**Week 2 assignment**

**Svm**

When using the Support Vector Machine (SVM) regressor, different hyper-parameters. such as kernel type (“linear” or “rbf”), along with varying values for C and gamma, can significantly impact the model’s performance. Here’s a summary, analysis, and key findings when experimenting with these hyper-parameters.:

**Kernel = “linear” with various values for the C hyper-parameter:**

- The linear kernel in SVM aims to create a linear decision boundary. Altering the C hyper-parameter controls the trade-off between maximizing the margin and minimizing the classification error.

Analysis and Key Findings:

- Higher values of C indicate less regularization, potentially leading to over-fitting, while lower values may cause under-fitting.

- Increasing C can make the model focus more on individual data points, which might not generalize well to unseen data.

- The model might perform well on linearly separable data with an optimal C value, but might struggle with non-linearly separable datasets.

**Kernel = “rbf” with various values for the C and gamma hyper-parameters.:**

- The radial basis function (RBF) kernel is capable of modeling complex, non-linear relationships by mapping data into a higher-dimensional space.

Analysis and Key Findings:

- C controls the trade-off between smooth decision boundaries and correctly classifying training points.

- Gamma determines the influence of a single training sample, affecting the shape of the decision boundary.

- Higher values of gamma can lead to over-fitting by creating overly complex decision boundaries that fit the training data too closely.

- A smaller gamma tends to produce smoother decision boundaries, but may under-fit the data by oversimplifying the model.

**Overall Observations:**

- Model performance heavily depends on the choice of hyper-parameters.

- Grid search or randomized search techniques can aid in finding the optimal combination of hyper-parameters. by cross-validating on different parameter values.

- Regularization (controlling C) is crucial to prevent over-fitting, especially when using non-linear kernels like RBF.

- The RBF kernel, with careful tuning of C and gamma, can handle more complex relationships compared to the linear kernel.

Key Takeaways:

- For linearly separable data with a simple relationship, a linear kernel with an appropriately chosen C might suffice.

- For complex, non-linear relationships, the RBF kernel with a well-tuned combination of C and gamma tends to perform better.

- Careful cross-validation and hyper-parameter tuning are essential to find the best-performing model configuration.

**Knn**

Building a K-Nearest Neighbors (KNN) classifier that achieves 97% accuracy on the MNIST dataset involves specific considerations due to the nature of KNN as a simple yet effective algorithm for classification. Here’s a summary, analysis, and key findings for achieving such accuracy with the KNN classifier:

**Summary:**

1. Data Preparation:

- MNIST dataset preprocessing involves loading the images and reshaping them into a suitable format for KNN, typically flattening the 28x28 pixel images into a single 1D array of 784 features.

- Normalizing pixel values: Scaling the pixel values to a range between 0 and 1 can improve KNN’s performance.

2. Model Building:

- Using the KNN algorithm: KNN is a non-parametric and lazy learning algorithm that classifies new instances based on majority class among its k-nearest neighbors.

- Choosing an appropriate value for ‘k’: Experimenting with different values of ‘k’ to find the optimal balance between bias and variance. Too low ‘k’ can lead to over-fitting, while too high ‘k’ can introduce under-fitting.

3. Hyper parameter Tuning:

- Finding the optimal ‘k’: Performing grid search or cross-validation to determine the best ‘k’ value that maximizes accuracy on the validation set.

- Distance metric selection: Choosing an appropriate distance metric (e.g., Euclidean, Manhattan) can significantly impact the performance of KNN.

4. Model Evaluation:

- Splitting the dataset into training and test sets: Using a separate test set to evaluate the model’s performance on unseen data.

- Assessing accuracy and potentially other metrics: Understanding the classifier’s performance not only in terms of accuracy but also precision, recall, and F1-score, especially for individual classes.

**Analysis and Key Findings:**

1. Impact of ‘k’ value:

- The choice of ‘k’ affects the bias-variance trade-off. Smaller ‘k’ values result in a more complex model prone to noise, while larger ‘k’ values may oversimplify the decision boundary.

- The optimal ‘k’ value may vary based on the dataset and its characteristics.

2. Computational Complexity:

- KNN’s prediction time increases as the dataset size grows since it requires comparing the new instance with all training samples.

- Managing larger datasets might lead to longer prediction times.

3. Sensitive to Distance Metrics:

- The choice of distance metric impacts the classification accuracy. Experimenting with different distance measures can lead to varying performance.

Key Takeaways:

- KNN is a simple yet effective algorithm for classification, but its performance heavily depends on the choice of ‘k’, distance metrics, and dataset characteristics.

- Tuning ‘k’ and selecting an appropriate distance metric are crucial for achieving higher accuracy.

- KNN might struggle with larger datasets due to its computational complexity during inference.