy improves up to 50 trees and then plateaus, indicating that 50 trees are sufficient to achieve high performance.
- **Diminishing Returns**:
  - Beyond 50 trees, there is no significant improvement in accuracy. This is common in ensemble methods where increasing the number of base learners eventually leads to minimal gains.

2. SVM

2.1.a

The primal optimization problem for SVM can be formulated as follows:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$$

subject to

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i$$

Where:

- w is the weight vector.
- b is the bias term.
- ξ are the slack variables.
- C is the regularization parameter.
- yi are the labels.
- xi are the input features.

$$\xi_i \geq 0 \quad \forall i$$

the matrices are set up for the primal problem:

- **Q Matrix:**

$$Q = \begin{bmatrix} 0 & 0^T & 0^T \\ 0 & I & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Represents the quadratic part of the objective function.

Where I is the identity matrix.

- **p Vector**:

Represents the linear part of the objective function.

$$p = \begin{bmatrix} 0 \\ 0 \\ C \end{bmatrix}$$

- **G Matrix**:

Represents the inequality constraints.

$$G = \begin{bmatrix} -y \cdot 1 & -y \cdot X & -I \\ 0 & 0 & -I \end{bmatrix}$$

- **h Vector**:

Represents the upper and lower bounds for $\mathcal{E}i$ and the G inequality constrains

y(Xw + b) > 1 - Ei and Ei ≥ 0

$$h = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

the result with all data

Penalty Value	Accuracy
0.01	0.950
0.1	0.949
1	0.947
10	0.944
100	0.944

Regularization in SVM

Regularization in SVM aims to prevent overfitting by penalizing overly complex models. It does this by balancing two objectives:

1. **Maximizing the Margin**: Ensuring that the decision boundary is as far away as possible from any data point of either class.
2. **Minimizing Classification Error**: Ensuring that the number of misclassified points is minimized.

The Role of *C*

The penalty parameter *C* controls the balance between these two objectives:

1. **High *C* (Low Regularization)**:
   - A high value of *C* places more emphasis on minimizing classification error.
   - The model will try to classify all training examples correctly, potentially at the cost of a smaller margin.
   - This can lead to overfitting, where the model captures noise in the training data, resulting in poor generalization to new data.
2. **Low *C* (High Regularization)**:
   - A low value of *C* places more emphasis on maximizing the margin.
   - The model allows some misclassifications but achieves a larger margin.
   - This can lead to better generalization, as the model is simpler and less sensitive to noise in the training data.

This is because the increasing penalty value causes the margin to be more strict since slack variables are penalized more. This causes the model to be more sensitive to the training data and less generalizable to the test data. Although the accuracies are good the computation took more than 60 hours. Hence I searched for the scikit's implementation of the SVM by reading the libsvm paper. I found that shrinking and caching is used to optimize the computation time. So I tried to

implement caching and shrinking, which still worked but did not optimize the time and increased it. Then I decided to do sampling instead of trying to optimize the computation time which also yielded in similar results for accuracy. The codes are in the notebook.

A peek from the quadratic solver's steps with the sampled 2500 data to see if it's working:

```
Training with penalty value: 0.01
     pcost       dcost       gap    pres   dres
 0:  1.7203e+00  1.7005e+02  1e+04  3e+00  2e+04
 1:  3.9163e+01 -2.8554e+02  3e+02  5e-02  3e+02
 2:  2.9244e+01 -3.6674e+01  7e+01  8e-03  5e+01
 3:  1.4164e+01 -8.4052e+00  2e+01  3e-03  1e+01
 4:  5.9967e+00 -1.9613e+00  8e+00  9e-04  5e+00
 5:  3.0410e+00 -1.1282e-01  3e+00  3e-04  1e+00
 6:  1.4206e+00  7.3606e-01  7e-01  5e-05  3e-01
 7:  1.1010e+00  9.1832e-01  2e-01  8e-06  5e-02
 8:  1.0147e+00  9.6845e-01  5e-02  1e-06  8e-03
 9:  9.8951e-01  9.8422e-01  5e-03  4e-08  2e-04
10:  9.8661e-01  9.8642e-01  2e-04  1e-09  6e-06
11:  9.8651e-01  9.8650e-01  4e-06  2e-11  1e-07
12:  9.8650e-01  9.8650e-01  1e-07  5e-13  3e-09
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0:  2.0777e+00  1.9280e+02  1e+04  3e+00  1e+04
 1:  3.8908e+01 -3.2995e+02  4e+02  6e-02  3e+02
 2:  3.0344e+01 -3.7445e+01  7e+01  9e-03  4e+01
 3:  1.4793e+01 -7.5367e+00  2e+01  2e-03  1e+01
 4:  7.6596e+00 -2.1610e+00  1e+01  1e-03  5e+00
 5:  3.7385e+00  2.0909e-01  4e+00  3e-04  1e+00
 6:  1.9419e+00  1.1034e+00  8e-01  5e-05  3e-01
 7:  1.5440e+00  1.3279e+00  2e-01  2e-06  1e-02
 8:  1.4578e+00  1.3766e+00  8e-02  6e-07  3e-03
 9:  1.4131e+00  1.4034e+00  1e-02  2e-15  1e-13
10:  1.4079e+00  1.4074e+00  5e-04  2e-15  1e-13
11:  1.4077e+00  1.4077e+00  2e-05  2e-15  5e-13
12:  1.4077e+00  1.4077e+00  3e-07  2e-15  7e-12
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0:  2.1534e+00  1.9791e+02  1e+04  3e+00  1e+04
 1:  3.8486e+01 -3.7867e+02  4e+02  8e-02  3e+02
 2:  3.1049e+01 -4.5954e+01  8e+01  1e-02  4e+01
 3:  1.6280e+01 -9.2442e+00  3e+01  3e-03  1e+01
 4:  7.5939e+00 -1.6166e+00  9e+00  9e-04  4e+00
 5:  3.0091e+00  1.0370e+00  2e+00  2e-04  7e-01
 6:  2.1296e+00  1.5540e+00  6e-01  3e-05  1e-01
 7:  1.9098e+00  1.6778e+00  2e-01  1e-05  4e-02
 8:  1.7944e+00  1.7411e+00  5e-02  2e-15  4e-14
 9:  1.7693e+00  1.7590e+00  1e-02  2e-15  2e-13
10:  1.7637e+00  1.7631e+00  7e-04  2e-15  3e-13
11:  1.7634e+00  1.7633e+00  3e-05  2e-15  4e-13
12:  1.7634e+00  1.7634e+00  6e-07  2e-15  8e-12
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0:  1.3841e+00  1.4497e+02  1e+04  3e+00  2e+04
```

```
 1:  3.8141e+01 -3.4460e+02  4e+02  7e-02  4e+02
 2:  2.9349e+01 -3.7291e+01  7e+01  9e-03  5e+01
 3:  1.4252e+01 -8.6516e+00  2e+01  2e-03  1e+01
 4:  6.6771e+00 -2.6780e+00  9e+00  9e-04  5e+00
 5:  3.0923e+00 -5.2377e-01  4e+00  3e-04  2e+00
 6:  1.4180e+00  3.3089e-01  1e+00  7e-05  4e-01
 7:  9.1160e-01  5.8460e-01  3e-01  5e-06  3e-02
 8:  7.7303e-01  6.5092e-01  1e-01  1e-06  6e-03
 9:  7.1714e-01  6.8059e-01  4e-02  2e-07  9e-04
10:  6.9860e-01  6.9183e-01  7e-03  5e-09  3e-05
11:  6.9514e-01  6.9450e-01  6e-04  4e-10  2e-06
12:  6.9480e-01  6.9477e-01  4e-05  2e-11  1e-07
13:  6.9478e-01  6.9478e-01  5e-07  3e-13  2e-09
Optimal solution found.
Accuracy with penalty 0.01: 0.92
```

2.1.b

In the scikit explanation of the SVC method it is stated that
"The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets
consider using ~sklearn.svm.LinearSVC or
~sklearn.linear_model.SGDClassifier instead, possibly after a
~sklearn.kernel_approximation.Nystroem transformer or
other kernel_approximation."

So I used the LinearSVC class to train a linear SVM model with different penalty values. I used the same penalty values as in 2.1.a. I used the fit method to train the model and the score method to calculate the accuracy of the model on the test data. I printed the accuracy of the model for each penalty value.

The training time of the model is much faster than the quadratic optimization in the previous question. So to not get that 72 hour runtime like in the first question, I sampled the data for 10000 samples and the results are as follows:

Penalty Value	Accuracy
0.01	0.944
0.1	0.934
1	0.921
10	0.909

the same relationship between penalty value and accuracy is observed in this question as well.

2.1.c

Dual Optimization Problem

The dual optimization problem for SVM with the RBF kernel is formulated as follows:

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C$$

Where:

- α are the Lagrange multipliers.
- yi are the labels.
- $K(xi,xj)$ is the kernel function, in this case, the RBF kernel.
- C is the regularization parameter.

the matrices are set up for the dual problem:

- **P Matrix**:

$$P = \text{matrix}(\text{np.outer}(y\_binary, y\_binary) * K)$$

Represents the quadratic part of the objective function.

$$P_{ij} = y_i y_j K(x_i, x_j)$$

- **q Vector**:

$$q = \text{matrix}(-\text{np.ones}(N))$$

Represents the linear part of the objective function.

$$q_i = -1$$

- **G Matrix**:

$$G = \text{matrix}(\text{np.vstack}((-\text{np.eye}(N), \text{np.eye}(N))))$$

Represents the inequality constraints.
Ensures

$$0 \leq \alpha_i \leq C$$

- **A Matrix**:

$$A = \text{matrix}(y\_binary, (1, N), 'd')$$

Represents the equality constraint.
Ensures

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

- **b Scalar**:

$$b = \text{matrix}(0.0)$$

The bias term for the equality constraint.

$$\text{rbf\_kernel}(x1, x2, \gamma): \text{ Computes the RBF kernel value between two points } x1 \text{ and } x2 \text{ using the parameter } \gamma.$$

$$\text{compute\_kernel\_matrix}(X, \gamma): \text{ Computes the full kernel matrix } K \text{ for the dataset } X.$$

$$\text{train\_one\_vs\_all\_svm}(X, y, \text{class\_labels}, C, \gamma): \text{ Trains an SVM classifier for each class using the one-vs-all strategy}$$

- **Optimal Configuration**: The combination of a high penalty (`C = 10` or `C = 100`) and a low gamma (`γ = 0.01`) provides the best accuracy (0.792). This indicates that a strong regularization (high `C`) with a less sensitive kernel (low `γ`) effectively balances the bias-variance tradeoff, leading to better generalization.
- **Overfitting**: Higher gamma values (0.05 and 0.1) lead to overfitting regardless of the penalty value, resulting in poor accuracy. The RBF kernel becomes too sensitive to individual data points, causing the model to capture noise rather than the underlying patterns.
- **Regularization Effect**: Increasing the penalty value generally improves accuracy up to a certain point (C = 1 to C = 10), but beyond that, the improvement is marginal. This suggests that there is an optimal range for the penalty parameter beyond which the benefits diminish.

2.1.d

Best parameters: {'C': 10, 'gamma': 0.01}
Training accuracy: 1.0
Test accuracy: 0.758

The best parameters are the same but the one from scratch yields to a better accuracy.

2.2

1.a with sample size 2500

Penalty Value	Accuracy without PCA	Accuracy with PCA
0.01	0.92	0.92
0.1	0.91	0.91
1	0.90	0.90
10	0.90	0.88
100	0.90	0.86

1.b with sample size 2500

0.01	0.92	0.90
0.1	0.90	0.89
1	0.90	0.88
10	0.89	0.86
100	0.89	0.84

1.c with sample size 2500

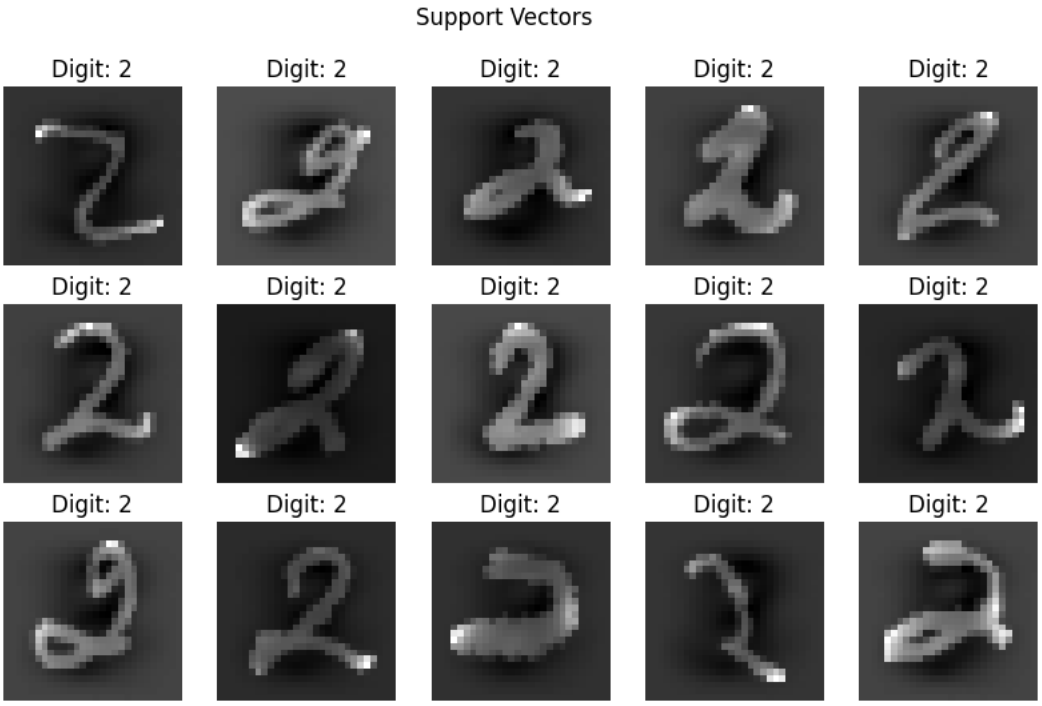
(`C = 10`) and a low gamma (`γ = 0.01`) again provides the best accuracy (0.81)

1.d with sample size 2500

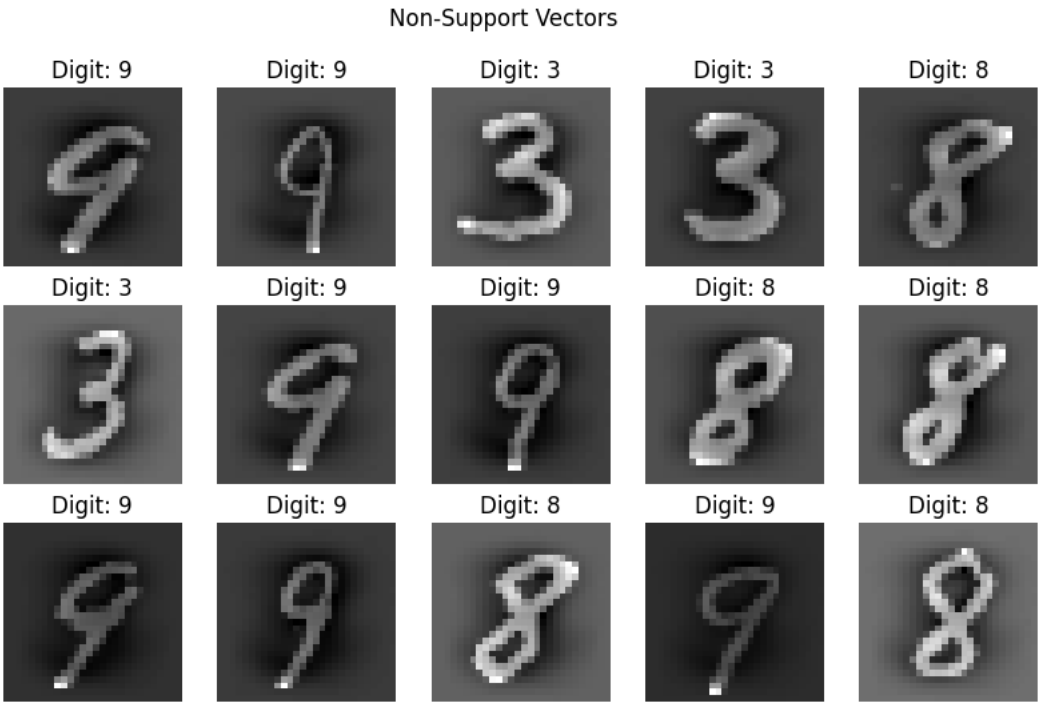
(`C = 10`) and a low gamma (`γ = 0.01`) again provides the best accuracy (0.79)

2.3

- **Support Vectors**:
  - Support vectors are the critical data points that influence the decision boundary of the SVM model. These vectors typically lie close to the boundary and are often more ambiguous or representative of edge cases.



- **Non-Support Vectors**:
  - Non-support vectors are usually more typical examples of each class and do not directly influence the decision boundary.



We also see here that we cannot identify the numbers in the Support Vectors as we can in the non-Support Vector ones.

3. CLUSTERING

3.1 Importance of Normalizing Data Points Before Running K-means

Normalizing data before running k-means clustering is crucial due to the nature of how k-means works. K-means clustering algorithm uses Euclidean distance to assign data points to clusters. If the features in the dataset are on different scales, the feature with a larger scale will dominate the distance calculation, potentially leading to biased or incorrect clustering. Normalization (such as scaling all features to have zero mean and unit variance or to fit within a specific range like 0 to 1) ensures that each feature contributes equally to the distance calculations, leading to more meaningful clusters.

3.2 Feature Extraction Effect

Euclidean

Clustering Accuracy: 0.7699004083972201
SSE: 1216249.2083248035

Clustering Accuracy with PCA: 0.7767070287311026
SSE with PCA: 1144197.7749010248

Cosine

Clustering Accuracy: 0.7484416421867164
SSE: 1217835.3655996728

Clustering Accuracy with PCA: 0.78125671705954
SSE with PCA: 1144577.8856176261

The clustering accuracy has slightly improved after applying PCA. This suggests that PCA has helped in reducing noise or removing irrelevant features, thereby enhancing the clustering performance.

SSE has decreased after applying PCA, indicating that the clusters are more compact. Lower SSE values indicate that data points are closer to their respective cluster centroids.

3.3 Difference in the Clustering Results

Observations:

- **Euclidean Distance**: This distance measure, which calculates the straight-line distance between two points in Euclidean space, resulted in a higher clustering accuracy. With a value close to 77%, this indicates a relatively strong alignment between the clusters formed and the actual categories of the dataset (digits 2, 3, 8, and 9 in this case). This might suggest that in terms of absolute spatial distances, the data points belonging to the same digit tend to cluster closely.
- **Cosine Distance**: Cosine distance, which measures the cosine of the angle between two vectors (thus is more about orientation than magnitude), showed a slightly lower accuracy of approximately 74.63%. This suggests that when the model is oriented towards identifying the direction rather than the magnitude of data points, there is a slight decrease in the effectiveness of clustering according to the true labels.

Analysis:
The observed difference in clustering accuracy, although not drastic, is statistically significant and implies a few key insights:

- **Data Characteristics**: Since MNIST data represents image pixels, the magnitude (represented by Euclidean distance) might carry more meaningful information about differences between digits than the orientation (represented by cosine distance). This could be due to the way digits are structured spatially in terms of pixel intensity.
- **Clustering Dynamics**: The clustering process with Euclidean distance might be more sensitive to the varied densities or distributions of points within the same digit category, thus capturing more accurately the inherent groups within the data.

References

https://pythonprogramming.net/soft-margin-kernel-cvxopt-svm-machine-learning-tutorial/

https://medium.com/@ahlawat.randeep/svm-from-scratch-using-quadratic-programming-90b4dbc5e1d2

ChatGPT, copilot