Comparing SVM (Support Vector Machine) and KNN (K-Nearest Neighbous in MNIST Image Classification

Description and Motivation

This is a Multiclassification problem, where we assign each image in the dataset to a specific number from 0 to 9. This dataset is sourced from Yann-Lecun's website. In this project, we will:

- Build two models for a multiclassification task using supervised learning algorithms SVMs and KNNs
- Perform hyper-parameter tuning then compare and analyse the performance using binary classification performance metrics of both of these models to determine the best one
- Compare results with studies done by Agarwal, A.K (2021) and Ghadekar, P. (2018)

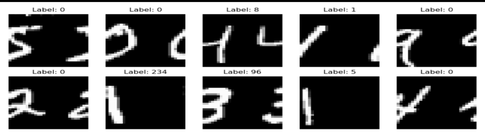
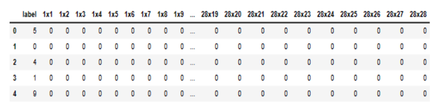


Figure 1: MNIST Images

Data Pre-Processing, Data Cleaning and EDA

- MNIST (Modern National Institute of Standards and Technology Database) is commonly used to train image processing systems. We source this dataset from the yann.lecun website who is one of "the three musketeers" of deep learning and was heavily involved in research with Computer Vision. There are 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau. It is known as the "Hello World" of Machine Learning and is used to test new classification algorithms, especially within Computer Vision. These images are all have dimensions of 28x28.
- We convert the dataset from the original format of a "gzip" file to a "csv" file to make it easier to use. We merge the images to the corresponding labels and split the data consisting of 70,000 images and labels into 60,000 images and labels in the training set, and 10,000 images and labels in the test set. We then split this into X_train, X_test, y_train and y_test with the "test" data being the label which is the target variable (what we are trying to predict) and the training data to be the images. There are 784 columns (28 x 28) which all consist of multiple numpy arrays which pertain to the image data in X_train and y_train.
- There are no null values, or corrupt images within this dataset. We do not need to filter out the images by resolution. Hence, we can move onto EDA and building machine learning models.
- Each number corresponds to a pixel you can see within the image. As you can see, there are plenty of different values, ranging from 0—254. Each value corresponds to a pixel within the image, with a higher number representing a higher pixel internsity. The numbers presented in figure 4 is just one image example of 70,000 images. Different numbers will have different pixel intensities and therefore different numbers when reshaped into a 28 x 28 image.
- Figure 5 shows the proportion of classes within the image dataset. We can see that there is no class imbalance, with only a 2.23% difference between the maximum and minimum classes (11.25% compared to 9.02%), so the results will be unbiased.
- We can see from figure 3, an example of how the image data has been reshaped into a dataframe, with each column representing a row of pixels.
- From Figure 2, we can see that the training and testing datasets have similar proportions of classes, reducing bias. There is no class imbalance. The class with the lowest number of images is 5, and the class with the highest number of images is 1.



label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	5	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 4: Data Frame snippet

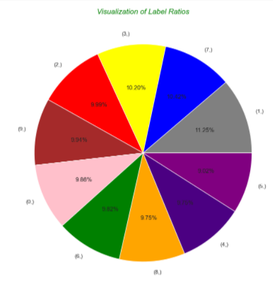


Figure 5: Pie Chart Showing Ratios of different Labels

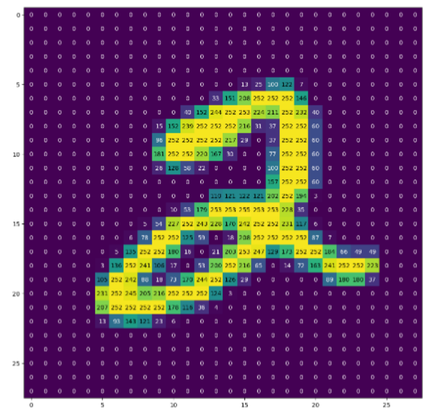


Figure 3: How Image Data can be reshaped into a dataframe.

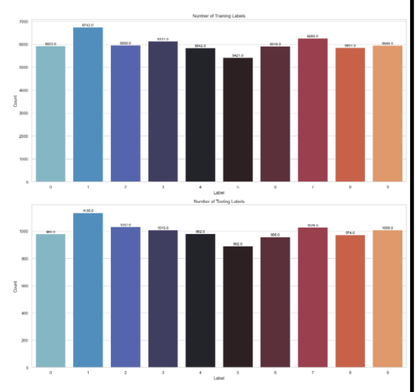


Figure 2: Count Plots showing Label statistics for training and test sets

K-Nearest Neighbours (KNN) Algorithm

The KNN (K-Nearest Neighbours) algorithm is a simple and intuitive machine learning algorithm which can also be referred to as an instance-based learning/ lazy learning algorithm as it does not build a traditional model during training. It stores data directly within memory and then uses it for prediction.

K is the hyperparameter indicator. A small K is where the algorithm pays more attention to the local patterns within the data. This can lead to a high cariance so predictions are more sensitive to noise within the data resulting in overfitting. A large K will result in a lower variance and a more stable model, but it might lead to underfitting by oversimplifying decision boundaries.

Here is a step-by-step method:

1. **Training Phase**: This involves storing the entire set of data within its memory
2. **Prediction Phase**: Given a new, unseen image, the algorithm identifies the k-nearest neighbour from the training dataset based on a distance metric, which measures how similar these two images are (**Euclidean** distance and **Manhattan** distance)
3. **Distance Classification**: For each training image and new image, the algorithm calculates the distance between pixel values with a chosen distance metric. Smaller distances indicate greater similarity between images.
4. **Voting Classification:** The K-NN performs a majority cote among the labels of the k nearest neighbours with the label appearing most frequently among the k neighbours being assigned to the predicted class label for the new image.

Advantages

- **Simplicity:** It is a very simple algorithm
- **Small Hyperparameter tuning:** Can choose which distance metric you want to use within the function (only one hyperparameter K needs to be tuned)
- **No training phase:** No training time needed for classification models
- **Learning:** Constantly evolves under new data
- **Non-Linear Capability:** Can learn non-linear decision boundaries for classification. It can also consider local patterns and does not need to be modified
- **Flexible for multiclass classification:** KNN can handle the 10 classes (digits 1—9) without modification

Disadvantages

- **Computational Complexity:** Determining K Values can lead to losses and MNIST has a large number of data points, 60,000 and KNN's prediction time grows with the dataset size; this can be **time-consuming** and **expensive** for large datasets
- **Curse of Dimensionality:** High dimensionality can lead to the "curse of dimensionality" which will affect the KNN
- **Optimal K Selection:** It is difficult to predict the correct value of K.
- **Sensitive to Noise:** MNIST digits have variations and noise which KNN is sensitive too and may incorrectly classify' sensitive to outliers
- **Extensive Feature Engineering and Feature Selection Required**

Support Vector Machine (SVM) Algorithm

SVM works by trying to find the optimal hyperplane which best separates the data points of different classes in a feature space. It converts the data points to a higher dimensional space (Kernel Trick) by implicitly computing the dot product (similarity) between the data points. We then map this 2D dataset into a 3D space and find a hyperplane which best separates the transformed data points before mapping it back down to the original 2D dimensional space using an inverse transformation. This results in a non-linear decision boundary. The algorithm features **support vectors** which are the points closest to the non-linear decision boundary and the **margin** is the distance between the former and latter.

In some cases, there are misclassifications, especially with data that has a higher number of classes as within the MNIST dataset, so we introduce a soft boundary with the **C** hyperparameter controlling the trade-off between maximising the margin and minimising the classification error. A smaller C value will allow for more missclassifications and a wider margin, but a larger C will minimise missclassifications at the cost of a potentially narrower margin.

We can also modify the **Kernel Function** which maps the input features into a higher-dimensional space. Options include the **linear** which applies no transformation and is used for linearly separable data, **polynomial** which maps the data into a polynomial space, **RBF (Radial Basis Function)** creating a Gaussian-Like transformation. These map input features into a higher-dimensional space to cater for linear separation (this may not be possible for nonlinear separation).

1. Input data points into the SVM model.
2. SVM model uses kernel trick to take input points into higher dimensional space computing the dot product of data points
3. A soft decision boundary is introduced to allow for missclassifications and reduce overfitting (C parameter)
4. A hyperplane is produced separating the points as best as possible, with support vectors being closes to the boundary.

Advantages

- **Effective in high-dimensional spaces:** SVM kernel functions can hangle higher dimensional data like MNIST images.
- **Regularisation: C** parameter allows you to control overfitting achieving good generalisation
- **Multiclass Classification:** Suitable for multiclass classification
- **Outlier Robustness**: Less sensitive to this as they rely on support vectors
- **Global Optimal Solution**: SVM optimisation aims to find a global solution, which is advantageous for achieving a well-fitting model.
- Can learn non-linear decision boundaries for classification

Disadvantages

- **Computationally Intensive:** There are a large number of samples and features, and this can be computationally expensive.
- **Data Scaling** : SVM's performance can be sensitive to data scaling. MNIST images might need pre-processing.
- **Kernel and Paramter Selection** : Highly expensive and time-consuming configuration needs to be done to find the best configuration.
- **Imbalanced Classes**: Class distributions are imbalanced in MNIST, causing SVMs performance to be affected
- **Parameter Sensitivity**: Sensitive to parameter choices, requiring careful hyperparameter tuning

Hypothesis Statement

Hypothesis Statement 1: My prediction is that the KNN model will perform better than the SVM model, because the problem is simpler, the MNIST digits are well-separated. [2] agrees with this, with the SVM (96.50 accuracy) compared to the KNN (97.00 accuracy) on the test set. [3] disagrees with 97.74% compared to 97.33% for SVM and KNN respectively

Hypothesis Statement 2: KNN will have a quicker training time and hyper-parameter tuning time as it has less hyper-parameters and is easier to train.

Experimental Results, Choice of Parameters

KNN Algorithm

- We fit the KNN algorithm on the training data, with the Euclidean distance metric (default) and k = 3
- After testing, the template KNN model had an accuracy of 0.931, and a Cohen's Kappa coefficient of 0.9178
- After optimisation, we test our final KNN model on test data and record results

Choice of Parameters

- The optimal K hyperparameter after optimisation using gridsearch was 1

Support Vector Machines (SVM)

- We train our template model with a Box Constraint Parameter of 0.1 and a Kernel Function "linear"
- After testing, the template SVM model has an accuracy of 0.842, and a Cohen's Kappa coefficient of 0.8245
- After optimisation, we test the final model on Unseen Data and record our final results

Choice of Parameters

- The optimal box constaint of 0.1 and a linear Kernel function were the optimal hyperparameters.

Confusion Matrices and Results Tables

Metric	KNN (K-Nearest Neighbours)	SVM (Support Vector Machine)
Precision	0.9336	0.8386
Recall	0.9297	0.8375
F1-Score	0.9304	0.8373

Figure 12: Precision, F1 and Recall Scores

C (box—Constraint)	Kernel Function	Accuracy (one vs all)
0.01	Linear	0.837
0.01	Polynomial	0.111
0.01	RBF	0.111
0.1	Linear	0.842
0.1	Polynomial	0.111
0.1	RBF	0.111
1.0	Linear	0.842
1.0	Polynomial	0.111
1.0	RBF	0.111
10	Linear	0.842
10	Polynomial	0.111
10	RBF	0.111

Figure 8: SVM (Support Vector Machines) Hyperparameter Tuning

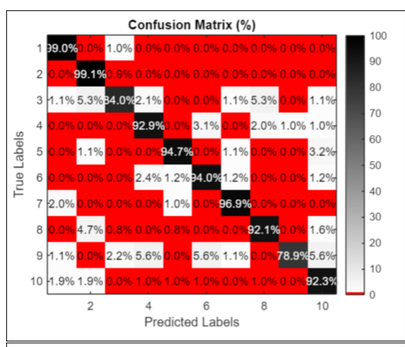


Figure 6: Confusion Matrix for KNN

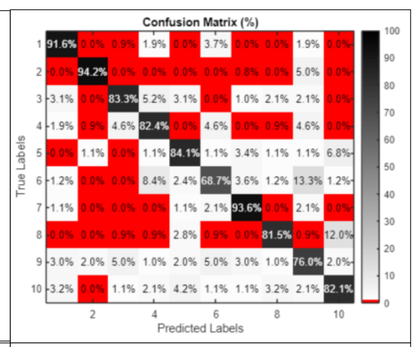


Figure 7: Confusion Matrix for SVM

K (Number of	Accuracy
1	0.931
3	0.926
5	0.928
7	0.924
9	0.925

Figure 9: KNN (K-Nearest Neighbours) Hyperparameter Tuning

ML Model	Cohen-Kappa
KNN Model	0.9178
SVM Model	0.8244
Optimised KNN Model	
Optimised SVM	

Figure 10: Cohen Kappa Coefficients

Model	Training Speed
KNN Model	53.4
SVM Model	346.2

Figure 11: Model Speeds (s)

Methodology

1. There are 70000 images within this dataset, and 70,000 corresponding labels ranging from 0-9. We initially download the gzip files, and convert them into csv format to make it easier for data cleaning, data transformation, EDA and machine learning model analysis.
2. We split the data into training and test sets, each with their respective labels. The split is done so there are 60,000 images with corresponding labels in the training set and 10,000 images with corresponding labels within the testing set.
3. We check this data for corrupted images, before splitting the target variable (label) into y_train and y_test and the image data stored as numpy arrays into x_train and x_test datasets.
4. We further split these datasets to end up with 5000 training images and labels (x_train, y_train) and 1000 testing images and labels (x_test, y_test) because of the time and processing power required to train and optimise models on 60,000 images.
5. We train the models on training images and labels. The K hyperparameter for the KNN is set to 3, and the SVM hyperparameters are C = 1 and kernel = "Linear".
6. We run these models on the testing images and record accuracies between predictions of the labels and the actual labels (y_test). We compute the confusion matrix of these models
7. We the optimise both models on the training data, with the hyper parameters shown in Figure 8 and 9, recording which hyper-parameter and hyper-parameter combination gives us the highest accuracy.
8. We measure and compare performance between original and final optimized models. The final optimized models are the end-product.
9. We compare and critically evaluate which of the two final models is better suited for this dataset.

Analysis and Evaluation of results:

- From our confusion matrix diagonal elements the diagonal elements representing **true positives** (top—left to bottom-right), we can see from Figure 7 that 68.7% of Class 6 labels were predicted accurately by the SVM, so Label 6 is the lowest accurately predicted class for the SVM, followed by Label 9 at 76.0%. Since 6 and 9 are the most similar numbers out of the classes, this may suggest that due to their similarity, pixel values might be identical in certain parts of the 2D-arrays, highlighting a possible SVM weakness in differentiating locations of similar pixel intensities
- Looking at Figure 6, featuring our confusion matrix for the KNN model, we see that Label 9 is predicted the least accurately at 78.9%, followed by Label 3 predicted at 84.0%. All other labels are predicted above 90% accurately, with Label 1 and Label 2 at 99.0% and 99.1% respectively.
- There are a significant number of off-diagonal elements (missclassifications), involving numbers which are outside the diagonal representing instances which are misclassified for the SVM. This is depicted in the colour-coding, with "white" representing misclassifications, and red representing a very small amount/ no (1.0% or below) misclassifications. The proportion of white to red in Figure 7 is 51:39 compared to Figure 6 which is 30:60 respectively. This suggests the SVM misclassifies much more than the KNN model and this can be seen across all the labels (due to the distribution of the colours in the confusion matrix) so we conclude KNN is more accurate. Label 6 and Label 8 are misclassified above 10% of the time, 13.3% and 12.6% as Label 9 and Label 10. The largest error we can see in the KNN, SVM model is that 5.6% of Label 9s are misclassified as Label 10s.
- The KNN model has a higher Cohen-Kappa coefficient indicating a very strong level of agreement between the predictions of the KNN model and actual classes in the MNIST dataset. Hence, KNN model's predictions are consistent and reliable. On the other hand, SVM's Cohen Kappa coefficient of 0.8244 indicates a substantial level of agreement. Comparing these two coefficients, we see that the KNN model's higher coefficient suggests it performs better in terms of consistency and agreement with true classes compared to the SVM model.
- As we can see from Figure 11, the KNN model was much quicker to train than the SVM model on the data, due to the simplicity of the algorithm. The KNN algorithm has a low K hyper-parameter(1) hence its training time is quick since it does not need to perform computations on the entire dataset. However, SVM's training time is much higher due to the size of the dataset and the time it takes to find the optimal hyperplane maximising the margin between different classes.
- After iterating through different combinations of hyperparameters as shown in Figure 8 and Figure 9, we can see for SVM that the only applicable Kernel Function was the Linear Function because of the nature of the data, with each image representing a handwritten digit, using a linear kernel was effective because the data is linearly separable, so a linear decision boundary is enough. Using other Kernels will over-complicate the hyperplane in feature space. A small C value (0.1) is better because it encourages a larger margin leading a model that generalises well to unseen data. For the KNN model, k=1, considered a "pure" version of the algorithm, where the class of a data point is determined by the class of the nearest neighbour. Using K = 1 can mitigate the curse of dimensionality, and it can capture fine-grained differences between high intra-class variability.
- The KNN outperforms the SVM across all three metrics (precision, recall, F1 Score), suggesting it is better at achieving overall balance between positive instances and minimises false positives and negatives. KNN having a higher precision indicates that it predicts positive classes more accurately, with a higher recall suggesting it is less likely to miss instances of the positive class and pick up false negatives more.
- From our results we can conclude that the **KNN is the superior model with a higher predictive power**. We can confirm that both **Hypothesis 1** and **Hypothesis 2** are correct. This is different to [2] A.K Agrawal who found SVM outperformed the KNN in their study but agrees with [3] P Ghadekar's results

Lessons Learned:

- For KNN, more feature selection could be done since there are some factors that affect the results more than others.
- We can look into different methods of SVM learning to perhaps give us a better understanding and insight as to why it was only able to predict one class accurately. "onevsall" does not do well for this dataset
- Image data takes a long time to process, and dimensionality reduction techniques can be employed to make this process quicker, despite reducing model accuracy. It is better to use Google Cloud/ AWS/ Azure to store and train models on this data.

Future Work:

- Look into using dimensionality-reduction techniques, such as PCA (Principal Component Analysis), LDA (Linear Discriminant Analysis), NCA (Neighbourhoods Component Analysis) and t-SNE
- We can evaluate other models, such as Logistic Regression, Naïve Bayes, Random Forests and also consider Neural Networks. Yann Lecun (1998) published work on a LeNet-5 architecture which is a CNN that outperformed other CNNs which outperformed traditional ML models.
- Further hyper-tuning parametrization can be done on SVMs, experimenting with different learning methods ("onevsall", "ECOC", "DECOC", "OvO") and then for KNN's we can also look at the optimal decision metric
- We can consider looking at decision trees, and can also consider pruning our final decision tree or consider boosting/ bagging to improve the performance

References

[1] Baldominos, A., Saez, Y. and Isasi, P., 2019. A survey of handwritten character recognition with mnist and emnist. *Applied Sciences*, *9*(15), p.3169.

[2] Agrawal, A.K., Shrivas, A.K. and kumar Awasthi, V., 2021, May. A Robust model for handwritten digit recognition using machine and deep learning technique. In *2021 2nd International Conference for Emerging Technology (INCET)* (pp. 1-4). IEEE.

[3] Ghadekar, P., Ingole, S. and Sonone, D., 2018, August. Handwritten digit and letter recognition using hybrid dwt-dct with knn and svm classifier. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)* (pp. 1-6). IEEE.

[4] Wu, M. and Zhang, Z., 2010. Handwritten digit classification using the mnist data set. *Course project CSE802: Pattern Classification & Analysis*, *366*.

[5] Zhang, H., Berg, A.C., Maire, M. and Malik, J., 2006, June. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (Vol. 2, pp. 2126-2136). IEEE.