

# SWE 582: Sp. Tp. Machine Learning for Data Analytic

## Assignment 2 – Support Vector Machines and K-Means Clustering

Feridun Berk Akyol

ID: 2022719192

Date: 26.05.2024

### 1- Support Vector Machines (SVM)

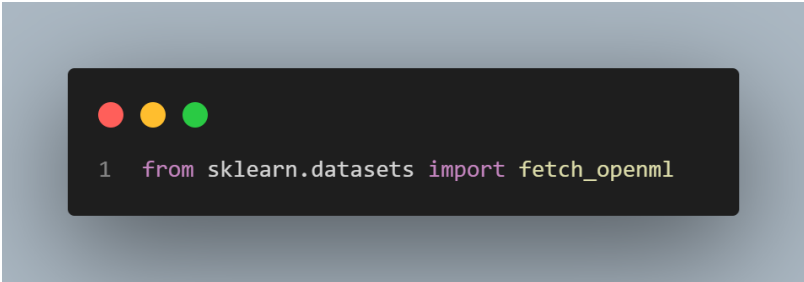
- **MNIST Database**

MNIST is a database of handwritten digits that is used for image processing.

In our assignment we will be using digits 2, 3, 8 and 9 which makes total number of 20000 samples for training set and 4000 samples for test set. Those 4 digits will be classes of our 4-class SVM.

We will also flatten the gray scale images.

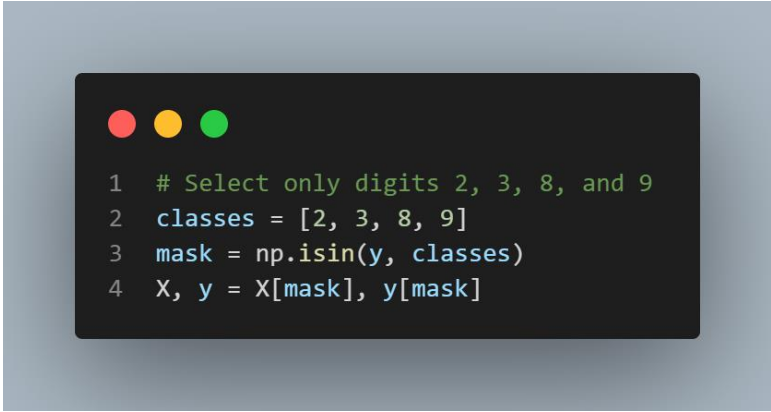
We import the mnist data from openml using sklearn(sciikit-learn)



```
1 from sklearn.datasets import fetch_openml
```

- **Preparing the data for training**

We will be training our SVM with 4 classes as stated above. So we only need the image data for those 4 digits



```
1 # Select only digits 2, 3, 8, and 9
2 classes = [2, 3, 8, 9]
3 mask = np.isin(y, classes)
4 X, y = X[mask], y[mask]
```

- **4-Class SVM with linear kernel.**

We will be starting by training a linear SVM mode using our dataset.

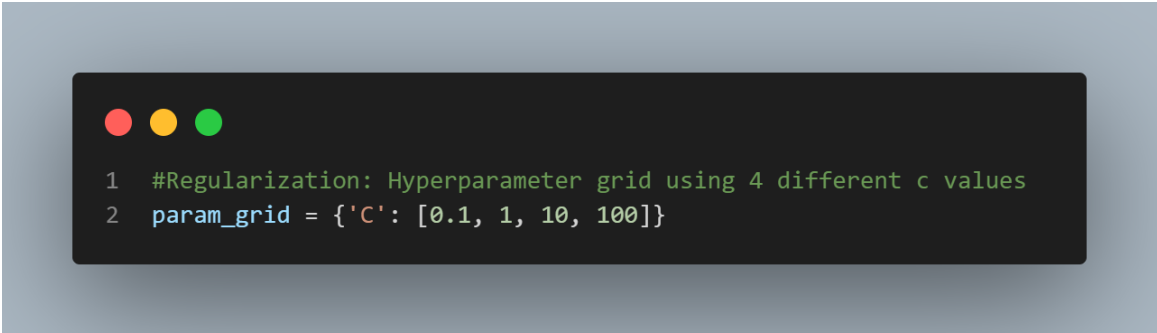
We will be using 5-fold cross validation technique to assess the performance of our model by validating it on five equally divided subsets of the data.

**Hyperparameters** are defined in the “C” matrix. The hyperparameter will determine the balance between training error and testing error. So we will be choosing the best c value corresponding with the best regularization result. Thus, preventing the overfitting of our model.

**Regularization:** “C” helps in regularizing the model. Regularization prevents overfitting by penalizing overly complex models that fit the training data too closely.

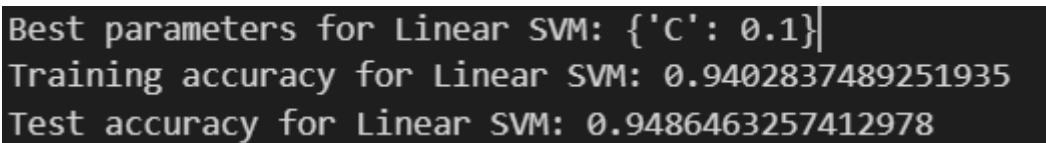
**Generalization:** By tuning “C”, we aim to find a balance that allows the model to generalize well to new, unseen data rather than just memorizing the training data.

We will be choosing the best possible parameter out of the 4 initial parameters we defined as follows.



```
1 #Regularization: Hyperparameter grid using 4 different c values
2 param_grid = {'C': [0.1, 1, 10, 100]}
```

#### Results for linear SVM.



```
Best parameters for Linear SVM: {'C': 0.1}
Training accuracy for Linear SVM: 0.9402837489251935
Test accuracy for Linear SVM: 0.9486463257412978
```

Screenshot of our training results. The smallest value for C turned out to be the best hyperparameter for our model.

**Small C value:** A small value of C means the SVM will have a wider margin and allow more misclassifications in the training data. This can lead to a simpler model that might underfit the data.

**Large C value:** A large value of C means the SVM will try to classify all training examples correctly, leading to a smaller margin and a more complex model. This can lead to overfitting the data.

- **4-Class SVM with non-linear kernel**

We will be using the Radial Basis Function (RBF) or Gaussian Kernel in our non-linear model.

The SVM with RBF is more accurate compared to linear SVM but its also computationally more demanding and time consuming.

We will be adding the gamma hyperparameter to our model.

**gamma** is a hyperparameter specific to certain kernel functions in SVMs, particularly RBF kernel. It defines how much influence a single training example has. The larger the gamma, the closer other examples must be to be affected.

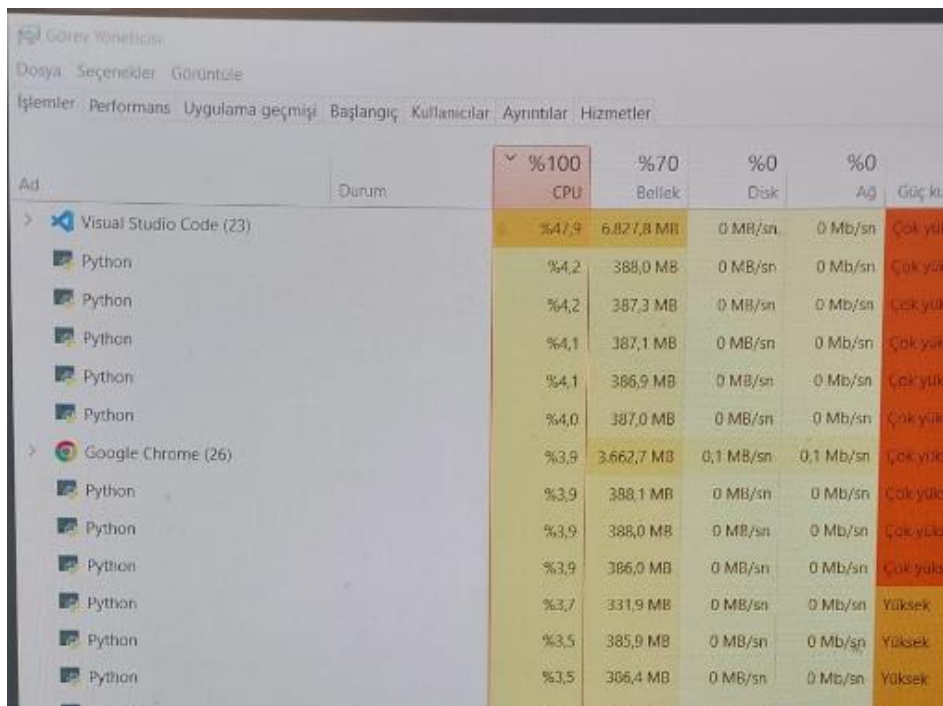
```
1 # Hyperparameter grid for RBF kernel
2 param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1]}
```

### Some problems I have encountered with the SVM RBF and how to deal with it?

The SVM with RBF was significantly slower compared to linear SVM as expected. After waiting for nearly an hour I decided to take mater in my own hands.

### Multithreading

The python was only using a single thread while running the code. I have a 12 core CPU I decided to utilize all my cores to run the code. Using all the 12 cores was a failed attempt and resulted in my computer freezing.



The screenshot shows the Windows Task Manager Performance tab. The 'CPU' column is highlighted in red, indicating 100% usage. The 'Memory' column shows usage around 70-80%. The 'Disk' and 'Network' columns show 0% usage. The 'Power' column shows 'Çok yük' (Very high) for most processes. The list of processes includes Visual Studio Code (23) and Google Chrome (26), both with multiple Python processes running under them.

Ad	Durum	%100 CPU	%70 Bellek	%0 Disk	%0 Ağ	Güç ku
Visual Studio Code (23)		%47,9	6.827,8 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%4,2	388,0 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%4,2	387,3 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%4,1	387,1 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%4,1	386,9 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%4,0	387,0 MB	0 MB/sn	0 Mb/sn	Çok yük
Google Chrome (26)		%3,9	3.662,7 MB	0,1 MB/sn	0,1 Mb/sn	Çok yük
Python		%3,9	388,1 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%3,9	388,0 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%3,9	386,0 MB	0 MB/sn	0 Mb/sn	Çok yük
Python		%3,7	331,9 MB	0 MB/sn	0 Mb/sn	Yüksek
Python		%3,5	385,9 MB	0 MB/sn	0 Mb/sn	Yüksek
Python		%3,5	386,4 MB	0 MB/sn	0 Mb/sn	Yüksek

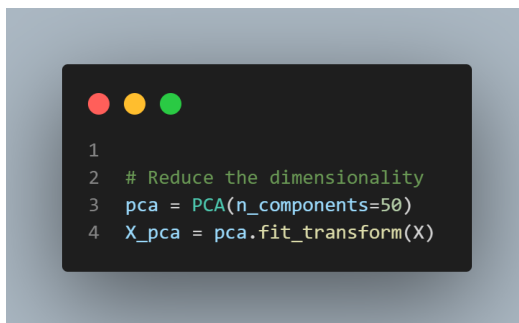
I ended up using 6 cores to run the code. Although it helped it was still not fast enough and took forever to train the model

### GPU Acceleration

I thought about using the GPU rather than CPU to train the model since GPUs are better at parallel computing. It turns out that unlike TensorFlow scikit\*learn doesn't support the GPU acceleration so it wasn't an option.

### Principal Component Analysis (PCA)

Finally, I decided to use a dimensionality reduction to reduce the number of features in dataset while keeping as much information as possible.



Finally, I have managed to decrease the training time around 10 minutes mark.

### Results of SVM with RBF after PCA and Multithreading

```
Best parameters for RBF kernel: {'C': 10, 'gamma': 0.01}  
Training accuracy for RBF kernel: 0.989767841788478  
Test accuracy for RBF kernel: 0.9924795874516545
```

### Visualization of support vectors

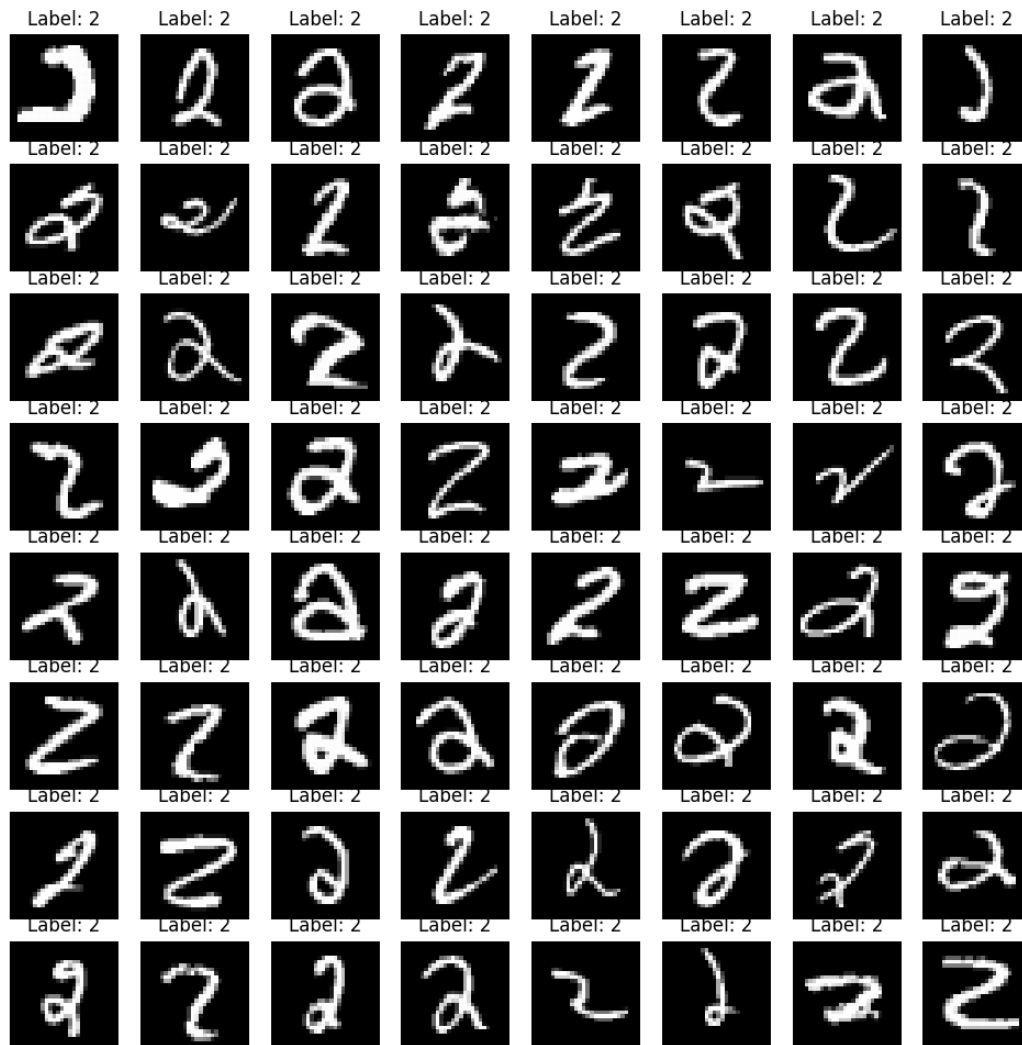
Support vectors are the data points that lie closest to the decision boundary in a Support Vector Machine model.

Support vectors are the key elements in determining the optimal hyperplane that separates different classes. The SVM algorithm aims to maximize the margin, which is the distance between the support vectors of different classes and the decision boundary.

Since support vectors are the points that are closest to the decision boundary, they can be thought of as edge cases. These are the points where the SVM struggles the most to classify, and thus, they have a direct impact on the positioning of the decision boundary.

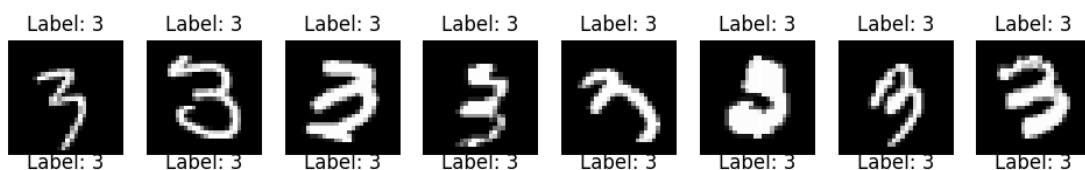
#### Support Vectors

Here you can see the visualization of support vectors for class "2".



Support vectors do not necessarily represent the typical examples of the classes in the training set. Instead, they are the critical points that are hardest to classify and are therefore closest to the margin between classes. These points might not look like typical examples because they are on the "edge" of the class distribution.

#### Support Vectors

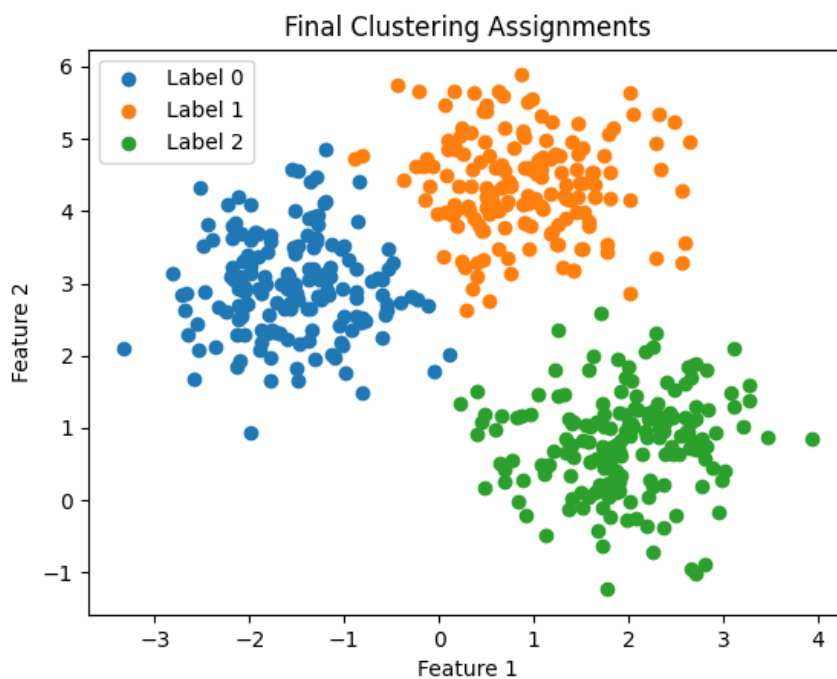
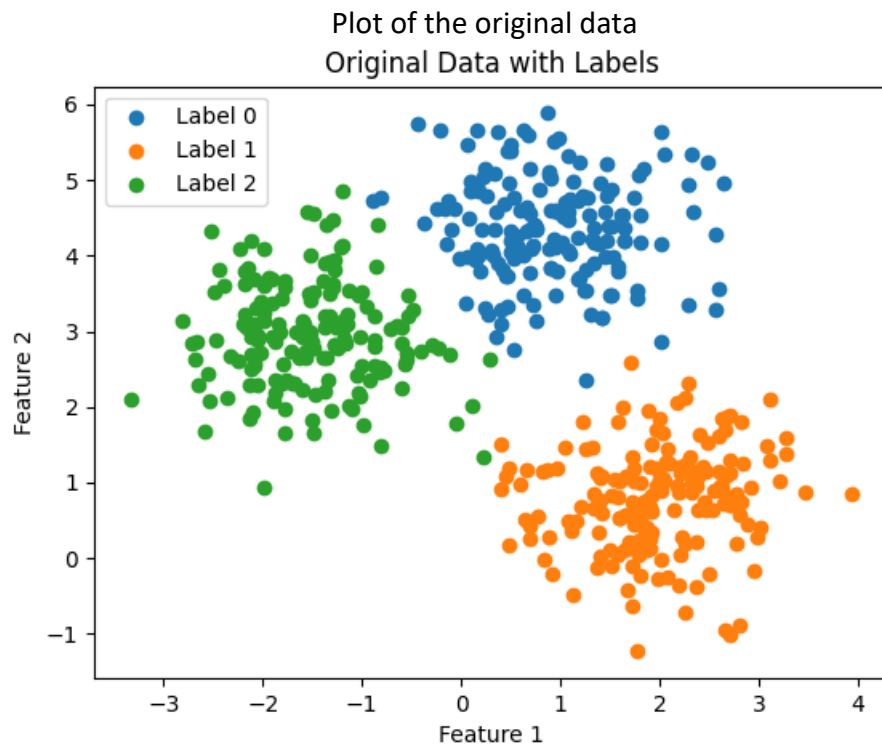


Support vector visualization for class 3

## 2- K-Means Clustering

- **Plotting the data**

Preparing the data was far easier compared to the SVM. The data has been given to us by our instructor. npy format. We started by plotting the initial data. This is important because we will be comparing the initial points with the points after clustering to see the effects of clustering.



Plot of the clustered data

## References

- [Link to the Github repository](#)
- SWE 582 course notes and slides
- ChatGPT for debugging the code
- <https://stats.stackexchange.com/questions/73032/linear-kernel-and-non-linear-kernel-for-support-vector-machine>
- <https://medium.com/data-folks-indonesia/step-by-step-to-understanding-k-means-clustering-and-implementation-with-sklearn-b55803f519d6>