COMP5318 Assignment 1

Jiaming(Jazlyn) Lin 470345744

September 2021

1 Introduction

In this study, students were required to perform image classification tasks to a given clothing dataset without being restricted to using existing packages. The aim of the study is to explore different types of machine learning classifiers, select several best-performing ones and then have a meaningful discussion of their performances, differences, limitations, and possible improvements. The study is important as it provides a good opportunity for students to put their knowledge learned so far into practice with a dataset on a much larger scale and learn how the Machine Learning projects in general works from the beginning to the end. Moreover, students were also encouraged to think critically of what they have learned by evaluating, reasoning, and penalizing the classification result. This can be great preparation for the future Machine Learning challenges they will encounter.

2 Method

2.1 Data Preprocessing

2.1.1 Normalization through Min-Max Scaling

Feature scaling is a common practice to ensure the good performance of various machine learning classifiers implemented in scikit-learn. [9] It matters to many algorithms and can provide many benefits. For example, distance-based algorithms such as K-nearest neighbors (KNN) with a Euclidean distance measure can be very sensitive to feature magnitudes and easily affected by dominant features without scaling. It also plays a key role in the performance of another preprocessing method adopted in this study - Principal Component Analysis (PCA), which can be skewed towards dominant features. Moreover, it also encourages faster convergence for Neural Networks algorithms. [12] [6]

One way to perform normalization is through MinMax Scaling ($MinMaxScaler$ in scikit-learn) and scale features to lie between 0 and 1. From the brief inspection, the features of both training and testing datasets already lie between such ranges, so we will skip this section in the implementation.

2.1.2 Dimension Reduction through PCA

PCA is a popular dimension reduction technique aiming to transform original data from a high-dimensional space into a low-dimensional space. The transformed data in such low dimensional space are represented by a limited number of principal components while it still retains some meaningful properties of the original data. [5]

Given the dataset has a huge feature dimension (i.e. 784), it is crucial to perform dimension reduction such as PCA so that the model complexity can be reduced and overfitting can be avoided as much as possible. By judging the below Variance Explained figure, the $n_{components} = 60$ was selected as it looks like an 'elbow' point [2] that balances the trade-off between feature dimension and data variance/retained information.

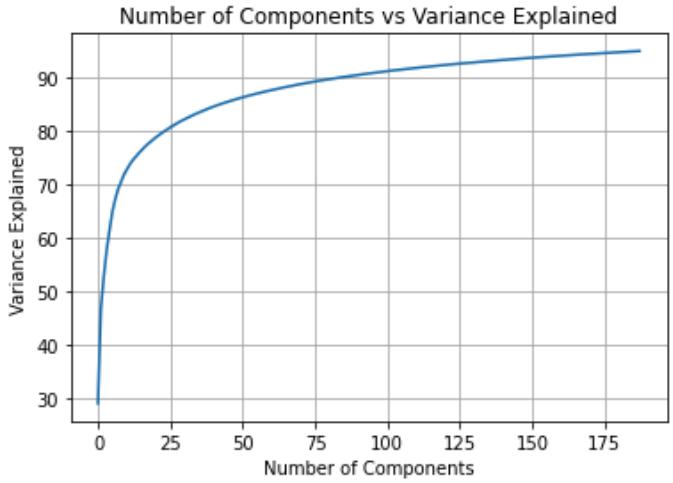


Figure 1: PCA Variance Explained

2.2 Classification Algorithms

Three classification algorithms were selected and applied to the preprocessed data.

Algorithm 1: K-nearest neighbors (KNN)

K-nearest neighbors (KNN) algorithm is a non-parametric classification method that can be applied for both classification and regression tasks. Essentially, this algorithm assumes that similar examples are close to each other and the similarity between examples can be defined by example distance. [4] In this classification study, KNN classifies a new image by predicting the label from its K closest neighbors' labels using majority voting. Note that the neighbor closeness can be measured by different distance types and the voting contribution of each neighbor can also be weighted differently. This naturally leads to the three key hyperparameters to tune - the number of neighbors (K), distance

measures (p), and neighbor weighting type. By utilizing $GridSearchCV$ method in scikit-learn, the best hyperparameter combination with the highest cross-validation accuracy can be selected for the best KNN model.

Algorithm 2: Support Vector Machines (SVM)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers detection. The goal of SVM is to construct a hyperplane or set of hyperplanes in a high-dimensional space such that the distance between the closest examples of any class, namely, margin, is maximized. Such hyperplane(s) will then be used to perform image classification[7]. SVM can be either linear or non-linear, depending on whether the data are linear-separable. In practice, the class decision boundary in data is constantly to be non-linear and we have to adopt non-linear SVM. Thanks to the kernel tricks, non-linear SVM can transform the data to be linearly separable in a higher-dimensional space while avoiding the huge computational cost in such space. As shown in Figure **??**, there are different kernel types and we can tune them to see which one fits the best. Besides the kernel type, there are also several other key hyperparameters such as the regularization hyperparameter C and the kernel coefficient $gamma$[8]. In this study, we will tune those three key hyperparameters using $GridSearchCV$ to investigate the best combination.

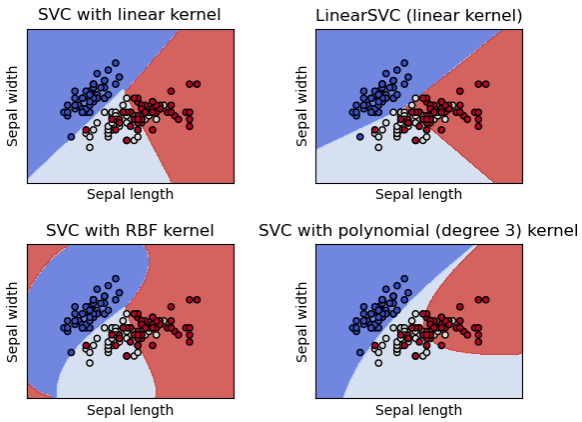


Figure 2: Different Types of SVM Kernels

Algorithm 3: Bagging with Support Vector Machines (SVM)

Bagging is a type of ensemble algorithms. It builds a group of classifiers over random samples from the original dataset and then aggregates the individual classifier's prediction to form the final prediction. For bagging with SVM's case here, a group of SVM classifiers was trained over random samples and each SVM's individual classification outcome was aggregated through majority voting to select the final classification outcome for a new image that needs to be classified. The key hyperparameter to

tune here is the number of base estimators.

3 Experimentation Result

3.1 Experiment Setup

1. Hardware:
    - MacBook Pro with 8GB RAM
2. Software:
    - macOS Big Sur
    - Jupyter Notebook with Anaconda
    - Python 3.7

3.2 Classifier Performance

Five classifiers, namely, KNN, SVM, Bagging with SVM, AdaBoost with Decision Trees, Bagging with Decision Trees were explored and experimented with 10-fold cross-validation using *GridSearchCV* from scikit-learn. Based on the testing performance, the top three classifiers (i.e. KNN, SVM, and Bagging with SVM) with the best hyperparameters tuned were selected.

In the remaining part of this section, we first summarise the macro performance of selected classifiers and compare such performance with their available benchmarks. Then we go through classifiers' micro performance as well as time performance and identify patterns. In the end, we dive deep into the numbers and patterns and provide possible justifications with evidence.

Performance Comparision Overall, *SVM* and *Bagging with SVM* have the best and most consistent testing performance, with 88% for testing accuracy, precision, recall, and F1 score and around 95% training accuracy. Also, it is worth mentioning that the accuracy achieved for both classifiers is very close to the benchmarks(with the same hyperparameters) online[3][14]. Though it is somewhat unclear if those benchmarks preprocessed the data in the same way as we do, this still confirms both classifiers are reaching most of their potentials and perform well enough.

In comparison, *KNN* has a little worse testing performance, where all testing performance metrics are 3% to 4% lower than those of the previous two classifiers. *KNN* also has the worst overfitting issue, with a 15% accuracy drop from training to testing, which is more than doubled the accuracy decrease of the other two classifiers (7%).

Classifier	Best Hyperparameter	Training	Testing				
		Accuracy (ours)	*Accuracy (ours)*	*Accuracy (benmark's)*	*Precision*	*Recall*	*F1 Score*
KNN	- n_neighbors = 9 - p = 1 - weights = 'distance'	100.0%	84.2%	85.1%	85%	84%	85%
SVM	- kernel='rbf' (default) - gamma = 'scale' (default) - C = 10	94.7%	88.1%	89.6%	88%	88%	88%
Bagging with SVM	- base = svm.SVC(kernel='rbf', C = 10, gamma = 'scale') - n_estimators = 20	95.0%	88.0%	N/A	88%	88%	88%

Figure 3: Comparison of Classifier Performance Metrics

As for the time-wise performance, there is usually a negative correlation between classifier training time and inference time. Out of all three classifiers, both *Bagging with SVM* takes much longer time to train but less time to infer/classify a new image, even though the training time of SVM is only $\frac{1}{8}$ of *Bagging with SVM*'s. Reversely, when it comes to *KNN*, it takes almost no time to train but also a much longer time to infer/classify a new image, where inference time is similar to the other two's.

Classifier	Preprocessing Time	Training Time	Inference Time
KNN	3.73s	0.01s	4.19s
SVM		16.51s	3.92s
Bagging with SVM		133.94s	3.95s

Figure 4: Comparison of Classifier Time Metrics

The above are all macro performances. If we split the metrics into classes and look at the micro metrics, as illustrated in confusion matrices and the classification reports, all three classifiers demonstrate similar patterns, overall ten classes. Note that here due to the page limit, the images were displayed in a compact way to demonstrate high-level patterns. However, a larger version of those images can always be inspected from Appendix if needed.

1. Micro performance seems to be class-dependent. While most classes have at least 80% micro performance scores, Shirt (class 6), T-shirt/Top (class 0), Coat (class 4) and Pullover (class 2) classes stand out with much lower scores.

2. Shirt (class 6) consistently has the lowest micro performance scores among all three classifiers. Precision, recall, and F1 Score are only between 60% and 70%, while those numbers for most

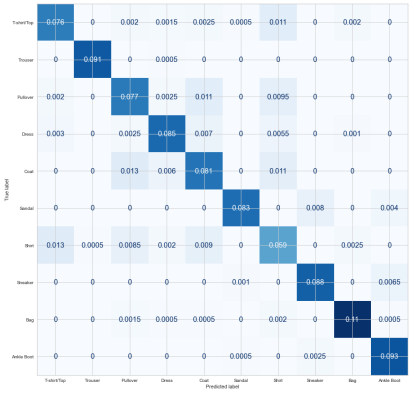


Figure 5: KNN

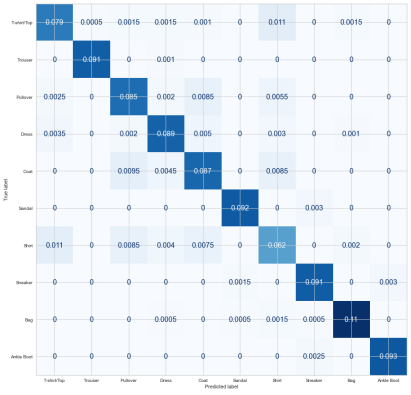


Figure 6: SVM

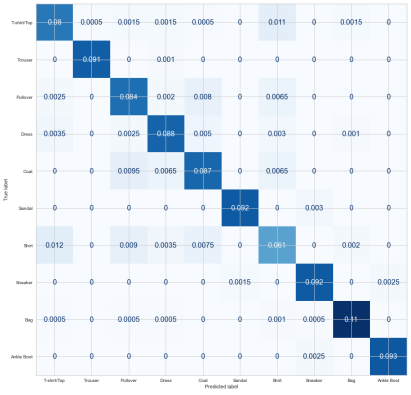


Figure 7: Bagging SVM

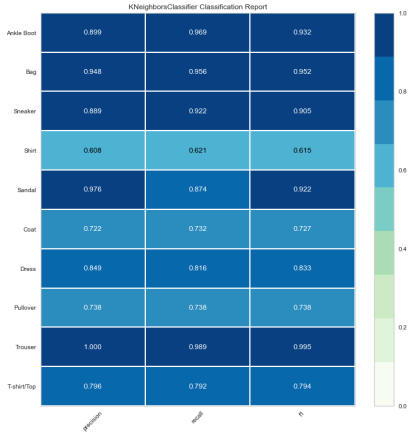


Figure 8: KNN Report

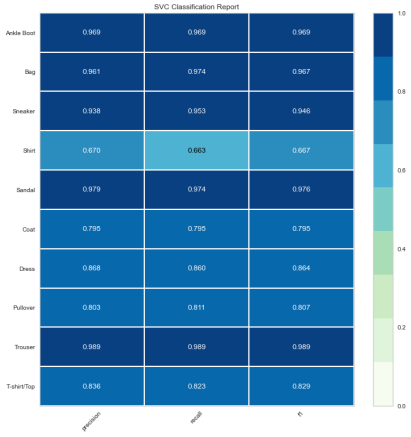


Figure 9: SVM Report

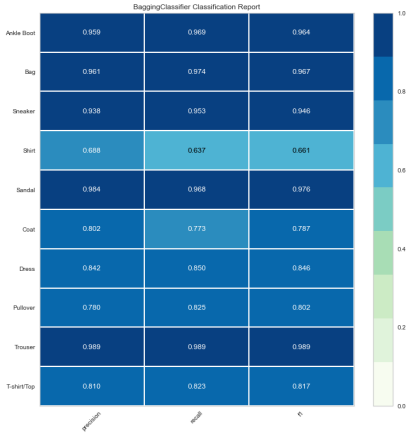


Figure 10: Bagging SVM Report

other classes are at least 80%. Those low scores can also be confirmed visually through the normalized confusion matrix, where we can see many images in Shirt (class 6) were consistently misclassified as T-shirt/Top (class 0), Coat (class 4), and Pullover (class 2).

3. T-shirt/Top (class 0), Coat (class 4) and Pullover (class 2) have less bad scores than Shirt (class 6) but the micro performance is still worse than the remaining classes, varying between 70% and 80% while other classes' over 80%. Similar to Shirt (class 6), as shown in the confusion matrix, they were consistently misclassified as each other including the Shirt (class 6).

.

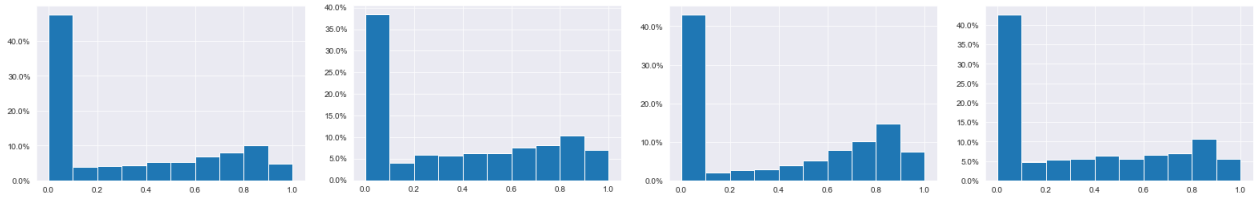
Performance Justification As for training time, it makes sense why KNN has almost no training time compared to the other two classifiers. During the training phase, unlike most classifiers, KNN does no model building but only calculating distances between all examples and stores the results. However, as a typical lazy learning approach, it is indeed slower to evaluate [1] as it delays the computational cost to the inference/evaluation stage. The distance of new examples and existing examples

needs to be calculated on the fly to determine K nearest neighbors than the label prediction.[13]

In contrast, SVM and Bagging with SVM take much longer time to train. The first reason is that SVM essentially solves a quadratic programming problem to separate support vectors from the other part of training data. [11] The SVM used in this study, implemented by *libsvm* usually has a time complexity between $\mathcal{O}(mn^2)$ and $\mathcal{O}(mn^3)$, where m is the number of feature dimensions and n is the number of training examples.[8] This does not scale well with a huge data size such as we have in this study (i.e. n = 30K and m = 60) and is justifiable to take such a long time to train.

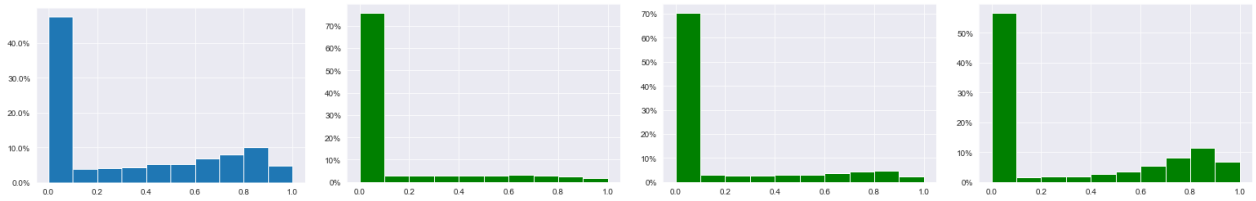
Bagging with SVM takes approximately 10x the training time of SVM. This is because, in this study, the classifier essentially needs to train 20 individual base SVM estimators before aggregating the individual predictions. Since the parameter *max_sample* wasn't set specifically, the default value 1.0 is taken, which means all $30K \times 1.0$ samples were drawn every time to train individual base estimators[10]. This massively boosts the training time and explains why it takes much longer to train.

With regard to micro performance split by classes, as shown in Figure 11, classes with low-performance scores share a more similar normalized pixel distribution compared to those with high-performance scores in Figure 12. Considering that the pixel is normalized and already between [0,1](which makes each distribution less distinguishable), such similarity in the distribution shape really confirms the intrinsic similarity between those low-performance classes. As illustrated in Figure **??**, the four misclassified images from four classes with low-performance classes are indeed very similar and are likely to be confused with each other even for human classification. Such intrinsic similarity possibly explains why classifiers also tend to confuse those classes and are more likely to misclassify them.



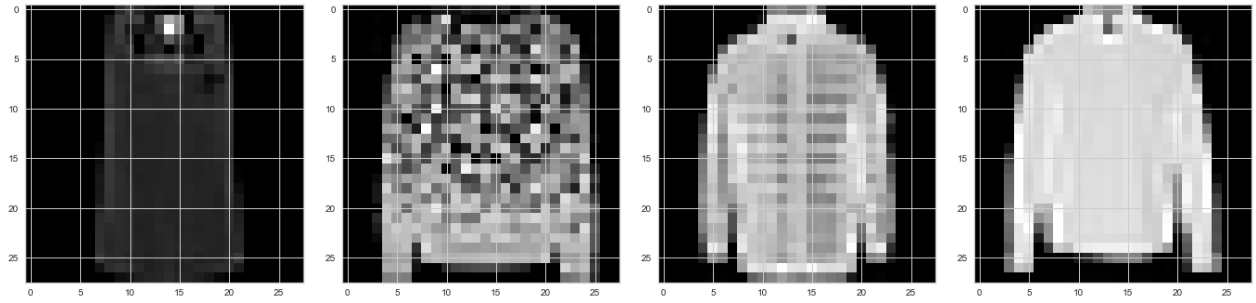
(a) (b) (c) (d)

Figure 11: The normalised pixel distribution of four confusing classes. From (a) to (d): T-shirt/Top(0), Pullover(2), Coat(4) and Shirt(6)



(a) (b) (c) (d)

Figure 12: The normalized pixel distribution of four less confusing classes. From (a) to (d): Trouser(1), Sandal(5), Sneaker(7) and Ankle Boot(9)



(a) (b) (c) (d)

Figure 13: Misclassified images from four confusing classes. From (a) to (d): T-shirt/Top(0) to Shirt(6), Pullover(2) to Shirt(6), Coat(4) to Shirt(6), Shirt(6) to Coat(4)

4 Discussion and Future Work

In this study, students were asked to build at least three multi-class image classifiers to classify the given greyscale datasets that are believed to be sampled from Fashion-MNIST datasets. My group (only myself) explored and experimented with five possible classifiers and selected three classifiers, namely K-Nearest Neighbor (KNN) classifier, Support Vector Machines (SVM) classifier as well as Bagging with SVM classifier in the end. All three classifiers achieved great performance overall as their accuracy is only 1.5% away from the comparable benchmarks available online. SVM is considered as our best classifier given its high testing performance metrics (i.e. highest testing accuracy, precision, recall, and F1 score) as well as relatively short training time (compared to Bagging SVM) and inference time (compared to KNN). However, when split into classes, the classifiers' micro performance all suffers to some extent. All classifiers seem to have trouble classifying certain confusing classes (e.g. Shirt (class 6)) correctly and based on the pixel distribution illustrated we believe it is partially due to the intrinsic similarity between images from certain classes.

Although we aim to achieve 90% accuracy, based on the benchmark statistics, it seems to be unlikely to realize such a goal utilizing the classifiers on the provided list and we only realize that it is possible to achieve such a goal utilizing Convolutional Neural Network (CNN) in the latter part of this study from reading benchmark online[3]. Hence for future work, we definitely aim to experiment with more advanced classifiers such as those submitted CNN classifier benchmarks for FashionMNIST[3] and obtain a big improvement in related performance metrics. Besides that, given certain confusing classes appear to be the accuracy bottleneck of all classifiers, we also hope to investigate and explore more about potential effective classifiers to improve the class-wise classification performance as well as the overall performance.

References

[1] Wikipedia Authors. Lazy learning. Available at https://en.wikipedia.org/wiki/Lazy_learning.

[2] Matt Brems. A one-stop shop for pca. Available at https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c.

[3] FashionMNIST developers. Fashionmnist benchmark dashboard. Available at http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/#.

[4] Onel Harrison. Ml basics with the knn algorithm. Available at https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[5] Lorraine Li. Pca for dimensionality reduction. Available at https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-115a3d157bad.

[6] Baijayanta Roy. All about feature scaling. Available at https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35.

[7] scikit-learn developers. 1.4. support vector machines. Available at https://scikit-learn.org/stable/modules/svm.html.

[8] scikit-learn developers. 1.4. support vector machines. Available at https://scikit-learn.org/stable/modules/svm.html#svm-classification.

[9] scikit-learn developers. 6.3. preprocessing data. Available at https://scikit-learn.org/stable/modules/preprocessing.html.

[10] scikit-learn developers. sklearn.ensemble.baggingclassifier. Available at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html.

[11] scikit-learn developers. sklearn.svm.svc. Available at https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[12] Nguyen H. Tran. Comp 5318 lecture notes. The University of Sydney.

[13] Stackoverflow user Jondiedoop. Why does test takes longer than training? Available at https://stackoverflow.com/questions/53133458/why-does-test-takes-longer-than-training.

[14] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

5 Appendix

5.1 Hardware and Software Setup

1. Hardware
    - MacBook Pro with 8GB RAM
2. Software
    - macOS Big Sur
    - Anaconda
    - Python 3.7

5.2 Instruction to run code

- Run section 1 to load the data and relevant libraries
- Run section 2 to preprocess the training and testing data
- Uncomment the code in section 3 to tune for the best hyperparameters, check cv accuracy, and testing accuracy.
- Run section 4 to implement classifiers with their best hyperparameters and obtain some time metrics.
- Run section 5 to obtain classifiers performance metrics such as confusion matrix, classification report, inference time, and other visualisations and compare their performance.
- Run section 6 to save the best classifier (SVM)'s output into the h5 file.

5.3 Figures

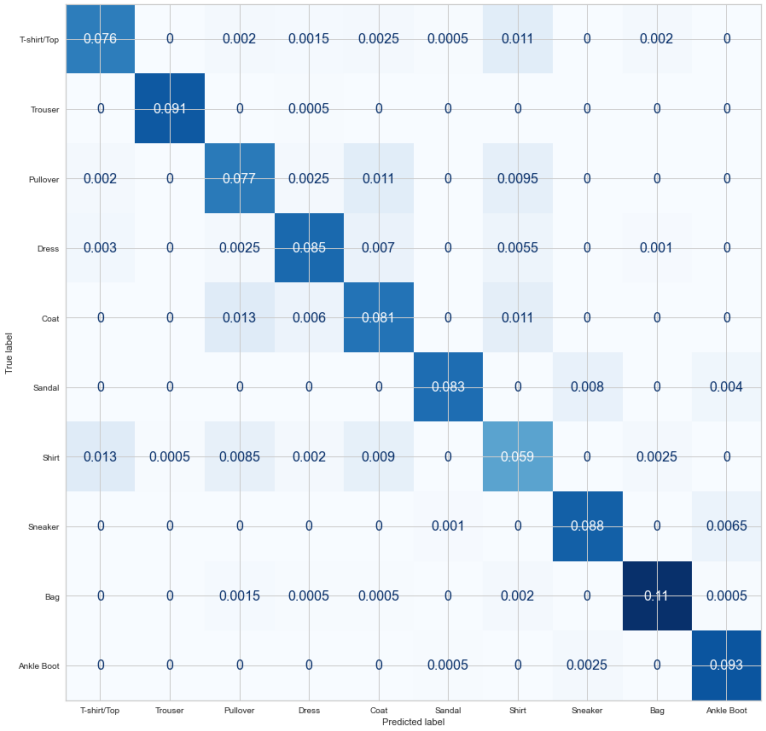


Figure 14: KNN confusion matrix

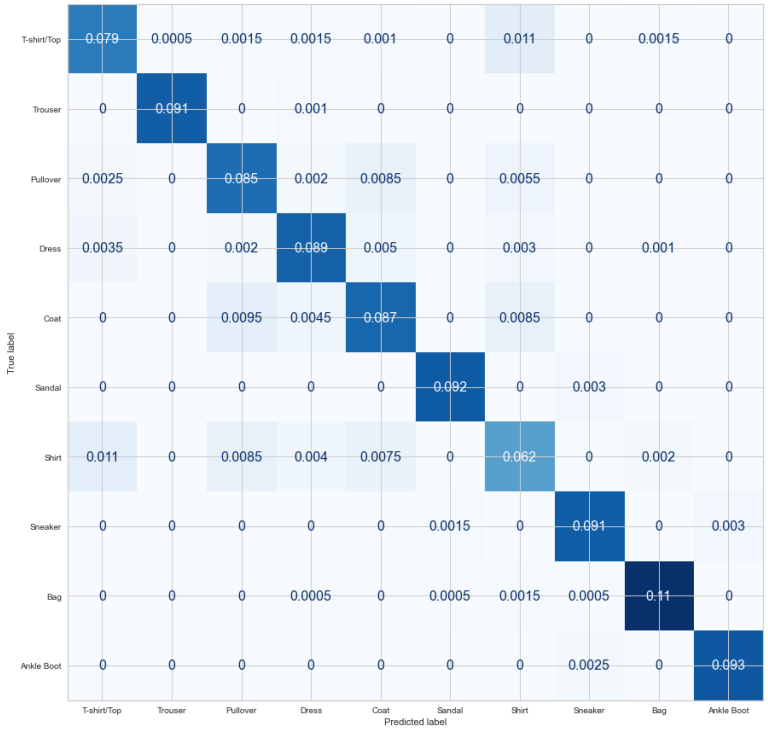


Figure 15: SVM confusion matrix

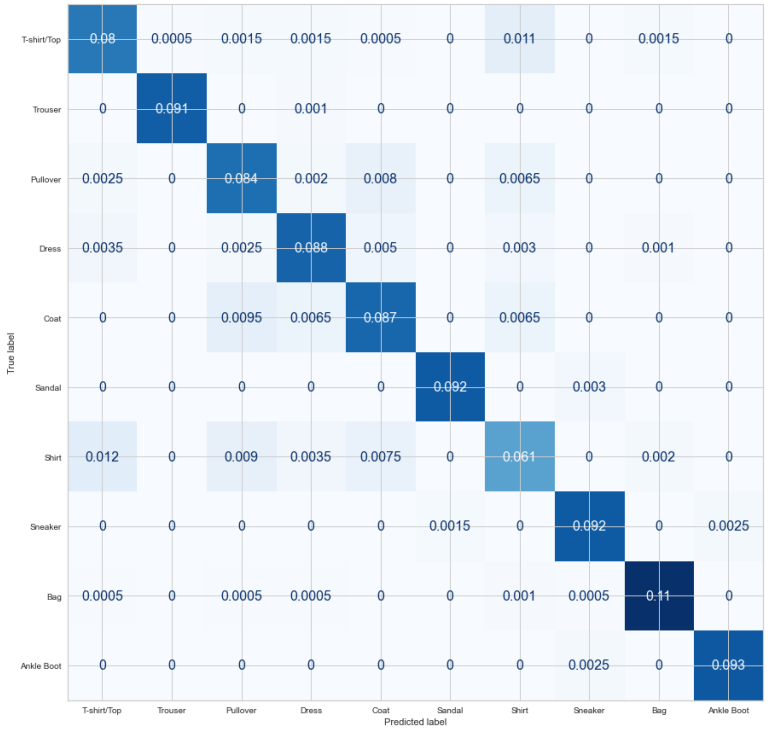


Figure 16: Bagging confusion matrix

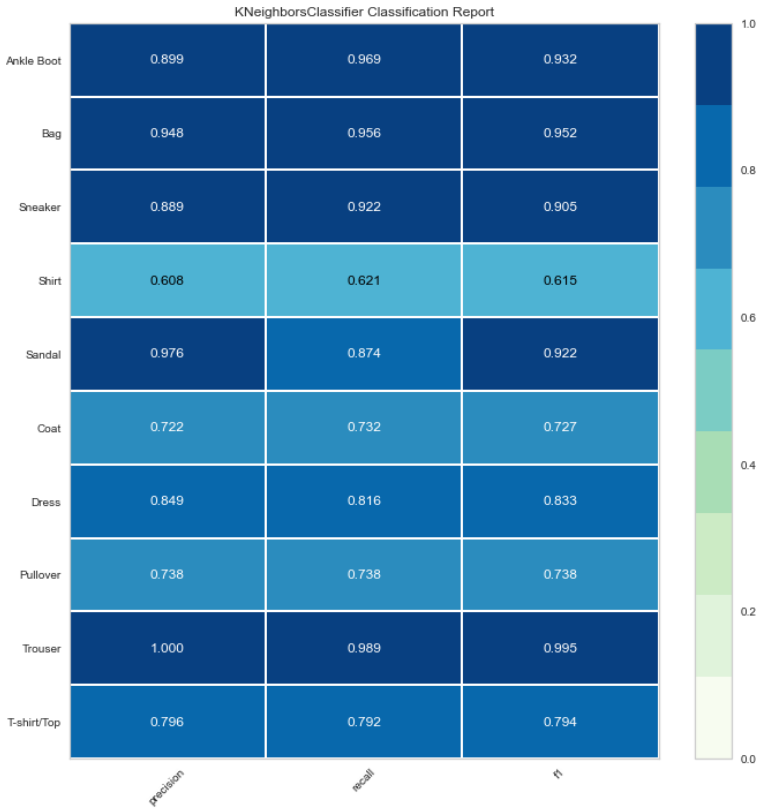


Figure 17: KNN classification report

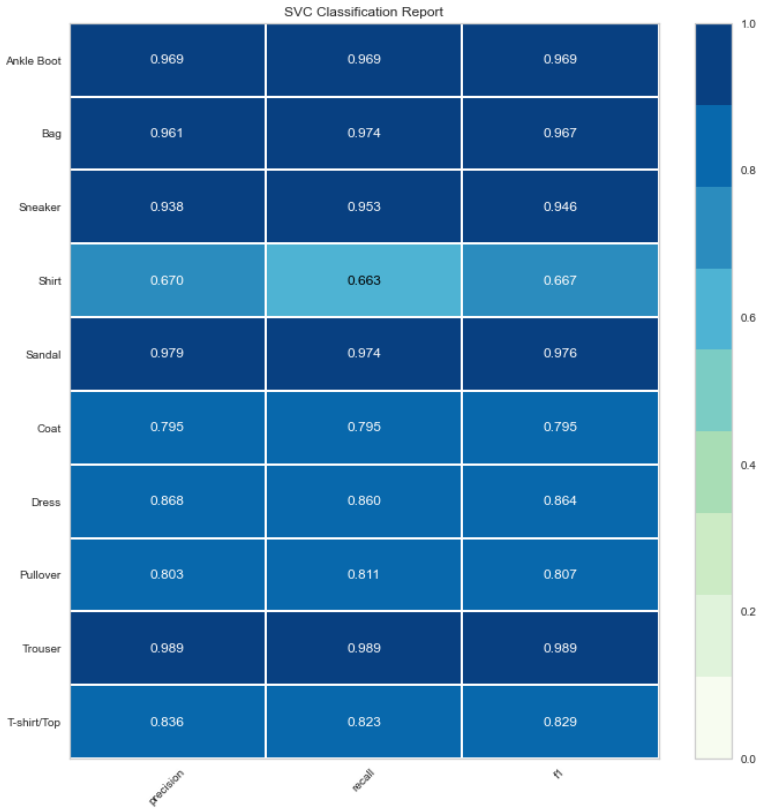


Figure 18: SVM classification report

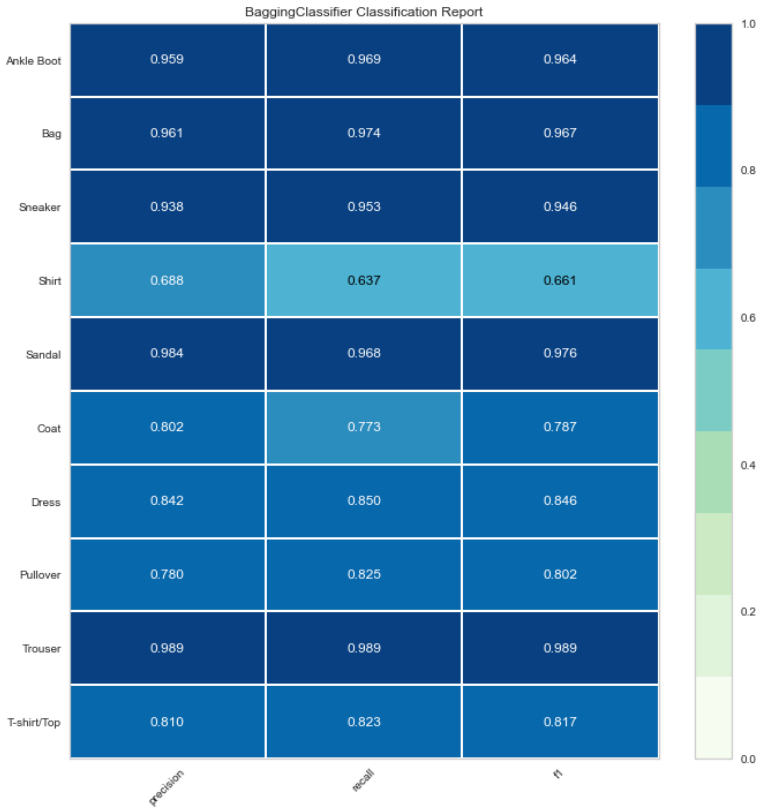


Figure 19: Bagging classification report