A Comparative Study of Machine Learning Algorithms for Fashion-MNIST Image Classification

COMP5318 Assignment 1

Tianlan Tang, HongFei Zhao
540446740, 510136736

I. Introduction

The dataset for this assignment is sourced from Fashion-MNIST, comprising 28x28 grayscale images. The dataset is categorized into 10 different classes, including T-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. It is split into a training set of 30,000 samples and a test set of 5,000 samples. Each sample is labeled, and the objective of this report is to identify the most effective model for image classification.

To accomplish this, four machine learning models were implemented and compared: k-Nearest Neighbors (KNN), Logistic Regression, Support Vector Classifier (SVC), and Random Forest. Each model was evaluated based on key performance metrics, including accuracy, F1-score, training time, and inference time

Through extensive experimentation, the report reveals that each model has its strengths and trade-offs. For instance, SVC and Logistic Regression achieve high accuracy and F1-score. KNN and Logistic regression, though simple and intuitive, provide competitive results with proper hyperparameter tuning.

In conclusion, we found that the SVC model, with a high regularization parameter and a moderate kernel coefficient, offers the best balance between classification accuracy and computational efficiency for this specific image classification task.

II. Methodology

A. Preprocessing

a. Standardization

Principle: Standardization transforms the input data by subtracting the mean and dividing by the standard deviation, so that the data has a mean of zero and a standard deviation of one. In image classification tasks, this process is applied to each pixel, where the intensity of each pixel (ranging from 0 to 255) is standardized based on all the pixels across the dataset.

Rationale: Standardization ensures that all input features (pixel values) are on the same scale, which helps improve the convergence speed of selected machine learning algorithms, particularly the SVC model, which is sensitive to feature scaling. By standardizing the features, it ensures that no feature dominates the model due to larger magnitude values.

b. Random Horizontal Flip

Principle: Random horizontal flip is a data augmentation technique used in image processing. It flips an image horizontally (from left to right) with a certain probability. This technique helps by creating different versions of the original images, which increases the variability in the dataset. By doing this, it makes the model less likely to rely too heavily on specific features of the training data, leading to better generalization when making predictions on test data.

Rationale: This method helps the model generalize better by making it robust to horizontal orientation changes. For example, in this dataset, a shoe should be classified correctly regardless of whether it is facing left or right. In addition, random horizontal flip is often applied to the convolutional neural network to reduce overfitting on the training set.

c. Convolutional Neural Network (CNN) Feature Extractor:

Principle: A Convolutional Neural Network (CNN) (Lecun, Bottou, Bengio, & Haffner, 1998) is a type of deep learning model. It works by using convolutional filters that scan over input images to identify patterns like edges, textures, and more complex structures. After going through the convolutional and pooling layers, the model produces a feature map that highlights important details in the image. These features are then flattened and passed into a classifier to make the final prediction.

The network consists of three convolutional layers, described as follows:

- The first convolutional layer has an input channel of 1 and an output channel of 32, with a kernel size of 3, stride of 1, and padding of 1. This is followed by batch normalization, a ReLU activation function, max-pooling (kernel size of 2, stride of 2), and a 30% dropout layer for regularization.
- The second convolutional layer expands the number of channels from 32 to 64, with other parameters similar to the first layer. This layer also uses batch normalization, a ReLU activation function, max-pooling, and a 30% dropout layer.
- The third convolutional layer further expands the number of channels from 64 to 128, with similar parameter settings. However, this layer uses average pooling instead of max-pooling, and the dropout rate is increased to 40%.

Additionally, the network includes a residual module (He, Zhang, Ren, & Sun, 2016) that employs a single-channel convolutional layer with a kernel size of 1 and a stride of 4. It also includes batch normalization and an average pooling layer.

Finally, the classifier section consists of two fully connected layers. The first linear transformation takes the

output from the convolutional layers (128 x 3 x 3) and reduces it to 256 dimensions. After applying ReLU activation and 50% dropout, a final linear layer produces output for 10 categories.

The model's forward propagation involves adding the residual connection and the three convolutional layers' output, followed by passing through the fully connected layers for classification.

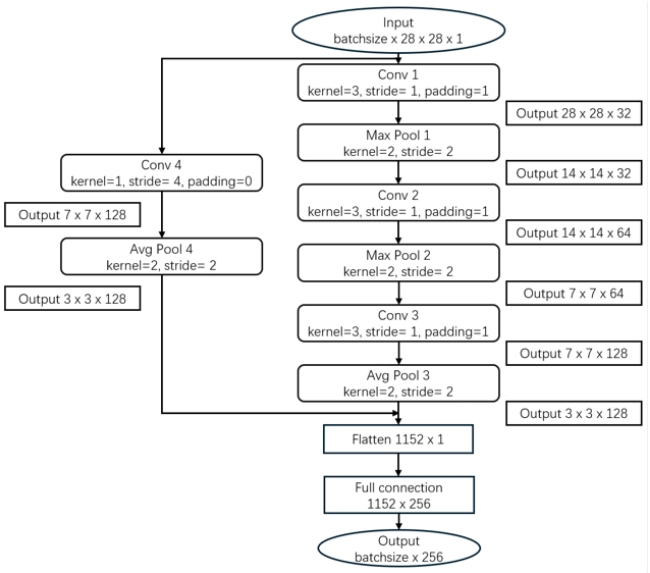

Figure 1 – Architecture of CNN model

The architecture of the model is shown in figure 1. In this assignment, the CNN model is first trained using the training data, after which the classifier is removed, and the convolutional layers are used as a feature extractor for subsequent training.

Rationale: Using a CNN as a feature extractor helps the model automatically learn important features from images. This is effective for image data because images have complex patterns and many dimensions. By using pre-trained CNN models, we can apply transfer learning to extract features that work well for classification tasks. This not only saves training time but also improves how well the model performs. After extracting these features, we can use traditional machine learning algorithms, like SVC or Random Forest, to make the final classifications.

B. Machine learning method

a. Support Vector Classifier (SVC) (Nguyen, 2024)

Principle: SVC aims at maximizing the margin between data points of different labels by find the optimal hyperplane that best separates the data. Through the selection of kernels (polynomial, Radial Basis Function), SVC can solve non-linear optimization problem. (Cortes & Vapnik, 1995)

The main hyperparameters of SVC are C (Regularization Parameter) and Gamma (Kernel Coefficient).

C controls the penalty for errors and balances the trade-off between correctly classifying samples and maximizing the margin. When the value is large, the model tends to correctly classify all training data, which may lead to overfitting. When the value is small, the model tends to underfitting.

Gamma is used for kernels, which controls the range of influence for individual training sample. When the gamma value is large, the model tends to memorize each training sample, which may lead to overfitting. When the gamma value is low, the model tends to underfitting.

Rationale: In an image classification task, the images themselves are high-dimensional data; for example, an image with a size of 28x28 has 784 dimensions. The SVC model can find complex decision boundaries between different image classes. In addition, SVC is sensitive to support vectors, and the presence of outliers can significantly affect the calculation of the hyperplane. For image data, especially the grayscale images in this task, pixel intensity is strictly limited to a range of 0-255, which reduces the likelihood of extreme values or outliers. This makes the SVC model particularly suitable for the current task.

b. Random Forest (Nguyen, 2024)

Principle: Random Forest is an ensemble method that combines multiple decision trees by taking the mean of the predictions from all the individual trees as its final prediction. Each tree is trained on a randomly selected subset of data and features, which helps improve robustness and prevent overfitting. (Leo, 2001)

The main hyperparameters of random forest are number of trees and maximum leaf node.

Number of trees are the number of decision trees in model. More trees can improve the model's generalization ability, but also increase the training time.

Maximum leaf nodes control the maximum number of leaf nodes, when the number increases, the tree become more complex, the model can thereby catch more patterns in the data but also increase the risk of overfitting. When the number is small, the model tends to underfitting.

Rationale: Random Forest is very effective for image classification because it can handle non-linear relationships between features, is resistant to overfitting, and less sensitive to noise in the data. Each tree in the forest is trained on a slightly different subset of the dataset, which makes the model more robust and better at generalizing to new data. The randomness added during the tree-building process (by randomly selecting data and features) helps to prevent overfitting, which is especially helpful when working with high-dimensional image features.

c. LogisticRegression (Nguyen, 2024)

Principle: Multiclass logistic regression is an extension of logistic regression used to classify data into more than two categories. It uses the softmax function to model the probability distribution over multiple classes.

Logistic regression for binary classification uses the sigmoid function to convert the linear output into probabilities between 0 and 1. For multiclass classification, the softmax function generalizes this idea by converting model outputs into probabilities across multiple classes.

The softmax function ensures that the sum of the probabilities for all classes equals 1, and the class with the highest probability is chosen as the prediction.

Rationale: Logistic regression is computationally efficient compared to more complex models like SVC or deep learning models. This is especially valuable for the Fashion MNIST dataset, which contains many training samples (30,000 images). Logistic regression can quickly train and make predictions. Fashion MNIST images consist of 784

features (28x28 pixels), many of which may not be equally important. Logistic regression supports L1 (which promotes sparsity) and L2 regularization (which prevents large weights), helping reduce overfitting. This is especially useful in high-dimensional data like Fashion MNIST.

d. k-Nearest Neighbors (Nguyen, 2024)

The k-Nearest Neighbors (KNN) algorithm is one of the simplest and most intuitive machine learning algorithms used for both classification and regression tasks. It is a non-parametric, instance-based learning method, which means that it makes predictions based on the instances in the training dataset without explicitly learning a model or parameters.

The k-Nearest Neighbors (KNN) algorithm has several important hyperparameters that can significantly affect its performance.

n_neighbors (K) is the most critical hyperparameter and defines the number of neighbors to consider when making predictions.

Small K, the algorithm becomes more sensitive to noise and outliers but captures more local patterns (higher variance).

Large K, the algorithm becomes more stable by considering more neighbors (lower variance), but it might lead to underfitting if k is too large.

KNN uses a distance metric to measure the proximity between the query point and points in the training data. Common metrics include: Euclidean distance, Manhattan distance.

Weighted KNN, a variation of KNN assigns different weights to the neighbors based on their distance to the query point. Closer neighbors are given more weight, which can improve accuracy in some cases.

C. Evaluation Metric

a. Accuracy:

Accuracy refers to the proportion of correctly predicted samples out of the total samples. The formula is

$$\boldsymbol{accuracy} = \frac{\boldsymbol{Number\ of\ correct\ predictions}}{\boldsymbol{Total\ number\ of\ predictions}}$$

Accuracy is an intuitive and effective metric when the class distribution in the dataset is balanced. However, the data is imbalanced in this assignment.

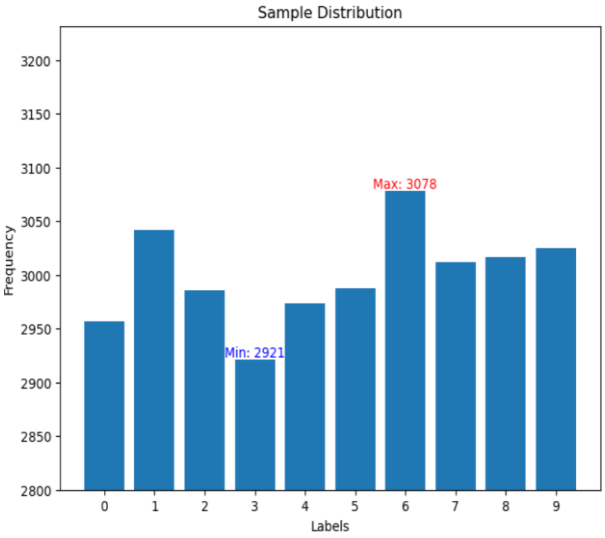

Figure 2 – Distribution of samples and labels

As shown in the figure 2, the samples are not uniformly distributed. The training samples labeled as "6" are the most numerous, with a total of 3,078, while the samples labeled as "3" are the fewest, with a total of 2,921. Therefore, when the data is imbalanced, accuracy may have a poor performance for minority classes.

b. Weighted F1-Score:

F1-Score is the harmonic mean of Precision and Recall, used to balance both metrics. Weighted F1-Score is calculated by weighting the classes based on the number of samples. The formula is:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{Ture\ Positives + False\ Negatives}$$

$$Weighted\ F1 - Score = \sum \left(\frac{Samples\ in\ class\ i}{Total\ samples} \times F1_i \right)$$

Weighted F1-Score is well-suited for the assignment which is imbalanced in samples.

c. Training and inference time:

Training and inference time are also important indicators for measuring model performance. In scenarios that require repeated training or fast inference, the faster the training and inference speed, the more practical the model becomes.

III. Result And Discussion

A. Implementation Process

- Load the dataset, including both data and labels, without splitting the data.
- Apply preprocessing steps: first, reshape the image pixel intensity data from a 30000 x 768 format into a 30000 x 28 x 28 matrix format; second, calculate the mean and standard deviation for the entire dataset; third, subtract the mean and standard deviation from the data; fourth, apply random horizontal flipping, with a 50% probability of flipping the image data; fifth, train the neural network model with a batch size of 256 for a total of 128 iterations.

- Remove the classification head from the trained network model and use it as a feature extractor to extract features from the standardized and randomly flipped data, resulting in a 30000 x 256 feature matrix.
- Through the above preprocessing steps, reduce the original 30000 x 768 matrix to a 3000 x 256 size, significantly reducing the number of features and thereby reducing the training time for subsequent model training.
- Deploy various classification models, including KNN, logistic regression, support vector machine classification, and random forest. Use grid search and 5-fold cross-validation to find the optimal hyperparameter combinations.
- Use the optimal hyperparameter combination for each classification model to train on the full training sample and compare the performance of each model on the test set.
- Use the best model to make predictions on the prediction set and save the prediction results.

B. Hyperparameter Tuning

a. SVC

The grid search setting for SVC is defined as follows:

C (regularization parameter) and Gamma (kernel coefficient) are searched through 5-fold cross-validation grid search. C is chosen between [0.1, 1, 10], and gamma is chosen between [0.001, 0.01, 0.1, 1, 10]. The kernel function used is Radial Basis Function (RBF) kernel.

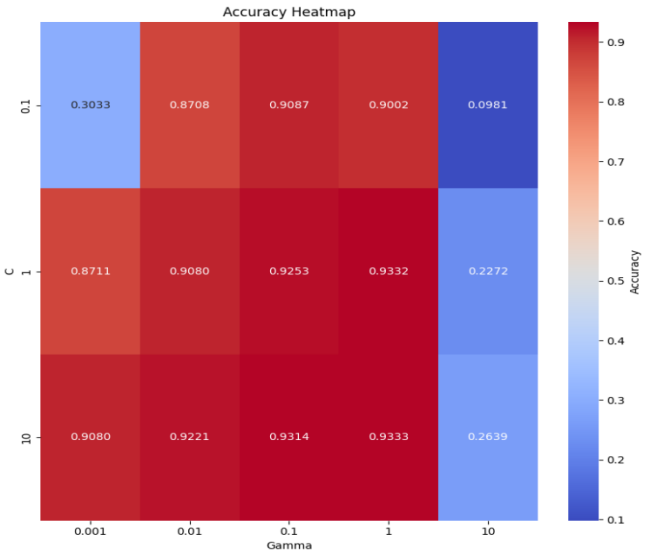

Figure 3 – Accuracy of SVC model

The heatmap above shows the results of hyperparameter tuning for the SVC model, with a grid search performed over the hyperparameter C and Gamma. Each combination of these hyperparameters was evaluated based on accuracy. The color intensity of the heatmap indicates performance.

The combination of lower Gamma value and lower C value result in relatively poor accuracy, especially when C = 0.1 and Gamma = 0.001, the accuracy is low at 30.33%. This indicates that when the penalty is small (controlled by C) and the influence of individual samples (controlled by Gamma) is low, the model tends to underfitting. In addition, when Gamma is too large (10), the model tends to memorize each individual sample and become overfitting.

The best combination is observed with a high C (10) and moderate Gamma (1), meaning that the model performs better when there is less tolerance for misclassification.

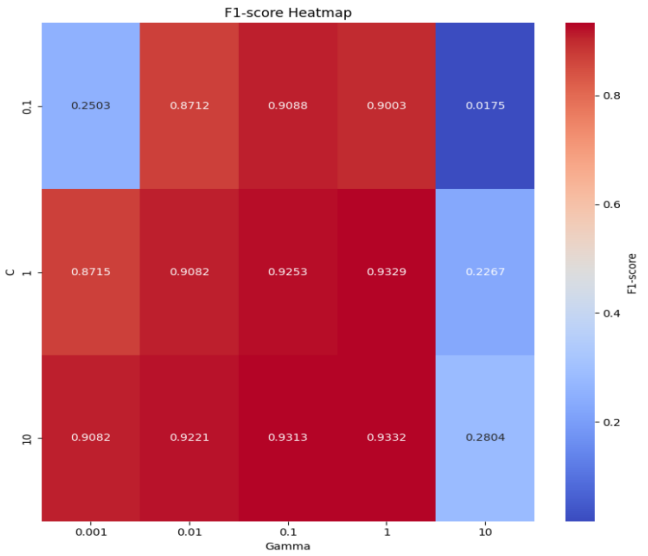

Figure 4 – F1 score of SVC model

The F1-score heatmap follows a similar trend to the accuracy heatmap. The highest F1-scores are observed in the region where C = 1 or 10 and Gamma = 1, reaching 93.29% and 93.32% respectively.

The best performance of both accuracy and F1-score is achieved when C is moderate to high (around 1 or 10) and Gamma is around 1. These settings strike a good balance between overfitting and underfitting, leading to better generalization.

hyperparameter	training_time (seconds)	inference_time (seconds)
c = 0.1, gamma = 0.001	46.62	16.65
c = 0.1, gamma = 0.01	20.17	12.17
c = 0.1, gamma = 0.1	7.55	6.54
c = 0.1, gamma = 1	13.56	7.61
c = 0.1, gamma = 10	47.44	16.41
c = 1, gamma = 0.001	21.42	13.23
c = 1, gamma = 0.01	8.06	7.20
c = 1, gamma = 0.1	3.82	3.72
c = 1, gamma = 1	10.90	5.63
c = 1, gamma = 10	57.41	17.20
c = 10, gamma = 0.001	6.63	6.05
c = 10, gamma = 0.01	3.21	3.41
c = 10, gamma = 0.1	2.71	2.70
c = 10, gamma = 1	10.41	5.76
c = 10, gamma = 10	48.13	16.49

Table 1 – Training and inference time of SVC

The table shows the training and inference times for different hyperparameter combinations of SVC. It can be observed that when Gamma is either relatively high or low, both the training time and inference time tend to increase. This is likely due to the model's sensitivity to these extreme Gamma values, where very small or very large values can lead to overfitting or underfitting, requiring more time to compute optimal decision boundaries.

In the current experimental setup, as C increases, both training time and inference time decrease.

In summary, considering the accuracy, F1-score, and training/inference time, and given that this task places more

emphasis on accuracy, we do not choose the combination with the optimal time but instead select C = 10 and Gamma = 1 as the best hyperparameter combination, since the time is within an acceptable range.

b. Random Forest

The grid search setting for Random Forest is defined as follows:

Number of tress (n_estimator) and maximum tree nodes (nodes) are searched through 5-fold cross-validation grid search. n_estimator is chosen between [100, 300, 500], and nodes is chosen between [10, 16, 32].

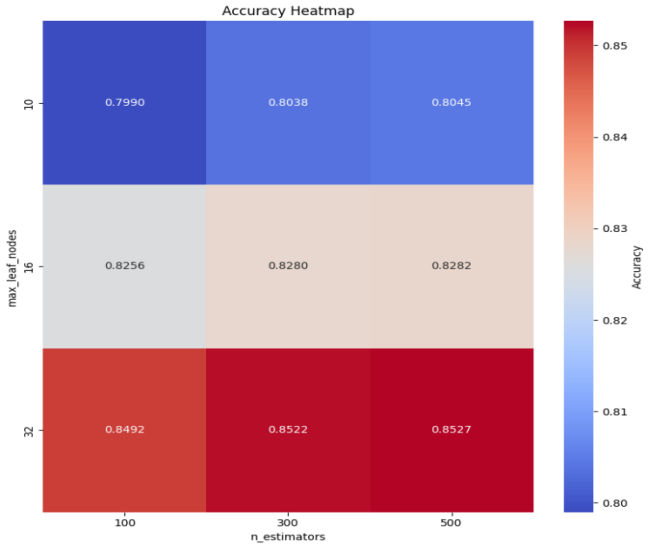

Figure 5 –Accuracy of random forest model

The heatmap above shows the results of hyperparameter tuning for the Random Forest model, with a grid search performed over the hyperparameter number of trees (n_estimator) and maximum leaf nodes (nodes). Each combination of these hyperparameters was evaluated based on accuracy. The color intensity of the heatmap indicates performance.

It can be seen from the figure that as the number of trees increases from 100 to 500, accuracy gradually increases. When the maximum number of nodes is 32 and number of trees is 500, the model achieves the highest accuracy at 85.27%. It indicates that more trees provide better average of predictions, leading to better generalization. Besides, low number of nodes in a tree leads to underfitting in the dataset.

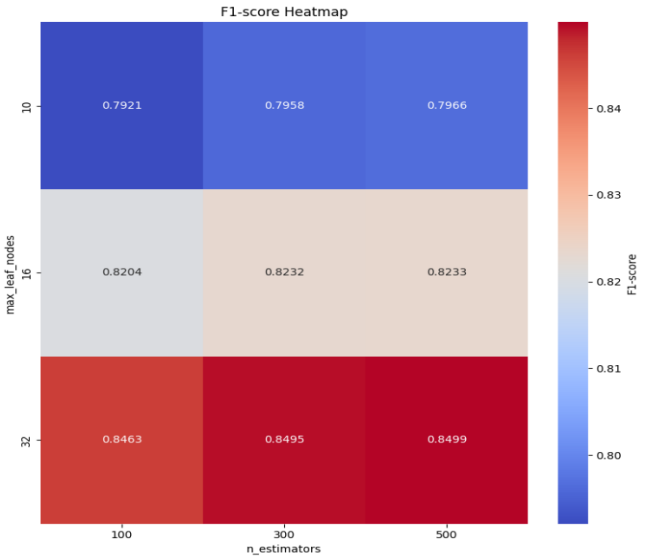

Figure 6 – F1 score of random forest model

The F1-score follows a similar trend as the accuracy. The highest F1-score is 0.8499 at n_estimators = 500 and nodes = 32. Like accuracy, F1-score improves with more trees and more nodes. This indicates that the balance between precision and recall improves as the model becomes more complex (with more estimators and deeper trees).

The optimal setting for the Random Forest in this hyperparameter grid is n_estimators = 500 and nodes = 32, which results in the highest accuracy and F1-score.

hyperparameters	training_time (seconds)	inference_time (seconds)
n_estimators = 100, nodes = 10	11.76	0.03
n_estimators = 100, nodes = 16	13.58	0.03
n_estimators = 100, nodes = 32	15.50	0.03
n_estimators = 300, nodes = 10	35.04	0.07
n_estimators = 300, nodes = 16	39.50	0.08
n_estimators = 300, nodes = 32	46.69	0.10
n_estimators = 500, nodes = 10	57.91	0.11
n_estimators = 500,nodes = 16	66.09	0.13
n_estimators = 500, nodes = 32	74.36	0.16

Table 2 – Training and inference time of Random Forest

The table shows the training and inference times for different hyperparameter combinations of Random Forest. Both the training time and inference time increase when the model becomes more complex (including more trees and more complex for a single tree).

However, despite the computation time of the complex model being several times that of the simpler model, considering the significant performance improvements brought by the complex model (in terms of accuracy and F1-score), we choose node = 32, n_estimators = 300 as the optimal hyperparameter combination for the Random Forest.

c. k-Nearest Neighbors

The grid search setting for K-Nearest Neighbors is defined as follows:

number of neighbors (n_neighbors): 5, 7, 9, 11, distance metric (p): Manhattan distance (p=1) and Euclidean distance (p=2).

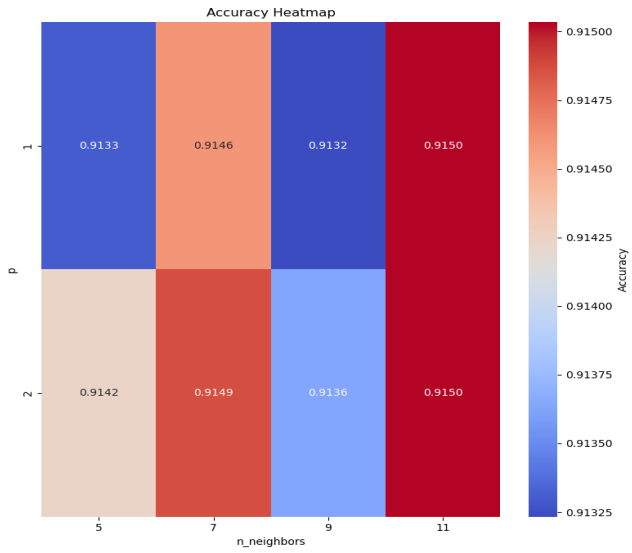

Figure 7 – Accuracy of KNN

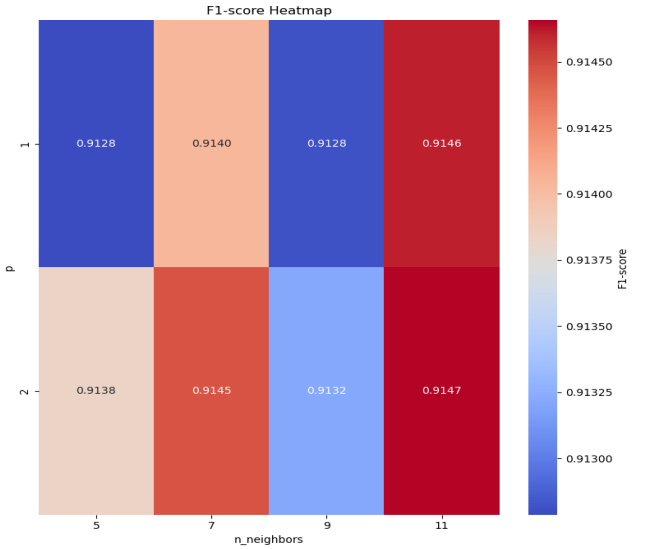

Figure 8 – F1 score of KNN

The results show that the distance metric has a significant influence on both performance and efficiency. When using Manhattan distance (p=1), inference times were notably longer compared to Euclidean distance (p=2). For instance, with n_neighbors=5, the inference time for p=1 was 1.9121 seconds, whereas for p=2, it was just 0.2898 seconds. This pattern held true across different neighbor counts. In addition to faster inference times, models using Euclidean distance also achieved slightly better accuracy. For example, when n_neighbors =7, the accuracy for `p=1` was 0.9132, while for p=2 it was 0.9150.

Hyperparameters	Training Time (seconds)	Inference Time (seconds)
n_neighbors = 5, p = 1	0.0034	1.9121
n_neighbors = 5, p = 2	0.0034	0.2898
n_neighbors = 7, p = 1	0.0035	1.8907
n_neighbors = 7, p = 2	0.0033	0.2881
n_neighbors = 9, p = 1	0.0033	1.8612
n_neighbors = 9, p = 2	0.0031	0.2977
n_neighbors = 11, p = 1	0.0036	1.8914
n_neighbors = 11, p = 2	0.0036	0.3069

Table 3 – Training and inference time of KNN

Regarding the number of neighbors, the accuracy and F1 score improved slightly as n_neighbors increased from 5 to 7. However, further increases to 9 or 11 yielded diminishing returns. For example, the accuracy with n_neighbors=9 was 0.9149, very close to the result for n_neighbors=7. Inference time was slightly affected by changes in the number of neighbors, with shorter times observed for higher neighbor counts, especially when using Manhattan distance.

Training time remained consistent across all hyperparameter combinations, ranging between 0.0031 and 0.0036 seconds, as the KNN algorithm primarily focuses on storing data during the training phase.

Based on this analysis, the combination of n_neighbors=7 and p=2 (Euclidean distance) is recommended as it strikes an optimal balance between accuracy (0.9150), F1 score (0.9146), and inference speed (0.2881 seconds). This setup provides strong performance with much faster inference times, making it a good fit for situations where quick predictions are important. In the future, it would be interesting to study how KNN performs on larger or more complex datasets. Additionally, exploring optimization techniques could help improve the model's efficiency even further.

d. LogisticRegression

The grid search setting for LogisticRegression is defined as follows:

C is the inverse of the regularization strength, controlling how much the model is penalized for misclassified samples. It balances between fitting the data and regularization. A larger C value means weaker regularization, while a smaller C value means stronger regularization.

The penalty parameter specifies the type of regularization, with two commonly used types in logistic regression: L1(also known as Lasso regularization) and L2 regularization (also known as Ridge regularization).

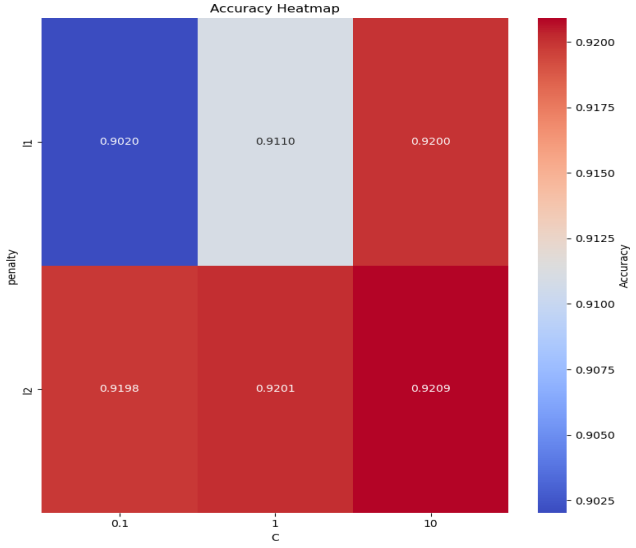

Figure 9 – Accuracy of Logistic Regression

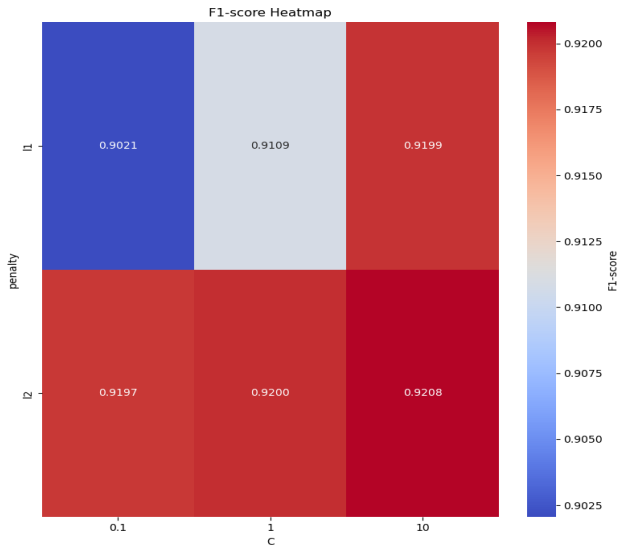

Figure 10 – F1 score of Logistic Regression

Hyperparameters	Training Time (seconds)	Inference Time (seconds)
C = 0.1, penalty = l1	8.9633	0.0023
C = 0.1, penalty = l2	1.806	0.0014
C = 1, penalty = l1	38.6745	0.0016
C = 1, penalty = l2	5.2656	0.0013
C = 10, penalty = l1	119.219	0.0018
C = 10, penalty = l2	225.9885	0.0018

Table 4 – Training and inference time of LogistiRegression

The regularization type, whether L1 (penalty=l1) or L2 (penalty=l2), significantly influences the model's accuracy and computational efficiency. L2 regularization consistently outperforms L1 regularization in terms of both accuracy and training time. For instance, when C=0.1, L2 regularization achieves an accuracy of 0.9110 and an F1 score of 0.9109, whereas L1 regularization reaches only 0.9020 accuracy with a similar F1 score of 0.9021. The training time for L2 regularization (1.8060 seconds) is also much shorter compared to L1 (8.9633 seconds). L1 regularization tends to require significantly more training time as the value of C increases. For example, with C=1, the training time for L1 regularization increases to 38.6745 seconds, while L2 regularization remains faster at 5.2656 seconds for the same C value. Despite L1 regularization taking longer to train, the difference in accuracy between L1 and L2 is marginal when C=1 or higher. At C=10, L1 regularization achieves 0.9201 accuracy and 0.9200 F1 score, which is very close to L2 regularization's 0.9209 accuracy and 0.9208 F1 score.

The regularization strength (C), which controls the trade-off between fitting the data and regularization, has a notable impact on the model's performance and training time, Lower values of C (C=0.1) lead to faster training times but lower accuracy. For example, with C=0.1, the accuracy for L2 regularization is 0.9110, and the training time is only 1.8060 seconds. In contrast, as C increases, the model fits the data more closely, resulting in higher accuracy but longer training times.

Higher values of C (C=10) improve accuracy slightly, but this comes with a significant increase in training time. For instance, when C=10, L2 regularization achieves the highest accuracy of 0.9209, but the training time escalates to 225.9885 seconds—over 100 times longer than with C=0.1.

While increasing C slightly boosts accuracy, the performance gain diminishes at higher values. Moving from C=1 to C=10, the accuracy improves marginally (from 0.9198 to 0.9209 for L2 regularization), suggesting diminishing returns as the regularization becomes weaker.

Training time is heavily influenced by both the regularization type and the strength of regularization (C). L1 regularization generally requires more time to converge, especially for higher values of C. This is because L1 regularization promotes sparsity in the coefficients, which increases the computational complexity of optimization, especially when using solvers like saga. L2 regularization is much faster across all values of C. Even at high values of C (such as 10), L2 regularization remains relatively efficient compared to L1, though the training time increases as regularization weakens. For example, training time with C=10 is 119.2190 seconds for L1 and 225.9885 seconds for L2, with L2 taking significantly longer at this extreme.

Inference time across all configurations is very short and largely unaffected by the choice of hyperparameters.

This analysis shows that L2 regularization generally offers better performance with faster training times compared to L1 regularization across all values of C. While increasing the regularization strength (C) slightly improves accuracy, the gains diminish as C becomes larger, and the training time increases substantially. L2 regularization with C=1 provides a strong balance between accuracy (0.9198) and training time (5.2656 seconds), making it a suitable choice for most applications where training efficiency is important.

e. Model comparison

In conclusion, the best hyperparameters combination for each machine learning model is set as follows:

- SVC (C = 10, Gamma = 1)
- Random Forest (nodes = 32, n_estimators = 300)
- KNN (n = 7, distance = Euclidean distance)
- Logistic Regression (L2 Reg, C = 1)

Models	Accuracy	F1-Score	Training Time(s)	Inference Time (s)
SVC	0.9333	0.9332	10.41	5.76
Random Forest	0.8522	0.8495	46.69	0.1
KNN	0.9149	0.9145	0.0033	0.2881
Logistic Regression	0.9201	0.92	5.2656	0.0013

Table 5 – Model comparison

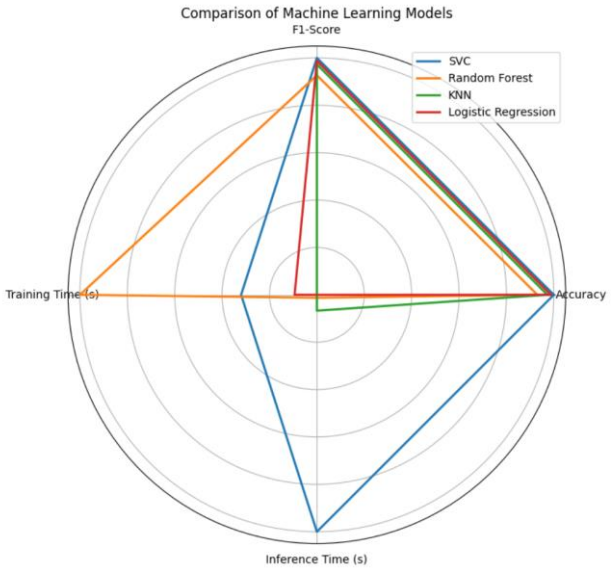

Figure 11 – Model comparison

Table 5 and Figure 11 compare the final test data evaluation of each model based on Accuracy, F1-Score, Training Time (s), and Inference Time (s). Each model shows different pros and cons.

The SVC model, while achieving the highest accuracy and F1-score, requires a significantly longer training time than other models.

In contrast, the Random Forest requires even longer training than SVC but significantly lower inference time. Which indicates that Random Forest is very practical in scenarios where inference time is prioritized over training time.

KNN has a very fast training speed, making it extremely suitable for scenarios that require repeated training. Additionally, as an unsupervised learning method, it is well-suited for unlabeled scenarios, offering greater flexibility.

Finally, Logistic Regression performs well in all metrics. In memory-constrained scenarios, it stands out as a compact model. While its accuracy and F1-score are not particularly outstanding, it remains an acceptable choice.

In summary, considering that accuracy is the most important metric in this experiment, our optimal model is the SVC model, as it significantly outperforms the other two models in both accuracy and F1-score.

IV. Conclusion

In this report, we explored the performance of four machine learning models, k-Nearest Neighbors (KNN), Logistic Regression, Support Vector Classifier (SVC), and Random Forest—on the Fashion-MNIST dataset. The primary objective was to determine the most efficient model for classifying fashion images based on several evaluation metrics, including accuracy, F1-score, training time, and inference time.

Our findings suggest that the SVC model, with a high regularization parameter (C=10) and a moderate kernel coefficient (Gamma=1), performs best in terms of accuracy and F1-score, achieving over 93% on both metrics. Random Forest also exhibited strong performance with an accuracy of 85.66% when optimized with 300 trees and 32 maximum leaf nodes. Logistic Regression provided a computationally efficient solution, making it suitable for large datasets due to its relatively fast training and inference times. Finally, while KNN was the simplest model, it showed competitive performance when tuned correctly, especially when using the Euclidean distance metric.

However, this study also encountered several limitations. First, the dataset was imbalanced, which impacted the accuracy and F1-score, particularly for less frequent classes. Despite using the weighted F1-score to mitigate this, future work could explore more sophisticated techniques, such as data resampling or class-specific weighting. Additionally, the computational cost of training complex models like SVC and Random Forest was relatively high, which may be a limitation in resource-constrained environments.

Looking ahead, potential areas for future work include experimenting with deep learning models like convolutional neural networks (CNNs) as a classifier instead of feature extractor to further enhance classification accuracy. Incorporating transfer learning techniques with pre-trained models could also improve performance, reduce training time, and provide more robust feature extraction. Finally, applying additional data augmentation techniques or exploring more advanced hyperparameter tuning methods such as Bayesian optimization may lead to further improvements in both model efficiency and accuracy.

References

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Leaming, 20, 273-297. doi:https://doi.org/10.1007/BF00994018.

HeKaiming, ZhangXiangyu, RenShaoqing, & SunJian. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (770-778). Las Vegas, NV, USA. doi:10.1109/CVPR.2016.90.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. doi:10.1109/5.726791.

Leo, B. (2001). Random Forests. Machine Learning, 45, 5-32. doi:https://doi.org/10.1023/A:1010933404324.

Nguyen H. Tran (2024). Lecture 6: Support vector machines [Powerpoint Slide]. COMP4318/5318 Machine Learning and Data Mining. https://canvas.sydney.edu.au/courses/59516

Nguyen H. Tran (2024). Lecture 5: Ensembles [Powerpoint Slide]. COMP4318/5318 Machine Learning and Data Mining. https://canvas.sydney.edu.au/courses/59516

Nguyen H. Tran (2024). Lecture 3: Linear regression. Logistic regression [Powerpoint Slide]. COMP4318/5318 Machine Learning and Data Mining. https://canvas.sydney.edu.au/courses/59516

Nguyen H. Tran (2024). Lecture 2: k-Nearest Neighbor [Powerpoint Slide]. COMP4318/5318 Machine Learning and Data Mining. https://canvas.sydney.edu.au/courses/59516

APPENDIX

How to run the code:

1. Copy the data folder to the same directory with .ipynb file, the folder should be names as “data”.

2. Install pytorch follow instructions on Start Locally | PyTorch or run on google colab.

3. Install other missing packages using “!pip install xxx” command.

4. Run all the cells by order.

5. The output will be named “test_output.csv” and saved in the current directory.

Important:

Due to the randomness in the model training process and the cross validation split, the results may not be identical (primarily due to the training of CNN), but the final accuracy typically varies within 1.5%.

The hardware is i3-12100 and RTX2070.