Exploring Unsupervised Learning Techniques with the Fashion MNIST Dataset

Carlos Avelar
168605
c168605@dac.unicamp.br

Tiago Chaves
187690
t187690@dac.unicamp.br

***Abstract*—In this project we explore unsupervised learning techniques in the Fashion MNIST Dataset, which is composed of 60,000 labeled 28 × 28 grayscale images for training and 10,000 for testing, divided into 10 classes.**

We begin by setting a baseline with Neural Networks, achieving 91% accuracy in the validation set using a CNN. Then, we move to dimensionality reduction, comparing the use of PCA to Autoencoders (with varying number of principal components and latent vector size, respectively). Finally, we use three clustering techniques in the dimensionally reduced data: K-Means, Mini-Batch K-Means, and Agglomerative Clustering.

Although the best results were obtained by the baseline convolutional net, MLP networks using only 84 features (just 10% of the total 784 = 28 · 28) achieved around 88% accuracy, less than 4% below than the baseline.

I. Introduction

The Fashion MNIST Dataset [1] is supposed to be an easy introduction to machine learning, and to serve as a drop-in replacement for the classic Digit MNIST Dataset (as it is sometimes considered to be too simple).

The dataset is composed of 70,000 28 × 28 grayscale clothing images, divided in 10 classes. It has a train and test split, with 10,000 images already separated for testing, and we decided to use 20% of the remaining data for our validation set, resulting in 12,000 validation and 48,000 training images [1]. Thus, each image input can be considered as a 784-dimensional vector, so we will be using dimensionality reduction techniques to discover if we can reach comparable classification results with effectively less data.

II. Problem

In this assignment we are investigating a labeled dataset, but this is not always the case, in reality, that is almost always the exception. Most datasets don't have labels and that's where the unsupervised learning techniques are essential.

We aim to evaluate different unsupervised learning clustering techniques, and compare the use of dimensionally reduced features — with PCA and Autoencoders — to two baseline networks that use the raw images, in supervised learning manner.

III. Proposed solution

We have used Scikit-Learn's [2] clustering algorithms, performance evaluation metrics, and PCA implementation, while all network models were written using TensorFlow 2.0's Keras API [3].

We present below an overview of the concepts that will later be revisited, and a brief theoretical summary of the clustering algorithms and metrics that were used.

A. Baseline

We have implemented both a MLP and a CNN as baselines for this project. The MLP has two hidden layers with size 128 and 64 and ReLU activation. The CNN has two hidden blocks of a convolutional layer followed by max-pooling, and an output dense layer.

B. Principal Component Analysis (PCA)

The Principal Component Analysis (PCA) algorithm reduces the dimensionality of the data while keeping a certain variance (which can be interpreted as the amount of represented information). It calculates the variance associated with each new feature dimension, so it is possible to select the number of components such that the explained variance is greater than a specified percentage value.

We test 5 variances: 99%, 95%, 90%, 80% and 60%.

C. Autoencoder

An autoencoder is a special neural network architecture that allows for unsupervised learning, as it uses the input as its output. Autoencoders have an interesting shape, that starts big at the input layer and shrinks with each layer in, until getting to the end of the encoding portion of the network, the code/bottleneck layer, then it "grows" back with each layer until it hits input size on the output layer.

It is trained by inputing some data to the network and optimizing it to obtain an output as similiar to the input as possible, thus the code layer is forced to learn an efficient encoding of the input information. That's useful for applications such as dimensionality reduction and data compression.

Our simple autoencoder uses a flattened representation of the images, with both the encoder and the decoder being single layers. We have tested setting the encoded/latent vector size to 24 and 84, based on the results of PCA that kept 80% and 90% variance.

D. K-Means Clustering

The K-means algorithm is one of the better known clustering techniques, as its working is quite simple to understand. It receives as parameters the data and the number of clusters, K, to be formed, so it randomly places K centroids and

[1] Our validation set can be downloaded at drive.google.com/open?id=1-6MiZAtVVybVJvVIQAgRK9ennDyJlYjY

iteratively minimize the squared distance between the data points and the centroids. At each iteration the data points are assigned to the cluster of the centroid nearest to them, and at the end of the assignment the new centroids are calculated by finding the center of the newly formed clusters.

We note that instead of starting with random coordinates, the Scikit-Learn's implementation actually uses the K-Means++ algorithm, which is a more clever seeding of the initial cluster centers [4].

E. Hierarchical Agglomerative Clustering

Hierarchical clustering techniques are usually based on either merging or splitting clusters — bottom-up *vs.* top-down approaches — we have used Scikit-Learn's Agglomerative Clustering algorithm, which successively merges clusters.

In it each data point starts with it's own cluster, then the clusters are merged together by a linkage criteria. The linkage criteria used in this project is the Ward linkage, that minimizes the sum of the squared differences within all clusters, which is in turn quite similar to the K-means objective function. [2]

F. Adjusted Rand Index (ARI)

This function measures the similarity of two assignments, ignoring permutations and with chance normalization [5]. Although it is quite robust, its downside is that it utilizes the ground truth labels, making it useless without them.

ARI values are bounded between -1 and 1, with scores close to 0 indicating random label assignments (i.e. the data and clusters are randomly matched) and 1 being a perfect match.

G. Silhouette Coefficient

Since the ARI is inapplicable if we don't have ground truth labels and, as mentioned, this is the norm, we analyse the Silhouette Coefficient for each cluster as well.

This function is a measure of how similar a sample is to its own cluster (cohesion) compared to other clusters (separation), calculated as $\dfrac{b-a}{max(a,b)}$, where a is the mean distance between a sample and other points in the same class (cohesion), and b is the mean distance to points in the next nearest cluster (separation).

Its values are also bounded between -1 and 1, as -1 indicates incorrect clustering and 1 implies highly dense clustering. Scores around 0 suggest overlapping clusters.

IV. Data Preprocessing

The dataset has 70,000 images divided in 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot.

The 28×28 single-channel images were first divided by 255, to initially bound all values between 0 and 1, then Z-score standardized (i.e. subtracted the mean and divided by the standard deviation), remembering to use the mean and standard deviation of the training set on the validation and test as well.

Lastly, as the labels of the input images were represented by a scalar value from 0 to 9, we one-hot encoded them into arrays of size 10.

V. Results

A. Neural Networks Baseline

We tested two Neural Networks, a fully-connected (MLP) using the flattened image representation and a convolutional network (CNN), both trained with the Adam optimizer [6] and Categorical Cross Entropy loss [7]. We've also added an Early Stop callback with 0.003 `min_delta` and 3 `patience`, and L2 kernel regularization with `l` equal to 0.001 on their output layers to avoid overfitting.

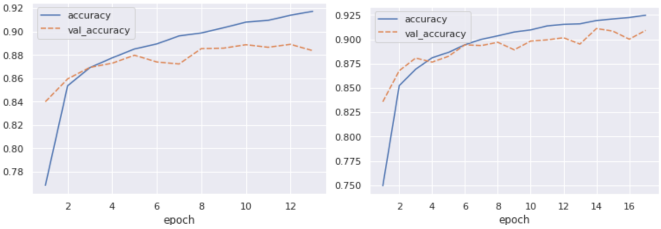
1) MLP: The MLP was composed of two hidden layers of 128 and 64 neurons with ReLU activation, and Softmax for classification on the output layer. It reached 88.9% accuracy on the validation set and 0.8800 combined F1 score (i.e. the harmonic mean of each class' F1 score [8]).

2) CNN: The CNN was composed of two blocks of a 2D Convolutional layer of followed by a 2x2 Max Pooling layer, the first one with 32 filters and 5x5 kernel size and the second with 64 filters and 3x3 kernel size, whose output is flattened and passed to a dense output layer. The activations were also ReLU for the hidden layers and Softmax for the output.

This was our best model, reaching 91.1% accuracy and 0.9035 combined F1 score.

Model	Best Accuracy	Combined F1 score
MLP	0.8887	0.8800
CNN	0.9109	0.9035

Table I: Validation set statistics for the baseline models



(a) MLP accuracy *vs.* epochs

(b) CNN accuracy *vs.* epochs

Figure 1: Accuracy *vs.* epochs for the baseline Neural Networks

B. PCA

We've created 5 models considering PCA for input dimensionality reduction, with 99%, 95%, 90%, 80% and 60% minimum explained variance. The neural network used to fit the models had the same architecture as the baseline MLP, only differing on the Input Layer, which had to be adapted for the new input sizes — that is, the number of principal components used to reach the determined variance values.

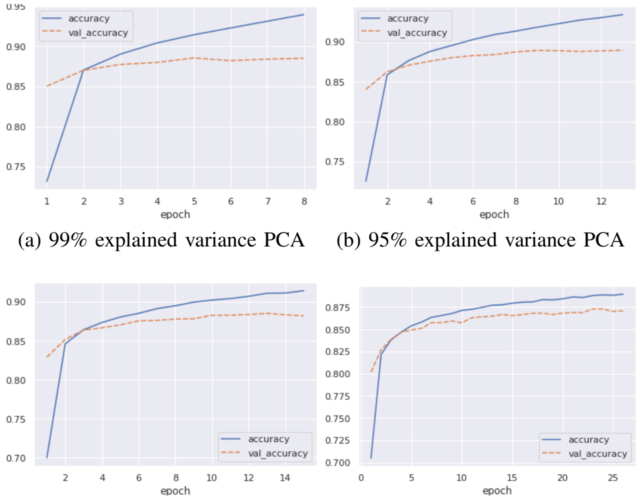
Interestingly, the results obtained with the 99% variance model were slightly worse than the ones of 95% and 90% variance, as the former hit 88.2% accuracy and the later two reached 88.9% and 88.4%, respectively.

Even when using 24 principal components, just 3% of the total 784, the MLP classifier obtained 87.1% accuracy. We only see a big decrease in accuracy when keeping just 60% explained variance, which uses merely 5 components.

The combined F1 scores follow similar rates, as shown on Table II.

Variance	Components	Best Accuracy	Combined F1 score
99%	458	0.8822	0.8735
95%	187	0.8888	0.8794
90%	84	0.8836	0.8734
80%	24	0.8709	0.8600
60%	5	0.7458	0.6923

Table II: Validation set statistics for the models using PCA dimensionality reduction



(a) 99% explained variance PCA

(b) 95% explained variance PCA

(c) 90% explained variance PCA

(d) 80% explained variance PCA

Figure 2: Accuracy *vs.* epochs for models with PCA reduced input

C. Autoencoders

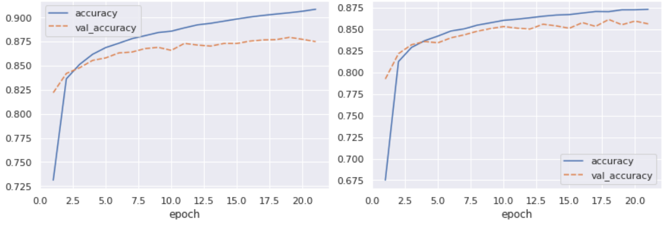
The autoencoder architecture consists of 3 layers: an input of size 784 (as it receives the flattened image), a dense layer with the latent vector size, and an output with the same size as the input. We have tested two different values for the latent dimension based on the results obtained with PCA: 24 and 84, which correspond to number of principal components kept to obtain variances of 80% and 90%.

Using the encoder part of the trained network we can then reduce the input size of a MLP classifier from 784 to the autoencoder's "bottleneck" size, just like we did with PCA.

With a latent dimension of 84, the accuracy achieved by the MLP was 87.7%, around 0.7 percentage points less than when we used the 90% explained variance PCA components. However, with the 24 latent vector size encoder, the accuracy hit 86.0%, which is 1.1 percentage points worse than the 80% variance PCA.

Latent Vector Size	Best Accuracy	Combined F1 score
84	0.8771	0.8660
24	0.8597	0.8455

Table III: Validation set statistics for the models using the Autoencoder's latent representation



(a) Latent vector size of 84

(b) Latent vector size of 24

Figure 3: Accuracy *vs.* epochs for models with autoencoder's encoded representation as input

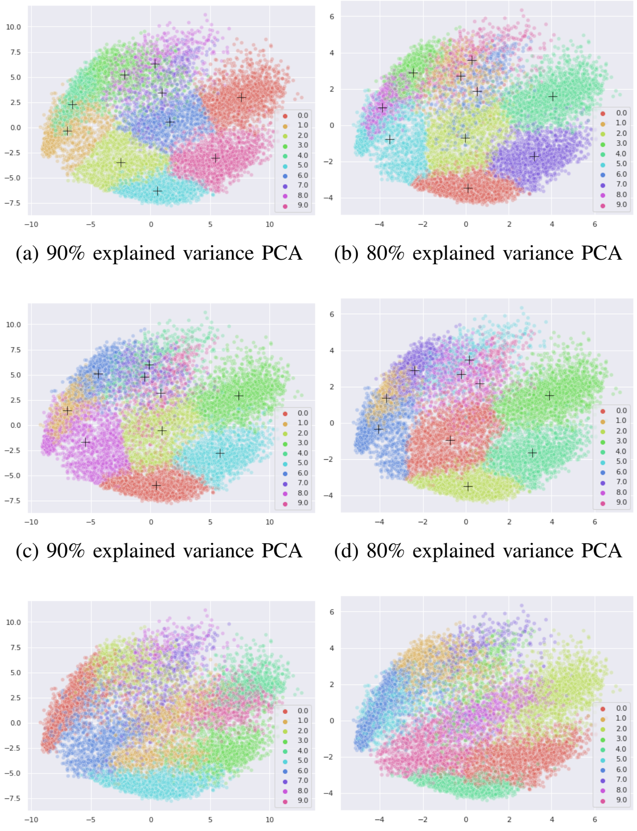
D. Clustering techniques

Considering the PCA reduced features that keep 90% and 80% explained variance — by using the first 84 and 24 principal components — we analysed the outcome of three clustering algorithms [9]: K-Means, Mini-Batch K-Means and Agglomerative clustering. While Agglomerative is a bottom-up approach to hierarchical clustering, the K-Means variants are a form of centroid-based clustering.

To evaluate the quality of the clusters (see Figure 4), we have computed the Adjusted Rand Index (ARI) and the Sillhouette Coefficient, as explained in Section III. The scores for each clustering algorithms is shown on Table IV — recall that, for both measurements, the closer to 1.0 the better the clustering.

Algorithm	PCA used	Sillhouete Coef.	ARI
K-Means	90% var.	0.1679	0.3731
K-Means	80% var.	0.2367	0.3458
Mini-Batch K-Means	90% var.	0.1862	0.3373
Mini-Batch K-Means	80% var.	0.2324	0.3494
Agglomerative	90% var.	0.1476	0.3420
Agglomerative	80% var.	0.1835	0.3830

Table IV: Clustering performance evaluation



(a) 90% explained variance PCA

(b) 80% explained variance PCA

(c) 90% explained variance PCA

(d) 80% explained variance PCA

(e) 90% explained variance PCA

(f) 80% explained variance PCA

Figure 4: Output clusters, with K-Means on the first row, Mini-Batch on the second and Agglomerative on the third

VI. Discussion

Our best model was the baseline CNN, achieving 91.1% accuracy. This is most likely due to convolutional layers being able to exploit the spatial structure information of image data.

Furthermore, the CNN was better at avoiding overfitting than the MLP, and comprised 50% less parameters (50,698 *vs.* 109,386). In the training plots for the MLP baseline in Figure 1 we see that the training and validation lines were starting to distance themselves (starting to overfit) at the end, and that early stopping also made it run for less epochs. A good way of countering that would be to add Dropout layers to the model or increase regularization.

Thus, we have used the convolutional network, with parameters that achieved the best scores during training, on the test set of Fashion MNIST and achieved 91.2% accuracy and 0.9057 combined F1 score, both slightly better than what we hit on the validation set (see Table I). Its confusion matrix is shown in Figure 5.

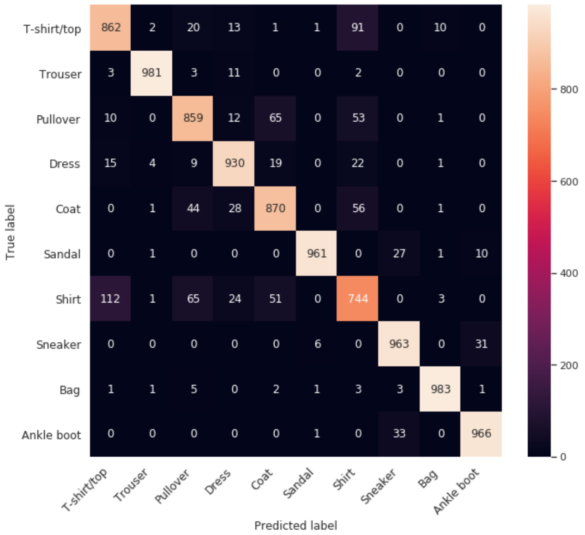


Figure 5: Confusion matrix of the test set image classification, made by our best model (baseline CNN)

Comparing the baseline MLP with the PCA reduced MLP, we can see that the performance gap is really low, or even nonexistent in the case of the 95% variance PCA. That makes the use of PCA really interesting when dealing with too many features or when trying to reduce the computational resources necessary to run a model. The parameter reductions might be quite substantial depending on the size of the model, for instance, the baseline MLP had 109,386 parameter vs. 32,970 with 90% variance PCA, less than a third of the original network.

The autoencoders were a bit disappointing for us, as they didn't yield results as good as PCA, however we see the potential that they have. Even with the simple architecture used, the yielded results were not far behind those of PCA, so there is still room for improvement. Adding more hidden layers and using convolutional layers instead of fully connected would be good changes for anyone trying to get better results with them.

One of the hardest parts of the whole project was making sense of the clusters, as it's quite hard to visualize high dimensional data, and also to match them to the labeled classes. On Figure 6 we see that the clustering algorithm has created two clusters for Bag (while Pullover was mixed with other classes), but only one cluster for Trouser and Dress, and even though the true positives for Sneaker is really good, it comes at the cost of really high false positives with Sandal and Ankle Boot.

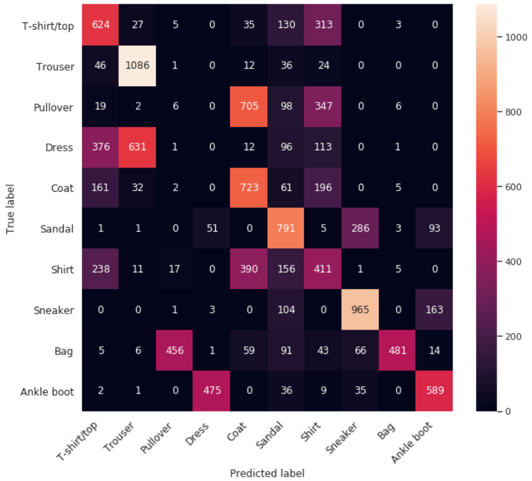


Figure 6: Confusion matrix of the K-means clustering on the 80% variance PCA

VII. Conclusions and Future Work

PCA has proved itself a very good dimensionality reduction technique, it is really interesting to see how discarding more than 75% of the features results in less than 1% point gap on accuracy.

Our autoencoder architecture was really simple but still got results not far from PCA, so we are confident that a more complex model could outperform PCA, such as using convolutional layers.

Lastly, the clustering algorithms didn't get incredible results, but considering the fact that they didn't need labels, it's not so bad. One experiment that we didn't have the time to run, but would be interesting considering unsupervised learning, would be to use the clusters as labels for training (as if we didn't have ground truth labels). Another interesting experiment would be to do clustering inside of clusters that had poor quality, to see if it would split some of the classes that were mapped to the same cluster on our experiments.

The notebook with the source code and the datasets used can be seen on github.com/laurelkeys/machine-learning (inside the `assignment-3` folder).

References

[1] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 1

[2] F. Pedregosa et al., "Scikit-learn: Machine learning in Python." 1, 2

[3] TensorFlow Core r2.0, "Module: tf.keras." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras 1

[4] "K-Means++ — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/K-means%2B%2B 2

[5] Scikit-learn, "Adjusted Rand index." [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#adjusted-rand-index 2

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization." [Online]. Available: http://arxiv.org/abs/1412.6980 2

[7] "Cross entropy — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Cross_entropy#Definition 2

[8] "F1 score — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/F1_score 2

[9] "Cluster analysis — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Cluster_analysis 3

Links accessed on November 2019.