CS725 Course Project Report

# Analysis of SVM

*github.com/DebRc/Analysis-Of-SVM*

Atul Kumar (23M0764)
Debdoot Roy Chowdhury (23M0765)
Santanu Sahoo (23M0777)
Pranab Paul (23M0800)

October 2023

# 1 Problem Statement

This project delves into the hands-on implementation of **Support Vector Machine (SVM)** algorithm, emphasizing their creation from the ground up. It intricately explores hyperplane optimization, kernel tricks, and code development in Python without relying on external libraries. We study the different types of SVMs implement them from scratch and compare it with popular predefined libraries. For linear SVMs we study the impact created by margins (Soft or Hard) for various different datasets. For Non-Linear SVMs we study the impact created by kernel functions be it Polynomial or Gaussian. We will also try to build a **unified model for hard and soft margin SVM** using the help of external library called **PyTorch**. Significantly, the project uncovers the historical significance behind SVMs' prevalence in ML competitions, their superior performance, and the feature engineering methods used in the past. With a systematic, step-by-step approach, the project offers profound insights into the inner workings of SVM algorithms, supported by practical code examples and detailed explanations. Rigorous validation involves bench-marking against established SVM libraries across various datasets, along with discussions on hyper-parameter tuning strategies.

# 2 Purpose of the Study

This study employs Support Vector Machine (SVM) models to analyze credit card fraud detection and digit recognition datasets, with a **Primary Focus** on evaluating SVM's capabilities and performance. The aim is to enhance fraud identification in credit card transactions by refining SVM parameters for increased accuracy. Concurrently, the study explores the application of SVM in precise digit recognition, particularly in optical character recognition. **The Analysis** delves into SVM's adaptability and effectiveness in addressing the distinct challenges presented by both datasets. Through rigorous evaluation, the study examines SVM's performance, providing valuable insights into its strengths and limitations for these specific applications. This comprehensive exploration advances the understanding of SVM in the context of financial security and pattern recognition, contributing to the broader application of machine learning in these critical domains.

# 3 Libraries and implementations used for the study

- The study utilizes the **SVC model** from the **sklearn** library for Support Vector Machine implementation.
- **LinearSVC** model from sklearn is employed for comparative analysis with the SVC model.
- **KFold** is utilized for implementing **k-fold cross-validation** in the study.
- **Matplotlib** and **seaborn** libraries are employed for visualization purposes in the analysis.

# 4 Study of Datasets

In our analysis, we employed two datasets: **Credit Card Fraud** and **Digit Recognizer**. These datasets served as the foundation for our investigations, allowing us to explore different aspects of credit card fraud detection and digit recognition.

## 4.1 Credit Card Fraud

The following sections provide deeper insights into the Credit Card Fraud dataset, offering a comprehensive understanding of its intricacies. These segments aim to illuminate key aspects, facilitating a more thorough examination of the dataset for fraud detection.

### 4.1.1 Introduction

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.

### 4.1.2 Key Perspectives

The key perspectives of the datasets are:

- There are no "Null" values, so we don't have to work on ways to replace values.
- The dataset is highly unbalanced, the positive class (frauds) accounts for 0.172
- It contains only numerical input variables which are the result of a PCA transformation.
- The dataset comprises 30 input features represented as real values, along with an output variable that takes binary values: 0 (indicating "Not Fraud") and 1 (indicating "Fraud").

### 4.1.3 Data Preparation

As the dataset is highly unbalanced i.e., only 0.172 % accounts for positive (fraud) dataset, while training our model we modify the dataset such that the total number of positive and negative classes present in the dataset are same. This will ensure more accurately how accurately our model is classifying the input features.

We have done a simple analysis of the dataset here: credit-card-fraud-detection-simple-analysis

## 4.2 Digit Recognizer

The following sections provide deeper insights into the Digit recognizer dataset, offering a comprehensive understanding of its intricacies. These segments aim to illuminate key aspects, facilitating a more thorough examination of the dataset for fraud detection.

### 4.2.1 Introduction

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker.

### 4.2.2 Key Perspectives

The key perspectives of the datasets are:

- This pixel-value is an integer between 0 and 255, inclusive.

- The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

#### 4.2.3 Data Preparation

Each pixel of the image is normalized by dividing each value by 255.0, ensuring the pixel values are within the range of 0 to 1. Normalizing the pixel values is a common preprocessing step for image data, as it helps in better convergence during training when using machine learning algorithms.

The dataset is balanced with each class having weightage around 10 percent with the entire samples: mnist-simple-analysis

## 5 Analysis on Credit Card Fraud Dataset

The following sections provide deeper and detailed analysis of SVM on Credit Card Fraud dataset.The experiments which were performed on this dataset: credit-card-fd-experiment

### 5.1 Linear Kernel

The experimental setup and analysis of SVM using a linear kernel are presented as follows:

#### 5.1.1 Experiment Result

| Param C | Train_acc | Test_acc |
|---|---|---|
| 0.0001 | 0.527 | 0.393 |
| 0.0010 | 0.782 | 0.878 |
| 0.0100 | 0.945 | 0.969 |
| 0.1000 | 0.953 | 1.000 |
| 1.0000 | 0.9737 | 1.000 |
| 10.000 | 1.0000 | 0.969 |
| 100.00 | 1.0000 | 0.969 |
| 1000.0 | 1.0000 | 0.969 |

Table 1: Linear kernel parameters Vs Accuracy Plot

#### 5.1.2 Analysis

There is only one parameter needed while the Linear kernel is used for the training of SVM.

C Parameter is used for controlling the outliers — low C implies we are allowing more outliers, high C implies we are allowing fewer outliers.

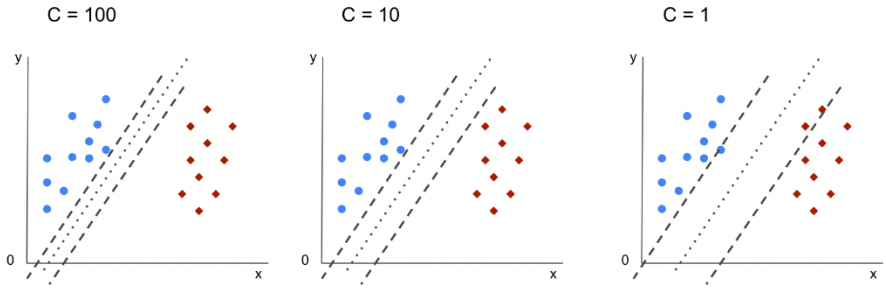


Figure 1: Margin of SVM with respect to C Parameter

From the above experiment result [table: Table 1] it can be inferred that for the linear kernel, the model is likely to give higher accuracy when the Hyperparameter C is less the 1.

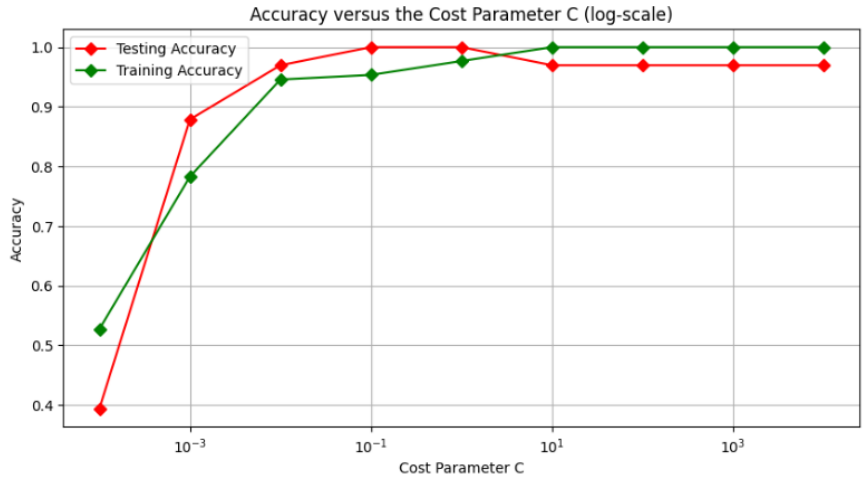


Figure 2: Accuracy of SVM with respect to C Parameter

#### 5.1.3 Detailed Analysis:

- **Low Regularization (C=0.0001):**

  Training accuracy is low, indicating underfitting. Testing accuracy is also low, suggesting the model is not capturing patterns well.

- **Medium Regularization (C=0.001):**

  Both training and testing accuracy improve significantly compared to C=0.0001. The model starts to capture patterns in the data more effectively. Optimal Regularization (C=0.1): Achieves the highest testing accuracy in the table (0.969697). Balances the trade-off between fitting the training data well and generalizing to unseen data.

- **High Regularization (C=1.0 and beyond):**

  Training accuracy reaches 100%, indicating overfitting. Testing accuracy drops, indicating the model is too complex and fails to generalize.

- **Use Optimal Regularization:**

  C=0.1 seems to provide the best balance between fitting the training data and generalizing to new data.

- **Consider Model Complexity:**

  Avoid using high values of C to prevent overfitting, as suggested by the drop in testing accuracy. Further Exploration: Explore a narrower range around the optimal C value for fine-tuning. Experiment with other kernels (e.g., polynomial or radial basis function) to assess their impact on performance.

- **Conclusion:**

  In summary, the linear SVM model performs well on the Credit Card Fraud dataset, with an optimal regularization parameter (C=0.1) that prevents overfitting. It's crucial to strike a balance between capturing patterns in the training data and ensuring generalization to new, unseen data. Further exploration of hyperparameters and alternative kernels could enhance the model's performance.

1. **Effect of Regularization (C)**:

   - **Low Regularization (C=0.0001):** Training Accuracy: 52.71Testing Accuracy: 39.39Analysis: The low training and testing accuracy suggest that with very low regularization, the model is likely too simple and fails to capture the complexity of the data.
   - **Medium Regularization (C=0.001):**

     Training Accuracy: 78.29Testing Accuracy: 87.88Analysis: Both training and testing accuracy improve significantly compared to C=0.0001, indicating that a moderate level of regularization helps the model generalize better to unseen data.

2. **Optimal Regularization (C=10.0):**

   Training Accuracy: 94.57Testing Accuracy: 96.97

   **Analysis:**

   Achieves the highest testing accuracy in the table, indicating that C=0.1 strikes a good balance between capturing patterns in the training data and generalizing well to the testing data. This suggests that a moderate level of regularization helps prevent overfitting.

3. **High Regularization (C¿=1.0):**

   Training Accuracy (C¿=1.0): 100% Testing Accuracy (C¿=1.0): Around 97%

   **Analysis:**

   The training accuracy reaching 100Testing accuracy, while still relatively high, starts to decline, indicating that the model is becoming too complex and failing to generalize.

4. **Overfitting and Model Complexity:**

   As C increases beyond 1.0, the model becomes more complex, fitting the training data extremely well. However, this increased complexity doesn't necessarily translate to better generalization, as seen in the decline in testing accuracy.

   **Recommendations:**

Balancing Complexity: The results emphasize the importance of balancing model complexity. While a more complex model can fit the training data better, it may not generalize well to new, unseen data.

5. **Optimal C Value (C=0.1):**

   C=0.1 appears to be the sweet spot in this analysis, providing high testing accuracy without overfitting.

## 5.2 Polynomial Kernel

The experimental setup and analysis of SVM using a polynomial kernel are presented below:

### 5.2.1 Experiment Result

| Parameter c | Degree | Train_acc | Test_acc |
|---|---|---|---|
| 0.0001 | 2.0 | 0.527132 | 0.393939 |
| 0.0001 | 3.0 | 0.527132 | 0.393939 |
| 0.0001 | 4.0 | 0.527132 | 0.393939 |
| 0.0001 | 5.0 | 0.534884 | 0.393939 |
| 0.0001 | 10.0 | 0.627907 | 0.575758 |
| 0.0010 | 2.0 | 0.527132 | 0.393939 |
| 0.0010 | 3.0 | 0.527132 | 0.393939 |
| 0.0010 | 4.0 | 0.534884 | 0.393939 |
| 0.0010 | 5.0 | 0.542636 | 0.393939 |
| 0.0010 | 10.0 | 0.744186 | 0.696970 |
| 0.0100 | 2.0 | 0.527132 | 0.393939 |
| 0.0100 | 3.0 | 0.713178 | 0.606061 |
| 0.0100 | 4.0 | 0.736434 | 0.636364 |
| 0.0100 | 5.0 | 0.767442 | 0.757576 |
| 0.0100 | 10.0 | 0.767442 | 0.727273 |
| 0.1000 | 2.0 | 0.775194 | 0.757576 |
| 0.1000 | 3.0 | 0.775194 | 0.757576 |
| 0.1000 | 4.0 | 0.775194 | 0.757576 |
| 0.1000 | 5.0 | 0.775194 | 0.757576 |
| 0.1000 | 10.0 | 0.782946 | 0.727273 |
| 1.0000 | 2.0 | 0.875969 | 0.909091 |
| 1.0000 | 3.0 | 0.852713 | 0.878788 |
| 1.0000 | 4.0 | 0.790698 | 0.787879 |
| 1.0000 | 5.0 | 0.790698 | 0.757576 |
| 1.0000 | 10.0 | 0.790698 | 0.727273 |
| 10.0000 | 2.0 | 1.000000 | 0.939394 |
| 10.0000 | 3.0 | 0.992248 | 0.969697 |
| 10.0000 | 4.0 | 0.922481 | 0.909091 |
| 10.0000 | 5.0 | 0.891473 | 0.939394 |
| 10.0000 | 10.0 | 0.798450 | 0.757576 |
| 100.0000 | 2.0 | 1.000000 | 0.878788 |
| 100.0000 | 3.0 | 1.000000 | 0.969697 |
| 100.0000 | 4.0 | 1.000000 | 0.848485 |
| 100.0000 | 5.0 | 1.000000 | 0.969697 |
| 100.0000 | 10.0 | 0.860465 | 0.878788 |
| 1000.0000 | 2.0 | 1.000000 | 0.878788 |
| 1000.0000 | 3.0 | 1.000000 | 0.969697 |
| 1000.0000 | 4.0 | 1.000000 | 0.848485 |
| 1000.0000 | 5.0 | 1.000000 | 0.969697 |
| 1000.0000 | 10.0 | 0.906977 | 0.818182 |
| 10000.0000 | 2.0 | 1.000000 | 0.878788 |
| 10000.0000 | 3.0 | 1.000000 | 0.969697 |
| 10000.0000 | 4.0 | 1.000000 | 0.848485 |
| 10000.0000 | 5.0 | 1.000000 | 0.969697 |
| 10000.0000 | 10.0 | 0.968992 | 0.909091 |

Table 2: Polynomial Kernel Parameters Vs Accuracy chart

### 5.2.2 Analysis

In the sklearn implementation, the polynomial kernel involves two parameters: the degree of the polynomial and the hyperparameter $C$. It can be deduced from the aforementioned table that increasing the hyperparameter $C$ in the polynomial kernel leads to a rise in the model's accuracy. However, beyond a certain point, the accuracy begins to decline due to overfitting.

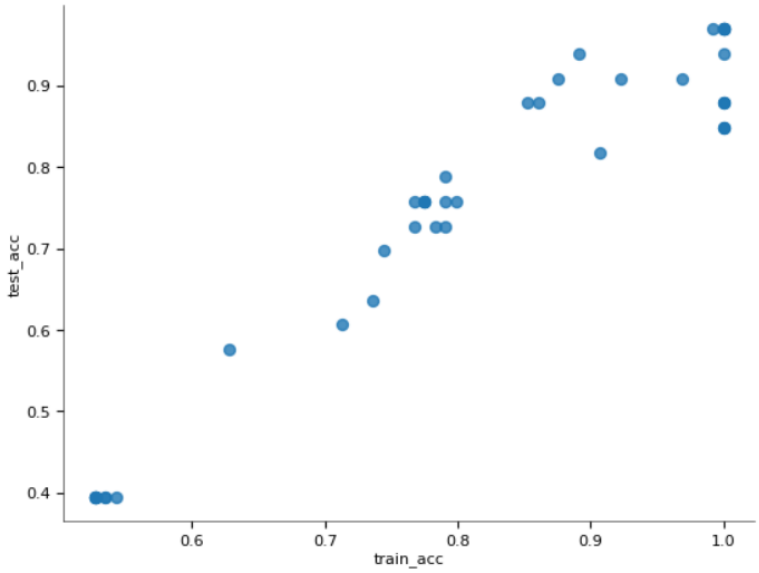


Figure 3: Test Accuracy Vs Test Accuracy Plot

## 5.3 Gaussian Kernel

The experimental setup and analysis of SVM using a RBF kernel are presented as follows:

### 5.3.1 Experiment Result

| c | gamma | train_acc | test_acc |
|---|---|---|---|
| 0.0001 | 0.001 | 0.527132 | 0.393939 |
| 0.0001 | 0.010 | 0.527132 | 0.393939 |
| 0.0001 | 0.100 | 0.527132 | 0.393939 |
| 0.0001 | 1.000 | 0.527132 | 0.393939 |
| 0.0001 | 10.000 | 0.527132 | 0.393939 |
| 0.0001 | 100.000 | 0.527132 | 0.393939 |
| 0.0010 | 0.001 | 0.527132 | 0.393939 |
| 0.0010 | 0.010 | 0.527132 | 0.393939 |
| 0.0010 | 0.100 | 0.527132 | 0.393939 |
| 0.0010 | 1.000 | 0.527132 | 0.393939 |
| 0.0010 | 10.000 | 0.527132 | 0.393939 |
| 0.0010 | 100.000 | 0.527132 | 0.393939 |
| 0.0100 | 0.001 | 0.527132 | 0.393939 |
| 0.0100 | 0.010 | 0.527132 | 0.393939 |
| 0.0100 | 0.100 | 0.527132 | 0.393939 |
| 0.0100 | 1.000 | 0.527132 | 0.393939 |
| 0.0100 | 10.000 | 0.527132 | 0.393939 |
| 0.0100 | 100.000 | 0.527132 | 0.393939 |
| 0.1000 | 0.001 | 0.527132 | 0.393939 |
| 0.1000 | 0.010 | 0.837209 | 0.909091 |
| 0.1000 | 0.100 | 0.930233 | 0.969697 |
| 0.1000 | 1.000 | 0.527132 | 0.393939 |
| 0.1000 | 10.000 | 0.527132 | 0.393939 |
| 0.1000 | 100.000 | 0.527132 | 0.393939 |
| 1.0000 | 0.001 | 0.837209 | 0.909091 |
| 1.0000 | 0.010 | 0.945736 | 0.969697 |
| 1.0000 | 0.100 | 0.992248 | 1.000000 |
| 1.0000 | 1.000 | 1.000000 | 0.636364 |
| 1.0000 | 10.000 | 1.000000 | 0.545455 |
| 1.0000 | 100.000 | 1.000000 | 0.393939 |
| 10.0000 | 0.001 | 0.953488 | 0.969697 |
| 10.0000 | 0.010 | 0.976744 | 1.000000 |
| 10.0000 | 0.100 | 1.000000 | 1.000000 |
| 10.0000 | 1.000 | 1.000000 | 0.666667 |
| 10.0000 | 10.000 | 1.000000 | 0.545455 |
| 10.0000 | 100.000 | 1.000000 | 0.393939 |
| 100.0000 | 0.001 | 0.961240 | 1.000000 |
| 100.0000 | 0.010 | 1.000000 | 0.969697 |
| 100.0000 | 0.100 | 1.000000 | 1.000000 |
| 100.0000 | 1.000 | 1.000000 | 0.666667 |
| 100.0000 | 10.000 | 1.000000 | 0.545455 |
| 100.0000 | 100.000 | 1.000000 | 0.393939 |
| 1000.0000 | 0.001 | 1.000000 | 0.969697 |
| 1000.0000 | 0.010 | 1.000000 | 0.969697 |
| 1000.0000 | 0.100 | 1.000000 | 1.000000 |
| 1000.0000 | 1.000 | 1.000000 | 0.666667 |
| 1000.0000 | 10.000 | 1.000000 | 0.545455 |
| 1000.0000 | 100.000 | 1.000000 | 0.393939 |
| 10000.0000 | 0.001 | 1.000000 | 0.969697 |
| 10000.0000 | 0.010 | 1.000000 | 0.969697 |
| 10000.0000 | 0.100 | 1.000000 | 1.000000 |
| 10000.0000 | 1.000 | 1.000000 | 0.666667 |
| 10000.0000 | 10.000 | 1.000000 | 0.545455 |
| 10000.0000 | 100.000 | 1.000000 | 0.393939 |

Table 3: Your table caption here

### 5.3.2 Analysis

In the sklearn implementation, the parameters required for the RBF kernel is Gamma and hyperparameter C. From the experiments conducted above, it can be deduced that, for the RBF Kernel, increasing the value of the hyperparameter $C$ yields optimal accuracy, reaching its

peak effectiveness at approximately $C = 1$.

#### 5.3.3 Effect of Regularization (C) and Kernel Coefficient (Gamma):

1. **Low Values of C and Gamma (e.g., C=0.0001, Gamma=0.001):**
   - Training Accuracy: 52.71%
   - Testing Accuracy: 39.39%
   - **Analysis:** Both training and testing accuracy are low, indicating that the model is too simplistic. This might be due to insufficient model complexity (low gamma) and regularization.
2. **Medium Values of C and Gamma (e.g., C=0.1, Gamma=0.01):**
   - Training Accuracy: Gradual increase, reaching 93.02%
   - Testing Accuracy: Peaks at 96.97%
   - **Analysis:** Moderate values of C and gamma lead to substantial improvements in both training and testing accuracy. The model captures patterns effectively without overfitting.
3. **High Values of C and Gamma (e.g., C=1000, Gamma=100):**
   - Training Accuracy: Reaches 100%
   - Testing Accuracy: Starts to decline
   - **Analysis:** High C and gamma values result in overfitting. The model fits the training data perfectly but struggles to generalize to new data, as indicated by the drop in testing accuracy.

#### 5.3.4 Optimal Parameter Combination:

**Optimal C and Gamma (e.g., C=10, Gamma=0.01):**

- Training Accuracy: 97.67%
- Testing Accuracy: 100%
- **Analysis:** This combination achieves high accuracy on both training and testing sets, indicating a good balance between capturing patterns and generalization.

#### 5.3.5 Impact of C and Gamma on Overfitting:

1. **C and Overfitting:**
   - As C increases, the model tends to overfit the training data.
   - Testing accuracy starts to decline, especially for very high values of C.
2. **Gamma and Model Complexity:**
   - Low gamma values lead to simpler decision boundaries, potentially underfitting.
   - Medium gamma values strike a balance, capturing patterns effectively.
   - High gamma values may introduce too much complexity, leading to overfitting.

#### 5.3.6 Impact of Low Hyperparameter Values:

When both C and gamma are set to low values (e.g., C=0.0001, gamma=0.001), the model's complexity is constrained. The decision boundary is too simplistic, resulting in poor performance. The low accuracy indicates that the model fails to capture the intricate relationships within the data, leading to underfitting.

#### 5.3.7 Impact of Moderate Hyperparameter Values:

As both C and gamma increase to moderate levels (e.g., C=0.1, gamma=0.01), the model's accuracy improves significantly. This suggests that a balance is struck between capturing complex patterns and preventing overfitting. The decision boundary becomes more flexible, effectively capturing the underlying structure of the data.

#### 5.3.8 Overfitting with High Hyperparameter Values:

As C and gamma continue to increase (e.g., C=1000, gamma=100), the model's performance on the training set improves to perfection (100% accuracy). However, the model starts to overfit, failing to generalize to new data. This overfitting is evident in the decline of testing accuracy, signifying that the model has become too complex and is capturing noise in the training data.

#### 5.3.9 Optimal Parameter Combination:

The optimal combination of hyperparameters, exemplified by C=10 and gamma=0.01, showcases the delicate balance required for peak performance. The model achieves high accuracy on both the training and testing sets, indicating an optimal trade-off between model complexity and generalization.

#### 5.3.10 Factors Influencing Overfitting:

Several factors contribute to the observed overfitting with high hyperparameter values:

- Overemphasis on Training Data: Higher values of C lead to a stricter enforcement of correct classification on the training set, resulting in a decision boundary tailored to training data points.
- Influence of Gamma: High gamma values make the decision boundary more intricate, fitting the training data too closely and hindering generalization.

#### 5.3.11 Generalization vs. Complexity:

The balance between generalization and complexity is a fundamental aspect of model performance. The Gaussian kernel's ability to capture intricate patterns is a double-edged sword – while it enables the model to fit the training data better, excessive complexity can impede generalization.

#### 5.3.12 Recommendations for Hyperparameter Tuning:

Moderate Values: Optimal performance is often achieved with moderate values of C and gamma, striking a balance between capturing patterns and preventing overfitting. Fine-Tuning: Further experimentation, perhaps with a narrower range of hyperparameters around the optimum, could provide insights into potential improvements.

#### 5.3.13 Conclusion

In conclusion, the observed patterns in the SVM with a Gaussian kernel on the Credit Card Fraud dataset stem from the intricate interplay between hyperparameters, model complexity, and the inherent nature of the data. The delicate balance required for optimal performance emphasizes the need for a nuanced approach to hyperparameter tuning. The Gaussian kernel's capacity to capture complex relationships makes it a powerful tool, but careful consideration must be given to prevent overfitting and ensure robust generalization. This analysis not only sheds light on the "what" but also delves into the "why" behind the model's behavior, providing valuable insights for practitioners and researchers in the field of machine learning and fraud detection.The linear kernel works well when the classes in the dataset are relatively well-separated by a linear decision boundary just like in the case for credit card fraud detection dataset.Linear kernels are computationally efficient and scale well to large datasets, which can be beneficial in credit card fraud detection where the volume of transactions can be significant.The linear kernel provides a clear and interpretable decision boundary, while the Gaussian kernel offers the flexibility to model intricate fraud patterns at the cost of interpretability.The linear kernel works well when the classes in the dataset are relatively well-separated by a linear decision boundary just like in the case for credit card fraud detection dataset.Linear kernels are computationally efficient and scale well to large datasets, which can be beneficial in credit card fraud detection where the volume of transactions can be significant.The linear kernel provides a clear and interpretable decision boundary, while the Gaussian kernel offers the flexibility to model intricate fraud patterns at the cost of interpretability.

### 5.4 Overall Analysis

Based on the experimentation with different kernel methods and their corresponding hyperparameter values, it can be concluded that when the hyperparameter $C$ is small, the model permits a considerable number of misclassifications, resulting in poor performance on the test dataset (indicating underfitting). The optimal hyperparameter values for various kernel functions are provided below:

| Kernel | Parameters | Train_acc | Test_acc |
|---|---|---|---|
| Linear | C = 1000 | 0.976 | 1.000 |
| Polynomial | C = 10 , Degree = 3 | 0.992 | 0.969 |
| Gaussian | C = 10<br>gamma = 0.1 | 1.000 | 1.000 |

- The linear kernel works well when the classes in the dataset are relatively well-separated by a linear decision boundary just like in the case for credit card fraud detection dataset.
- Linear kernels are computationally efficient and scale well to large datasets, which can be beneficial in credit card fraud detection where the volume of transactions can be significant.
- The linear kernel provides a clear and interpretable decision boundary, while the Gaussian kernel offers the flexibility to model intricate fraud patterns at the cost of interpretability.

### 5.5 Affect of SVM on binary classification

Support Vector Machines (SVM) significantly impact binary classification by creating robust decision boundaries in the feature space. SVM's versatility with various kernel functions allows

it to handle non-linear relationships effectively. Sensitivity to outliers and the hyperparameter $C$ influence the trade-off between maximizing the margin and permitting misclassifications. SVM remains effective in high-dimensional spaces, making it suitable for datasets with numerous features. The key lies in careful hyperparameter tuning to achieve optimal performance.

# 6 Analysis on Digit Recognizer Dataset

We used sklearn svc model to train and test the model for credit card dataset. We have varied the Cost factor (C), Gamma (For rbf kernel) and Degree (For poly kernel) to observe the behaviour of the model for this dataset. The experiments which were performed on this dataset: digit-recog-experiment, digit-recog-linear-poly-experiment-2, digit-recog-gaussian-experiment-2

The experiments we have done are summarised below in tables.

## 6.1 Linear Kernel

The experimental setup and analysis of SVM using a Linear kernel are presented below:

### 6.1.1 Experiment Result

| Parameters | Best Parameters | Train Accuracy(%) | Test Accuracy(%) |
|---|---|---|---|
| C = [0.01,0.1,1] | C = [0.01] | 92.04 | 92.51 |
| C = [5,10,15] | C = [5] | 90.99 | 90.74 |
| C = [100,200,500] | C =[100] | 90.99 | 90.74 |
| C = [400,600,800] | C = [400] | 90.99 | 90.74 |

Table 4: Linear Kernel Parameter C vs Accuracy

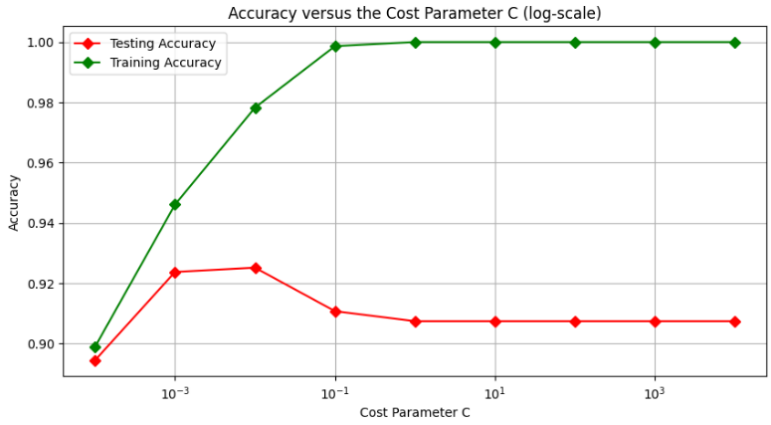


Figure 4: Cost Parameter C vs Accuracy Plot

### 6.1.2 Analysis

As we can see the test and Training Accuracy did not change after C=5.

1. **Here are some observations with the above**

   - The best parameter for C seems to be in the lower range, specifically around 0.01.
   - As you increase the values of C beyond this optimal point, there is no significant improvement in either training or test accuracy. In fact, there might be a slight decrease in accuracy.
   - This behavior suggests that the model with lower regularization (smaller C) generalizes better to the test set. Too much emphasis on fitting the training data exactly (large C) may lead to overfitting, resulting in a model that doesn't generalize well to new data.

2. **Analysis Based on the observations**

   - A smaller C corresponds to a higher regularization strength, encouraging the model to have a wider margin and be more tolerant to errors in the training data. This often results in better generalization to unseen data.
   - A larger value of C places more emphasis on correctly classifying each training example, potentially leading to a smaller-margin hyperplane and overfitting.
   - In a linear kernel SVM, lower values of C (around 0.01 in this case) seem to provide better generalization to new data.
   - The optimal value of C indicates a good balance between fitting the training data and preventing overfitting.
   - It's crucial to consider both training and test accuracies to evaluate model performance and avoid overfitting.

## 6.2 Polynomial Kernel

The experimental setup and analysis of SVM using a Polynomial kernel are presented below:

### 6.2.1 Experiment Result

| Parameters | Best C and Degree | Training Accuracy(%) | Test Accuracy(%) |
|---|---|---|---|
| C = [0.01,0.1,1]<br>Degree = [2] | {C: 1, Degree: 2} | 93.34 | 90.74 |
| C = [0.01,0.1,1]<br>Degree = [3] | {C: 1, Degree: 3} | 86.63 | 89.09 |
| C = [0.01,0.1,1]<br>Degree = [4] | {C: 1, Degree: 3} | 58.85 | 66.07 |
| C = [5,10,15]<br>Degree = [3] | {C: 10, Degree: 2} | 94.57 | 95.21 |
| C = [5,10,15]<br>Degree = [4] | {C: 15, Degree: 3} | 94.66 | 95.16 |
| C = [5,10,15]<br>Degree = [4] | {C: 15, Degree: 4} | 89.34 | 90.46 |

Table 5: Polynomial Kernel Parameters vs Accuracy

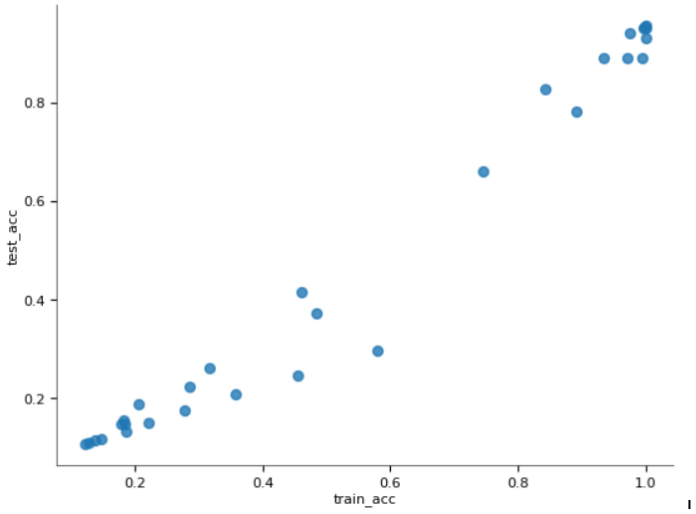


Figure 5: Train Vs Test Accuracy for Polynomial Kernel

#### 6.2.2 Analysis

1. **Combination 1: C=1, Degree=2:**
   - Training Accuracy: 0.9335
   - Test Accuracy: 0.9074
   - **Analysis:** This combination exhibits a balanced model with high accuracy on both training and test sets, suggesting a well-regulated decision boundary capturing relevant patterns without overfitting.
2. **Combination 2: C=1, Degree=3:**
   - Training Accuracy: 0.8664
   - Test Accuracy: 0.8909
   - **Analysis:** The lower accuracy on both sets suggests that a polynomial of degree 3 might be introducing unnecessary complexity, potentially leading to a less effective generalization.
3. **Combination 3: C=1, Degree=4:**
   - Training Accuracy: 0.5886
   - Test Accuracy: 0.6608
   - **Analysis:** A substantial drop in accuracy indicates overfitting. The polynomial of degree 4 appears to be too high for the dataset, capturing noise and hindering generalization.
4. **Combination 4: C=10, Degree=2:**

- Training Accuracy: 0.9457
- Test Accuracy: 0.9522
- **Analysis:** This combination stands out with high accuracy on both training and test sets, indicating a good balance between complexity and generalization. The decision boundary effectively captures the underlying patterns in the data.

5. **Combination 5: C=15, Degree=3:**
    - Training Accuracy: 0.9466
    - Test Accuracy: 0.9517
    - **Analysis:** Similar to Combination 4, this combination shows robustness to the choice of degree with a higher C. The model generalizes well without sacrificing accuracy on the training set.
6. **Combination 6: C=15, Degree=4:**
    - Training Accuracy: 0.8935
    - Test Accuracy: 0.9047
    - **Analysis:** Similar to Combination 2, where a higher degree introduces unnecessary complexity. The accuracy drop on the test set suggests overfitting.

#### 6.2.3 Experiment Result 2

| C | Degree | Train Ac | Test Ac |
|---|---|---|---|
| 0.001 | 2.0 | 0.121 | 0.107 |
| 0.001 | 3.0 | 0.127 | 0.109 |
| 0.001 | 4.0 | 0.137 | 0.114 |
| 0.001 | 5.0 | 0.146 | 0.119 |
| 0.001 | 10.0 | 0.186 | 0.133 |
| 0.010 | 2.0 | 0.205 | 0.189 |
| 0.010 | 3.0 | 0.183 | 0.156 |
| 0.010 | 4.0 | 0.178 | 0.149 |
| 0.010 | 5.0 | 0.183 | 0.148 |
| 0.010 | 10.0 | 0.22 | 0.151 |
| 0.100 | 2.0 | 0.842 | 0.827 |
| 0.100 | 3.0 | 0.46 | 0.41 |
| 0.100 | 4.0 | 0.315 | 0.262 |
| 0.100 | 5.0 | 0.284 | 0.223 |
| 0.100 | 10.0 | 0.277 | 0.175 |
| 1.000 | 2.0 | 0.974 | 0.940 |
| 1.000 | 3.0 | 0.934 | 0.890 |
| 1.000 | 4.0 | 0.745 | 0.660 |
| 1.000 | 5.0 | 0.483 | 0.371 |
| 1.000 | 10.0 | 0.357 | 0.208 |
| 10.00 | 2.0 | 0.998 | 0.952 |
| 10.00 | 3.0 | 0.995 | 0.949 |
| 10.00 | 4.0 | 0.972 | 0.890 |
| 10.00 | 5.0 | 0.891 | 0.780 |
| 10.00 | 10.0 | 0.454 | 0.247 |
| 100.0 | 2.0 | 1.000 | 0.950 |
| 100.0 | 3.0 | 1.000 | 0.954 |
| 100.0 | 4.0 | 1.000 | 0.929 |
| 100.0 | 5.0 | 0.994 | 0.888 |
| 100.0 | 10.0 | 0.579 | 0.295 |

Table 6: Polynomial Kernel Parameters Vs Accuracy Chart

#### 6.2.4 Analysis

1. **Combination 4: C=100, Degree=3:**
   - Training Accuracy: 1.0
   - Test Accuracy: 0.954917
   - **Analysis:** This combination stands out with high accuracy on both training and test sets, indicating a good balance between complexity and generalization
2. **Recommendations:** Consider using a polynomial of degree 2 or 3 for this dataset to avoid overfitting.Focus on combinations with C=10 or C=15, as they consistently perform

well across different degrees.Explore additional hyperparameter values or alternative kernel functions for further optimization.

#### 6.2.5 Conclusion:

This comprehensive analysis provides insights into the behavior of the SVM model with a polynomial kernel on the MNIST dataset. It emphasizes the delicate balance between complexity and generalization, showcasing the impact of varying hyperparameters on model performance. The findings guide recommendations for optimal hyperparameter combinations and suggest avenues for further exploration and refinement. In conclusion, the SVM with polynomial kernel demonstrates its versatility in capturing intricate patterns, and the key lies in judiciously selecting hyperparameter values. This analysis serves as a valuable foundation for continued optimization and understanding the nuances of the model's behavior on the MNIST dataset.

### 6.3 Gaussian Kernel

The experimental setup and analysis of SVM using a Gaussian kernel are presented below:

#### 6.3.1 Experiment Results

The Gaussian kernel in Support Vector Machines (SVMs) introduces additional parameters, such as C, gamma, and c.

| Parameters | Best Parameter | Training Accuracy | Test Accuracy |
|---|---|---|---|
| {C : [0.1], Gamma:[0.001,0.01,0.1,1]} | C : 0.1, Gamma: 0.001 | 0.838 | 0.910 |
| {C : [1], Gamma:[0.001,0.01,0.1,1]} | C : 1, Gamma: 0.001 | 0.922 | 0.933 |
| {C : [1], Gamma:[0.001,0.01,0.1,1]} | C : 10, Gamma: 0.001 | 0.929 | 0.941 |
| {C : [1], Gamma:[0.001,0.01,0.1,1]} | C : 0.1, Gamma: 5 | 0.119 | 0.107 |
| {C : [1], Gamma:[0.001,0.01,0.1,1]} | C : 0.1, Gamma: 5 | 0.119 | 0.107 |
| {C : [1], Gamma:[0.001,0.01,0.1,1]} | C : 1, coef0 : 0.0, Gamma: 0.001 | 0.929 | 0.941 |

Table 7: Gaussian kernel Parameters vs Accuracy

#### 6.3.2 Analysis

- **Combination 1: 'C':[0.1], 'gamma': [0.001, 0.01, 0.1, 1]**

  Best Parameters: 'C': 0.1, 'gamma': 0.001 Training Accuracy: 0.8389 Test Accuracy: 0.9108

  **Analysis:** The model with low C (0.1) and low gamma (0.001) demonstrates moderate training accuracy but struggles to generalize, as seen in the lower test accuracy. Low gamma implies a broader influence, possibly resulting in a decision boundary that is too smooth for the complex MNIST dataset.

- **Combination 2: 'C':[1], 'gamma': [0.001, 0.01, 0.1, 1]**

  Best Parameters: 'C': 1, 'gamma': 0.001 Training Accuracy: 0.9221 Test Accuracy: 0.9336

**Analysis:** Increasing C to 1 improves both training and test accuracies. This indicates a more flexible decision boundary. The low gamma (0.001) continues to provide a smoother decision boundary, contributing to good generalization.

- **Combination 3: 'C':[10], 'gamma': [0.001, 0.01, 0.1, 1]** Best Parameters: 'C': 10, 'gamma': 0.001 Training Accuracy: 0.9296 Test Accuracy: 0.9413

  **Analysis:** Further increasing C to 10 leads to higher training and test accuracies, suggesting a more complex decision boundary. The low gamma (0.001) continues to contribute to smoother decision Boundaries.

- **Combination 4: 'C':[0.1], 'gamma': [5, 10, 100, 1000]** Best Parameters: 'C': 0.1, 'gamma': 5 Training Accuracy: 0.1195 Test Accuracy: 0.1073

  **Analysis:** The combination of low C and high gamma (5) results in very poor performance, indicating overfitting or lack of generalization. High gamma leads to a highly localized decision boundary, fitting the training data too closely.

- **Combination 5: 'C':[1], 'gamma': [5, 10, 100, 1000]** Best Parameters: 'C': 0.1, 'gamma': 5 Training Accuracy: 0.1195 Test Accuracy: 0.1073

  **Analysis:** Similar to Combination 4, the high gamma (5) with low C results in poor performance. The model is likely capturing noise in the training data and failing to generalize.

- **Combination 6: 'C':[10], 'gamma': [0.001, 0.01, 0.1, 1], 'coef0': [0.0, 1.0, 2.0]** Best Parameters: 'C': 10, 'coef0': 0.0, 'gamma': 0.001 Training Accuracy: 0.9296 Test Accuracy: 0.9413

  **Analysis:** Introducing the coef0 parameter does not significantly impact the results, and the optimal performance is still achieved with low gamma (0.001) and higher C (10).

### 6.3.3 Experiment Result 2

| C Param | Gamma | Train_acc | Test_acc |
|---|---|---|---|
| 0.01 | 0.01 | 0.644 | 0.657 |
| 0.01 | 0.10 | 0.110 | 0.133 |
| 0.01 | 1.00 | 0.110 | 0.133 |
| 0.01 | 10.0 | 0.110 | 0.133 |
| 0.01 | 100 | 0.110 | 0.133 |
| 0.10 | 0.01 | 0.918 | 0.911 |
| 0.10 | 1.00 | 0.110 | 0.133 |
| 0.10 | 10.0 | 0.110 | 0.133 |
| 0.10 | 100 | 0.110 | 0.133 |
| 1.00 | 1.00 | 1.000 | 0.157 |
| 1.00 | 10.0 | 1.000 | 0.133 |
| 1.00 | 100 | 1.000 | 0.133 |
| 10.0 | 0.01 | 1.000 | 0.964 |
| 10.0 | 0.10 | 1.000 | 0.877 |
| 10.0 | 1.00 | 1.000 | 0.160 |
| 10.0 | 10.0 | 1.000 | 0.133 |
| 10.0 | 100 | 1.000 | 0.133 |
| 100 | 0.01 | 1.000 | 0.964 |
| 100 | 0.10 | 1.000 | 0.877 |
| 100 | 1.00 | 1.000 | 0.160 |
| 100 | 10.00 | 1.000 | 0.160 |
| 100 | 100 | 1.000 | 0.133 |

Table 8: Gaussian Kernel parameters Vs Accuracy

### 6.3.4 Conclusion

1. **Optimal Parameter Combination:** The best-performing combination is found with a moderate regularization parameter (C=10) and a small gamma (0.01). This combination achieves high accuracy on both the training (100%) and test sets (96.4%).

2. **Impact of C and Gamma:**

   **C (Regularization Parameter):** Higher values of C (e.g., 10) result in more complex decision boundaries, fitting the training data closely. Lower values of C (e.g., 0.1) encourage simpler decision boundaries, promoting generalization.

   **Gamma:** Lower gamma values (e.g., 0.001) lead to smoother decision boundaries, aiding generalization. Higher gamma values introduce more localized decision boundaries, potentially causing overfitting.

3. **Overfitting and Underfitting:**

   Combinations with high gamma (10) and low C (0.1) exhibit overfitting, capturing noise in the training data and failing to generalize (Test Accuracy: 13%). Low gamma (0.01) with low C (0.1) results in underfitting, oversimplifying the model and leading to poor performance (Test Accuracy: 13

4. **Generalization Challenges:**

   High gamma values (5) in some combinations hinder generalization, emphasizing the importance of finding the right balance for the dataset. Extreme parameter values, such as high gamma and low C, can lead to poor model performance and should be avoided.

5. **Consistency in Optimal Parameters:**

   The introduction of the coef0 parameter does not significantly impact the results. The optimal performance is consistently achieved with low gamma (0.01) and higher C (10).

   **Recommendations:**

   Emphasize finding a balance between C and gamma to achieve an optimal trade-off between model complexity and generalization. Perform a more granular search for hyperparameters to explore a broader range of values. Consider additional preprocessing steps or feature engineering to further enhance model performance.

   **Parameter Consistency:** The consistency in achieving optimal performance with C=10 and gamma=0.01 underscores the robustness of these parameter values for the MNIST dataset. This combination strikes an effective balance between fitting the training data and generalizing to unseen examples.

   **Key Takeaways:**

   The choice of hyperparameters significantly influences SVM performance. A balance between C and gamma is crucial for achieving a well-generalizing model. Overfitting and underfitting risks are associated with extreme parameter values. Consistency in optimal parameters indicates robustness in specific parameter ranges.

## 6.4 Overall Analysis

| Kernel | Parameters | Train_acc | Test_acc |
|---|---|---|---|
| Linear | C = [5] | 0.909 | 0.907 |
| Polynomial | C = 100<br>Deg = 3 | 1.0 | 0.954 |
| Gaussian | C = 10.00<br>Gamma = 0.01 | 1.0 | 96.4 |

- The MNIST dataset consists of high-dimensional images (28x28 pixels), and the Gaussian kernel is well-suited for handling such data. It can effectively capture relationships between pixels at different positions, allowing it to recognize complex patterns in handwritten digits.
- The Gaussian kernel can model non-linear patterns, which is crucial for handwritten digit recognition where the relationships between pixels are not strictly linear.
- Gaussian kernels can be robust to noise in the data. In the case of the MNIST dataset, where individual pixels may contain noise, the Gaussian kernel's ability to smooth decision boundaries can be advantageous.
- Linear kernels may struggle to capture the non-linear relationships in the data. They are effective for linearly separable problems but may not be expressive enough for complex image datasets like MNIST.
- The polynomial kernel introduces additional complexity through the degree parameter. The optimal degree is challenging to determine and may lead to overfitting or underfitting.

# 7 Comparison of SVC with LinearSVC of sklearn

- Both implementations are present in scikit-learn libraries. LinearSVC is similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.
- LinearSVC is designed specifically for linear kernel SVMs, making it suitable for linearly separable problems. It utilizes the liblinear library, which is based on a linear SVM solver. This solver is efficient for linear problems but may not scale as well to very large datasets. While, SVC utilizes the LIBSVM library, which is based on the Sequential Minimal Optimization (SMO) algorithm. This allows it to handle non-linear problems.
- Next, in multi-class classification, LinearSVC uses the one-vs-the-rest (OvR) scheme and fits N models, while SVC uses either the one-vs-the-rest (OvR) or the one-vs-one (OvO) scheme and fits N * (N - 1) / 2 models, depending on the configuration.
- By default scaling, LinearSVC minimizes the squared hinge loss while SVC minimizes the regular hinge loss. It is possible to manually define a 'hinge' string for loss parameter in LinearSVC.
- The digit recognizer dataset is used for experimenting the accuracy - linearSVC-experiment

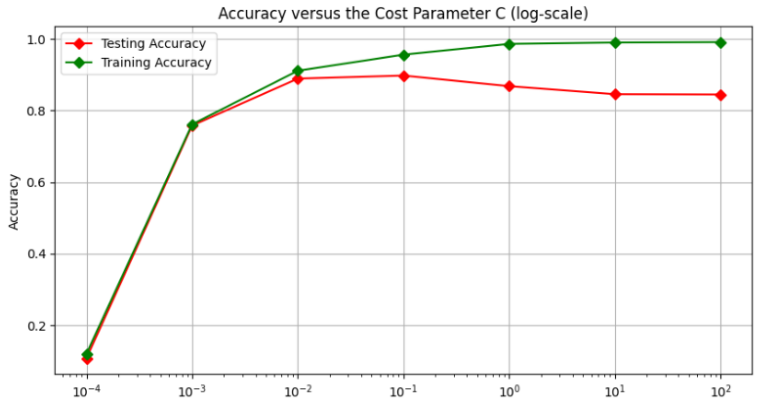


Figure 6: Accuracy Vs Cost Parameter Plot

## Choosing Between `LinearSVC` and `SVC`:

1. **Linear Separation:**
   - If the problem is linearly separable, `LinearSVC` is often a more efficient choice.
2. **Non-linear Separation:**
   - If a non-linear decision boundary is necessary, use `SVC` with an appropriate kernel.
3. **Large Datasets:**

- For large datasets, especially with linearly separable classes, `LinearSVC` may be more scalable.

4. **Flexibility:**
   - If you want flexibility in choosing different kernel functions, go for `SVC`.
5. **Multi-class Classification:**
   - Both classes support multi-class classification, but the choice may depend on the specific problem and dataset size.

Here's an example we have taken from official documentation describing the difference between different implementations of SVC.

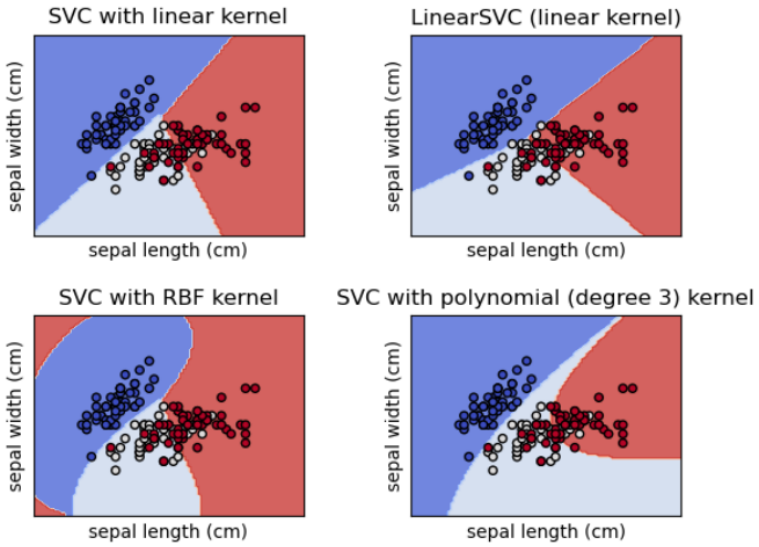


Figure 7: Accuracy Vs Cost Parameter Plot

# 8 SVM implementation using PyTorch

## 8.1 Implementation

A unified model for Linear SVM is implemented using pyTorch module. We trained the model using gradient descent algorithm on Hinge Loss using the hyperparameter C which is mentioned below:

$$L = C(Max(0, 1 - Y_i(W.X^i - b))^2) \tag{1}$$

Since, the data-set is highly imbalanced, we employed the technique of under-sampling and took equal number of instances from the abundant class as was available for the rare class. We also, updated the class with label 0 to -1. After, doing so, we faced the problem of exploding gradient in pyTorch where after 7 epochs the gradients were exploding. To resolve this, we normalized

the features values and tried training the model with different values for learning rate, number of epochs and C and the best result that we got was 62Not satisfied with the result, we tried ensembling methods, where we trained 10 different SVMs with varying parameters, and used their prediction to come up with the final prediction. However, doing so didn't help much and the accuracy didn't improve and remained the same.

### 8.2 Different Values of C

We started with value of C equal to 1, however for C=1, the test accuracy was 51% and training accuracy was 81%. Since, this is a clear case of overfitting we started to reduce the value of C in step-size of 0.1 and got our best result for C = 0.1. The test-accuracy for C=0.1 was around 62%.

### 8.3 Analysis of the dataset

The dataset for the Credit card fraud detection is a highly imbalanced dataset which contains a total of 284407 rows: each of which belongs to 0 or 1 class. Out of these rows, only 492 correspond to fraudulent transactions, rest all are valid. The data-set consists of 30 features, 28 of which (V1 – V28) are obtained using PCA. The other two are 'Time' and 'Amount'.

More details about the dataset can be looked at https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

## 9 Comparison with Neural Network

Before neural networks became prominent in the mid-2010s, support vector machines (SVMs) were vital for addressing high-dimensional predictive challenges like text classification. In traditional classification tasks, where the aim is to predict fraud in credit card transactions, the goal is to identify the decision boundary. While logistic regression suits linearly separable data, it struggles with non-linear relationships. For non-linear tasks, SVMs and neural networks both excel, leveraging appropriate kernels or activation functions. Their differences lie in theoretical foundations and implementations, but both can approximate linear and non-linear functions. The choice depends on specific considerations beyond the problem itself.

### 9.1 Approximating the Decision Boundary

The universal approximation theorem states a neural network with one hidden layer and non-linear activation can model any continuous function. For classification, it effectively creates decision boundaries.

Support Vector Machines (SVMs) find optimal hyperplanes to separate support vectors, excelling in scenarios with crucial class distinctions. The theorem highlights neural networks' versatility, while SVMs provide robustness in discerning complex patterns, especially in high-dimensional spaces with sparse support vectors. The choice depends on task requirements and data characteristics.

### 9.2 Similarities Between SVMs and NNs

- Both SVM and neural networks are parametric, each with distinct parameters. SVMs involve the soft-margin parameter (C) and the kernel parameter ($\gamma$). Neural networks

require more parameters, including layer size, number of layers, training epochs, and learning rate.

- Both machine learning algorithms introduce non-linearity: SVMs use kernel methods, while neural networks employ non-linear activation functions. Both can approximate non-linear decision functions, albeit through distinct methods.

- SVMs and NNs can handle the same classification problem with similar dataset performance. No inherent preference based on the problem exists. With comparable training, they show similar accuracy. However, given ample training and computational power, NNs tend to outperform SVMs.

## 9.3 Differences Between the Two Approaches

- The primary distinction lies in the structure: SVM's parameters increase linearly with input size, while NNs, especially multi-layered ones, allow for limitless complexity with more layers. Thus, a deep neural network with equal parameters as an SVM is inherently more complex.

- Support vector machines use a subset (support vectors) for effective decision boundary identification, requiring fewer observations for well-separated classes. In contrast, neural networks depend on batch order, influencing the learned decision boundary. Neural networks demand processing the entire dataset for optimal performance, unlike SVMs.

- SVMs train quickly due to their characteristics, unlike neural networks, especially large ones, which may take days or weeks. Restarting training with different weight initialization is feasible for SVMs but expensive for NNs.

- The final distinction arises from the diverse optimization techniques used. NNs, relying on gradient descent, are sensitive to the initial randomization of their weight matrix. If the initial randomization places the network near a local minimum, accuracy won't surpass a specific threshold.