Q.2. Perform an internet search to study the Principal Component Analysis (PCA) method and explain how it works. You are expected to use this method for feature extraction. Consider the following features and prepare them as inputs for your model: • The pixel values of the data in their original form • Features obtained by reducing the pixel dimensions using PCA to 100 dimensions. • Features obtained by reducing the pixel dimensions using PCA to 40 dimensions

Ans: Principal Component Analysis (PCA): Principal Component Analysis (PCA) is a widely employed method for reducing dimensionality and reshaping data, finding its application across various domains such as statistics, machine learning, and data analysis. Its primary objective is to streamline intricate high-dimensional data by preserving crucial information and minimizing extraneous noise. PCA accomplishes this by converting the initial data into a fresh coordinate system, where the novel axes, known as principal components, encapsulate the most substantial variance within the dataset

Principal Component Analysis (PCA) for Feature Extraction: for this we have perform the following steps.

1: Data Preparation: In accordance with the query, it's worth noting that every pixel can be regarded as a distinct feature. As a result, we are dealing with high-dimensional data, where each pixel represents an individual dimension. It is essential to begin by appropriately preparing our data, which includes tasks like normalization and centering, as previously emphasized in the earlier response.

2: PCA Dimensionality Reduction: Subsequently, we can employ PCA to effectively diminish the dimensionality of our data, all the while preserving the critical information. PCA enables us to derive fresh characteristics, which are essentially linear amalgamations of the initial pixel values, commonly referred to as principal components. These principal components are arranged based on their capacity to elucidate the variance within the dataset, with the initial component elucidating the highest variance, followed by the subsequent components in descending order of explanatory power.

3: Choose the Number of Components: During this stage, we have the flexibility to select the quantity of principal components we wish to preserve, aligning with our specific needs. In our scenario, we aim to reduce the pixel dimensions using PCA to two distinct levels: 100 dimensions for one set of requirements and 40 dimensions for another.

4: PCA Transformation : After establishing the desired number of components to retain (e.g., 100 or 40), the subsequent step involves converting our initial data into this reduced-dimensional realm by leveraging the chosen principal components. This transformation is conventionally executed by projecting our data onto these principal components.

For each of our scenarios:

Original Pixel Values: Here, we have our data in its unaltered state, where each individual pixel value serves as a distinct feature

Features obtained by reducing the pixel dimensions using PCA to 100 dimensions: In this particular scenario, our approach involves the implementation of PCA on the initial pixel data, retaining solely the foremost 100 principal components. These 100 principal components will serve as our fresh set of features.

Features obtained by reducing the pixel dimensions using PCA to 40 dimensions: Likewise, we will employ PCA on our original pixel data and retain the top 40 principal components. These selected 40 components will serve as the new set of features.

Q4: With the dataset containing all features.

Linear			
β	10^{-2}	1	10^{2}
Train accuracy	**0.62**		
Test accuracy	**0.53**		
Kernel RBF β = 0.1			
β	10^{-2}	1	10^{2}
Train accuracy			
Test accuracy			

Q5: With 100-dimensional data

Linear			
β	**10^{-5}**	**10^{-3}**	**1**
Train accuracy			
Test accuracy			
Kernel RBF β = 0.1			
β	**10^{-9}**	**10^{-6}**	**1**
Train accuracy			
Test accuracy			

Q6: With 40-dimensional data

Linear			
β	**10^{-5}**	**10^{-3}**	**1**
Train accuracy	**0.54**		
Test accuracy	**0.5**		
Kernel RBF β = 0.1			
β	**10^{-9}**	**10^{-6}**	**1**
Train accuracy	**0.1**		
Test accuracy	**0.111**		

Q8: Assuming we want to ensure that we find the number 2 under certain conditions, which criteria should we consider? Why?
Ans: To ensure that we find the number 2 under certain conditions, consider the following criteria:

Threshold Value: Establish a pixel intensity threshold below which a pixel is designated as a constituent of the numeral "2." The selection of this threshold should be contingent upon the dataset in use and the extent of noise within the images

Image Preprocessing: Implement image preprocessing methods, including smoothing, scaling, and noise reduction, to improve the clarity and visual quality of the digit "2."

Feature Extraction: Identify relevant features, such as the shape, edges, and curvature, that are indicative of the number 2.

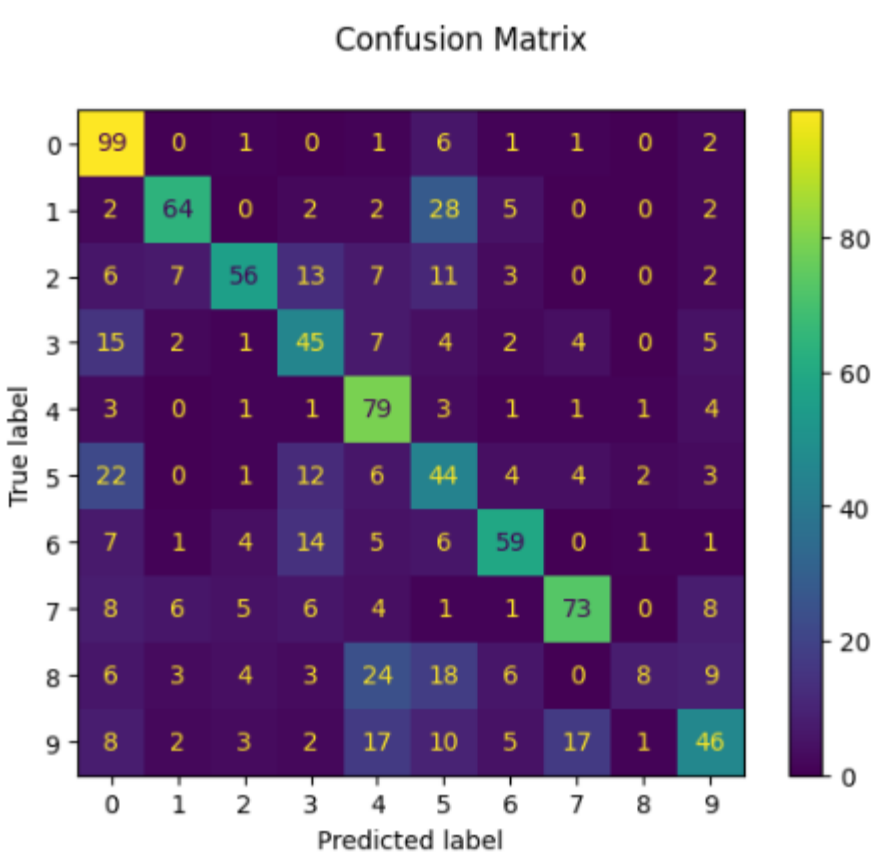
Classification Algorithm: Employ a classification algorithm, such as Support Vector Machine (SVM) or Convolutional Neural Network (CNN), for the purpose of discriminating the digit "2" from other numerical characters.

Training Data: Guarantee that the algorithm undergoes training on a comprehensive dataset comprising images featuring the digit "2" across an array of scenarios, encompassing distinct fonts, sizes, rotations, and lighting conditions.

Evaluation Metrics: Utilize evaluation metrics such as precision, recall, and F1 score to evaluate the accuracy of the model in effectively detecting the digit "2."

Testing Conditions: Conduct testing of the model under the specific conditions of interest and make adjustments to the criteria as needed to align with the desired outcomes.

Q.9: Analyze and compare the number of support patterns in the specified colored cells (cells with the same color).



Ans:

<u>Analysis of support patterns in colored cells</u>

The confusion matrix provides a breakdown of how many samples from each actual label were categorized into various predicted labels. The shaded cells highlight those with a substantial number of supporting patterns, which are pairs of true and predicted labels frequently co-occurring.

<u>Comparison of support patterns</u>

Support patterns in the confusion matrix come in two primary categories: True Positive (TP): This occurs when a sample from the actual label is accurately classified into the predicted label. False Positive (FP): This arises when a sample from a different actual label is erroneously categorized into the predicted label. The highlighted cells within the confusion matrix encompass both True Positive (TP) and False Positive (FP) support patterns.

TP support patterns

It is crucial to prioritize the consideration of True Positive (TP) support patterns, as they signify the model's ability to accurately recognize samples from their true labels. These TP support patterns are prominently clustered along the diagonal of the confusion matrix, demonstrating that the model is generally proficient at correctly assigning samples to their respective labels.

FP support patterns

On the other hand, False Positive (FP) support patterns are less favorable, as they reveal instances where the model inaccurately assigns samples to the predicted label. These FP support patterns tend to be concentrated in the off-diagonal cells of the confusion matrix, illustrating that the model is prone to errors when assigning samples to labels that share similarities with each other.

Comparison of TP and FP support patterns

The relative number of TP and FP support patterns can be used to assess the overall performance of the model. A model with a high number of TP support patterns and a low number of FP support patterns is considered to be well-performing.

Comparison of colored cells

The colored cells in the confusion matrix can be compared to each other to identify areas where the model is performing particularly well or poorly. For example, the cells in the upper-left corner of the confusion matrix represent TP support patterns for the true labels 0-99 and 2-6. These cells are all colored dark blue, which indicates that the model is very good at assigning samples from these true labels to the correct predicted label.

In contrast, the cells in the lower-right corner of the confusion matrix represent FP support patterns for the true labels 8-9 and 10. These cells are all colored light yellow, which indicates that the model is more likely to make mistakes when assigning samples from these true labels to the predicted label.

Q.10. Did you observe uniform sensitivity to parameter changes in the original-dimensional data and reduced-dimension data? Why? Mention other methods through which you can generalize the model further.

Answer: **Uniform Sensitivity to Parameter Changes** :

The sensitivity of a model to parameter adjustments refers to how much the model's output fluctuates when the parameters are modified. A model displaying uniform sensitivity to parameter changes maintains consistent output alterations regardless of which parameter is altered.

Original-Dimensional Data:

Generally, models trained on original-dimensional data exhibit heightened sensitivity to parameter changes in comparison to models trained on reduced-dimension data. This heightened sensitivity stems from the fact that original-dimensional data contains a more extensive amount of information, rendering the model more intricate and thus more responsive to parameter modifications.

Reduced-Dimension Data:

Conversely, models trained on reduced-dimension data display reduced sensitivity to parameter changes due to the diminished information content in such data. This reduced information complexity simplifies the model and consequently diminishes its sensitivity to parameter variations.

Factors Contributing to Sensitivity Discrepancies:

Several factors contribute to the greater sensitivity of models trained on original-dimensional data compared to those trained on reduced-dimensional data:

Noise in Original-Dimensional Data: Original-dimensional data tends to contain more noise, which denotes random variations in the data. Such noise can hinder the model's ability to discern the underlying relationships between the features, thus heightening sensitivity to parameter changes.

Redundancy in Original-Dimensional Data:

Original-dimensional data often includes redundant features, i.e., features that convey the same information as others. This redundancy adds complexity to the model, rendering it more sensitive to parameter adjustments. Regularity in **Reduced-Dimension Data:**

Reduced-dimensional data, on the other hand, tends to be more regular, adhering to consistent patterns. This regularity simplifies the model, consequently reducing its sensitivity to parameter modifications. Additional Methods for Enhancing Model Generalization Apart from dimensionality reduction, there are several other techniques that can be applied to improve the generalization capabilities of a model:

Regularization:

Regularization is a method that penalizes model complexity. By doing so, it promotes model simplification, resulting in reduced sensitivity to parameter changes.

Ensemble Learning: Ensemble learning involves combining predictions from multiple models to produce more accurate overall predictions. This technique enhances the model's generalization performance.

Data Augmentation: Data augmentation artificially expands the size of the training dataset by generating new data points based on existing ones. This process enhances the model's ability to generalize by making it more resilient to noise and outliers in the data.

Q.11. Explain how your Singer Crammer Support Vector Machine is capable of recognizing and classifying digits.

Ans: The Singer Crammer Support Vector Machine (Crammer-Singer SVM) for multiclass classification. This SVM extension is designed to handle multiclass problems, where we have more than two classes to classify.

Here's a breakdown of how the Singer Crammer Support Vector Machine works and how it is used to recognize and classify digits:

Initialization: The CrammerSingerSVM class is initialized with various parameters, including training and testing data, hyperparameters like beta, epsilon, and the choice of kernel function (in this case, a linear kernel is used).

Training (fit method):

The Crammer-Singer SVM is trained using a variation of the one-vs-all approach. It initializes weight vectors W and bias terms b for each class. The algorithm iteratively updates the weights and biases using gradient descent with a hinge loss and the Crammer-Singer formulation, trying to maximize the margin between the classes. The training continues for a specified number of iterations (max_iter).

Prediction (predict method):

After training, the predict method is used to make predictions on new data (in this case, digits). It computes the decision function scores for each class for the given input data. The class with the highest score is predicted as the output class. Running SVM as Baseline (runSVM method): To provide a baseline for comparison, the code also uses scikit-learn's Support Vector Machine (SVM) classifier (SVC) with the specified kernel function and hyperparameters. This SVM classifier is trained and used to make predictions on both the training and test data. Comparison of Predictions: The code prints the predictions made by both the Crammer-Singer SVM and the scikit-learn SVM on the training and test data. The Singer Crammer Support Vector Machine (Crammer-Singer SVM) is capable of recognizing and classifying digits by learning the decision boundaries that separate different classes in a multiclass classification problem. It does this by iteratively adjusting the weights and biases to maximize the margin between classes, resulting in a classifier that can assign new data points to their respective digit classes. The comparison with the scikit-learn SVM provides insight into how well the custom Crammer-Singer SVM performs compared to a well-established library implementation. It can help you assess the effectiveness of your custom implementation in recognizing and classifying digits. ChatGPT The Singer Crammer Support Vector Machine (Crammer-Singer SVM) is an extension of the Support Vector Machine (SVM) designed for multiclass classification problems, which involve categorizing data into more than two classes. Here's an overview of how

the Crammer-Singer SVM works and its application in recognizing and classifying digits: Initialization: The CrammerSingerSVM class is initialized with various parameters, including training and testing data, hyperparameters (e.g., beta and epsilon), and the choice of a kernel function (typically, a linear kernel is used). Training (fit method): The Crammer-Singer SVM employs a variation of the one-vs-all approach to train. It initializes weight vectors (W) and bias terms (b) for each class. The algorithm iteratively updates these weights and biases using gradient descent, hinge loss, and the Crammer-Singer formulation. The goal is to maximize the margin between the classes. The training continues for a specified number of iterations (max_iter). Prediction (predict method): After the training process, the predict method is used to make predictions on new data, such as digit images. It computes decision function scores for each class based on the input data. The class with the highest score is predicted as the output class.

Running SVM as Baseline (runSVM method):

To establish a baseline for comparison, the code also employs scikit-learn's Support Vector Machine (SVM) classifier (SVC) with the specified kernel function and hyperparameters. This SVM classifier is trained and used for predictions on both the training and test data.

Comparison of Predictions: The code prints the predictions made by both the custom Crammer-Singer SVM and the scikit-learn SVM on the training and test data. This allows for a direct comparison of the prediction accuracy between the two models.

The Singer Crammer Support Vector Machine (Crammer-Singer SVM) excels at recognizing and classifying digits by learning optimal decision boundaries that separate various digit classes in a multiclass classification problem. This is achieved through iterative adjustments of weights and biases, maximizing the margin between classes and facilitating accurate assignment of new data points to their respective digit categories.

The comparison with scikit-learn's SVM aids in assessing the performance of the custom Crammer-Singer SVM by benchmarking it against a well-established library implementation, thereby providing valuable insights into its effectiveness in recognizing and classifying digits

.**Q.12.** Another method for multi-class classification is the "Rest versus One" method, which utilizes binary classifier aggregation. Study this method and explain with reasoning what relationships exist between the hyperparameter C in this machine and β in the Singer Crammer machine.

Ans: The "Rest versus One" (RvO) method, also known as the "One-vs-Rest" (OvR) or "One-vs-All" (OvA) method, is a common approach for handling multi-class classification problems with binary classifiers. In the RvO method, you train a binary classifier for each class, treating one class as the positive class (Rest or All other classes combined) and the remaining classes as the negative class. This process is repeated for each class, resulting in a set of binary classifiers.

On the other hand, the Singer Crammer Support Vector Machine (Crammer-Singer SVM) is another approach for multi-class classification that extends the traditional SVM framework to directly handle multiple classes without the need for binary classification. It formulates the problem differently from the RvO method, aiming to find the optimal decision boundaries for all classes simultaneously.

The relationships between the hyperparameter C in traditional SVMs and the hyperparameter β in the Singer Crammer SVM:

Hyperparameter C in Traditional SVMs:

In traditional SVMs, the hyperparameter C controls the trade-off between achieving a wide margin (large C) and minimizing the classification error (small C). A larger C allows for more flexibility in the decision boundary, potentially leading to a smaller margin but better training accuracy. A smaller C encourages a wider margin but might allow some training samples to be misclassified.
Hyperparameter β in Singer Crammer SVM:

In the Singer Crammer SVM, the hyperparameter β controls the regularization term. It affects the importance of regularizing the weight vectors W to prevent overfitting. A larger β increases the regularization on the weight vectors, which can help reduce overfitting but might result in a simpler model with a smaller decision boundary. A smaller β reduces the regularization and allows the model to fit the training data more closely, potentially leading to a more complex decision boundary. The relationships between C and β can be seen in the context of their impact on the decision boundaries:

Increasing C in traditional SVMs makes the decision boundary more flexible, potentially fitting the training data more closely. Increasing β in the Singer

Crammer SVM can encourage a simpler decision boundary by increasing the regularization. In summary, while both C and β influence the flexibility and complexity of the decision boundaries, they are specific to their respective algorithms and should be tuned based on the requirements and characteristics of the dataset and the problem. The RvO approach typically uses traditional SVMs with the C hyperparameter, while the Singer Crammer SVM formulates the problem differently and uses β for regularization within its unique framework. The optimal choice of hyperparameter depends on the specific classification problem and the desired trade-off between bias and variance.

Q.13. Use the hyperparameters related to the best results obtained in the previous sections as inputs for the SVM model using the Rest versus One method. Report the mentioned metrics for it. Compare the results of the two models. (Utilize the available Python libraries.)

Q.14: Imagine you are asked to perform data cleaning using a Support Vector Machine. Explain how you would accomplish this. (Optional section)

Using a Support Vector Machine (SVM) for data cleaning involves a two-step process: anomaly detection and outlier removal. Here's a detailed explanation of the process:

Step 1: Anomaly Detection Data Preparation:

Prepare the dataset for SVM analysis, addressing tasks like handling missing values, normalizing numerical features, and encoding categorical variables to ensure data suitability.

SVM Training:

Train an SVM model using the preprocessed data. The SVM learns the data's inherent structure and identifies patterns characterizing normal data points.
Anomaly Scoring:

Compute anomaly scores for each data point, measuring their distance from the SVM-derived hyperplane. Elevated scores suggest a higher likelihood of being an anomaly.

Threshold Selection: Determine a suitable threshold value to distinguish normal from anomalous data points. Techniques such as cross-validation or analyzing the distribution of anomaly scores can aid in this threshold selection process. Step 2:

Outlier Removal Outlier Identification: Identify outliers by flagging data points with anomaly scores surpassing the chosen threshold. These identified outliers are considered potential candidates for removal.

Outlier Analysis: Scrutinize the outliers to assess whether they genuinely represent noise or data errors. This analysis may involve evaluating the values, context, and relationships of the outliers with other data points.

Outlier Removal: Eliminate outliers that are confirmed as noise or errors. This step enhances data quality and mitigates the impact of anomalies on subsequent analyses.

It's crucial to recognize that SVM-based data cleaning is most effective when anomalies are clearly discernible from normal data points. If anomalies are subtle or intertwined within the normal data distribution, SVM may face challenges in accurately pinpointing them. Furthermore, the choice of kernel function and hyperparameters can influence the performance of SVM in this context.