ML Project - Image Classification

Pietro Sittoni
`Matricola:` 2040637

Irene Caria
`Matricola:` 2040639

1. Introduction

In this project we worked on Image Classification using Machine Learning techniques. Our Image Classification problem is a supervised learning one: given a data set, we want to learn a function which captures the information content in the examples and therefore be able to predict the class of a new example. The goal is to classify new clothing images into one of 10 possible classes. To do this we implemented 4 different models: a regularized Support Vector Machine, a Random Forest, a K-Nearest Neighbors and a Neural Network. We compared them and look for the optimal method. We got the best results with the Support Vector Machine, but it was the slowest. However, it must be emphasized that, as we will see, there are common mistakes between SVM, Random Forest and K-NN and we tried to identify them and formulate a reason why they were made.

2. Dataset

We worked with the Fashion-MNIST dataset of Zalando's article images which consists of a training set of $60,000$ examples and a test set of $10,000$. Each example is a 28x28 grayscale image representing an item of clothing, divided into 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

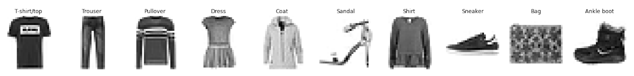


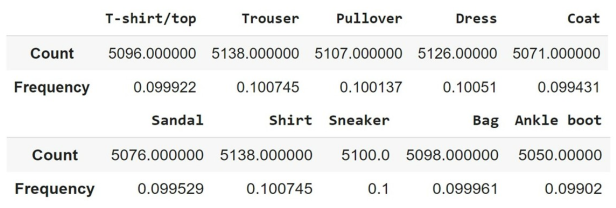
Figure 1: One image for each class

To be able to compare our models, we decided to extract 15% of the training set in order to use it as a validation set. Here we report the final shapes of our sets:

- **X_train** : $(51000\,,784)$ **Y_train** : $(51000\,,)$
- **X_val** : $(9000\,,784)$ **Y_val** : $(9000\,,)$
- **X_test** : $(10000\,,784)$ **Y_test** : $(10000\,,)$

2.1. Classes

The classes, in the training set, are almost balanced, the proportion of element per class is around 10%; the biggest has 5138 element, and the smallest has 5050, hence we decided not to rebalance the classes. See Figure 2



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
Count	5096.000000	5138.000000	5107.000000	5126.00000	5071.000000	5076.000000	5138.000000	5100.0	5098.000000	5050.00000
Frequency	0.099922	0.100745	0.100137	0.10051	0.099431	0.099529	0.100745	0.1	0.099961	0.09902

Figure 2: Classes' frequency table

2.2. Features

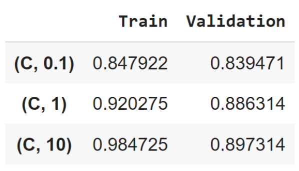
In first instance we noticed that the element of the features space were sparse (the vector with the smallest norm had norm 548.9098 and the vector with the highest norm had norm 5839.711). Therefore we decided to rescale the vectors of the features space. The reason behind our choice, is related to the fact that models are very often affected by hadling sparse data, in particular Neural Network. We chose the Standard Scaler and after rescaling, the vector with the smallest norm had norm 12.54 and the vector with the highest norm had norm 230.75.

3. Models

As explained in the introduction, to handle this classification problem, we implemented 4 different models: Support Vector Machine, Random Forest, K-Nearest Neighbors and Neural Network, using different combinations of hyperparameters and we selected the best one using the accuracy score, since the classes were balanced. We started with a 5-fold Cross Validation for the first three models, but the running time was too high. We tried a 3-fold Cross Validation and we saw that the accuracy scores did not change, so in the following we reported only the 3-fold version and the hyperparameters that we evaluate for each model.

3.1. Support Vector Machine

We chose a regularized version of the SVM and we tried to optimize the regularization parameter (indentified by the letter C) in the set $\{0.1, 1, 10\}$. We used Rbf as kernel function, and since it is a multiclass classification, we used a one-vs-rest strategy. It is important to note that it is very time inefficient; this is due to the fact that Kernelized SVMs require the computation of a distance function between each point in the dataset, which is the dominating cost of $O\left(n_{features} \text{ x } n^2_{observations}\right)$. As we can see from the Figure 3, increasing the C hyperparameter over 10 produces overfittng and increases considerably the execution time.

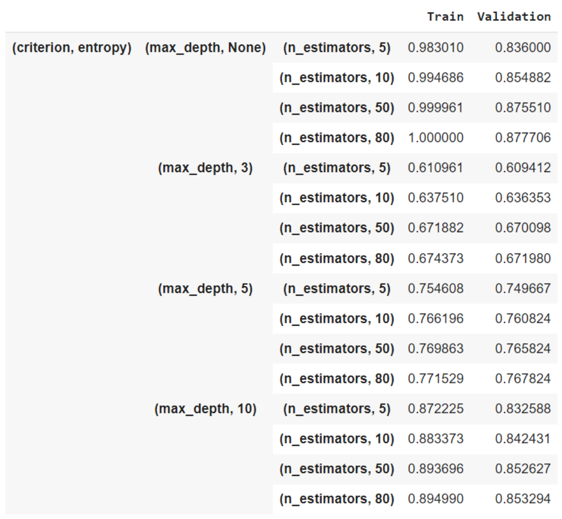


	Train	Validation
(C, 0.1)	0.847922	0.839471
(C, 1)	0.920275	0.886314
(C, 10)	0.984725	0.897314

Figure 3: CV accuracy results for the SVM

3.2. Random Forest

For the Random Forest we tried to optimize the number of trees in the set $\{5, 10, 50, 80\}$ and their depth in $\{\textit{None}, 3, 5, 10\}$ where *None* indicates that the nodes are expanded until all leaves are pure or until all leaves contain less than the minimum number of samples required to split an internal node. We used entropy as the function for evaluate the split. See Figure 4.



			Train	Validation
(criterion, entropy)	(max_depth, None)	(n_estimators, 5)	0.983010	0.836000
		(n_estimators, 10)	0.994686	0.854882
		(n_estimators, 50)	0.999961	0.875510
		(n_estimators, 80)	1.000000	0.877706
	(max_depth, 3)	(n_estimators, 5)	0.610961	0.609412
		(n_estimators, 10)	0.637510	0.636353
		(n_estimators, 50)	0.671882	0.670098
		(n_estimators, 80)	0.674373	0.671980
	(max_depth, 5)	(n_estimators, 5)	0.754608	0.749667
		(n_estimators, 10)	0.766196	0.760824
		(n_estimators, 50)	0.769863	0.765824
		(n_estimators, 80)	0.771529	0.767824
	(max_depth, 10)	(n_estimators, 5)	0.872225	0.832588
		(n_estimators, 10)	0.883373	0.842431
		(n_estimators, 50)	0.893696	0.852627
		(n_estimators, 80)	0.894990	0.853294

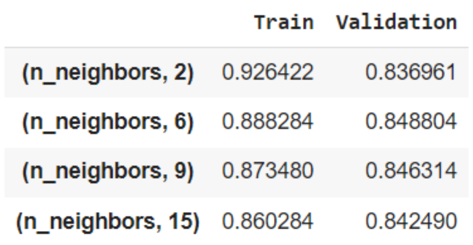
Figure 4: CV accuracy results for the Random Forest

3.3. K-Nearest Neighbors

For the K-NN we tried to optimize the number of relevant neighbors in the set $\{3, 6, 9, 15\}$. As we can see from the Figure 5, increasing the value of k, the bias rises and decreasing the value of k, the variance rises.

3.4. Neural Network

In the Neural Network, we used a sequential model. We built 12 different Neural Network, we tried to optimize the number of hidden layers in the set $\{2, 4, 8\}$ and for each of them we did vary the number of units in $\{15, 50, 100, 300\}$. The input and the output layers are vectors with respectively 784 and 10 components. For each hidden layer we used, as activation function, a Relu, and for the last layer a Softmax. For practical reasons we set an early stopping parameter, due to the fact that the number of features and examples was high, so to mitigate the running time we set it up. Even if the simplest NN with 2 hidden layers and 15 units for each of them (see Figure 6) gave us these accuracy results: train 0.9035 and val 0.8588, we decided to improve the model with the optimization written above. We enhanced the train scores of almost 0.07 and the val scores of 0.03 (see Figures 7, 8).



	Train	Validation
(n_neighbors, 2)	0.926422	0.836961
(n_neighbors, 6)	0.888284	0.848804
(n_neighbors, 9)	0.873480	0.846314
(n_neighbors, 15)	0.860284	0.842490

Figure 5: CV accuracy results for the K-NN

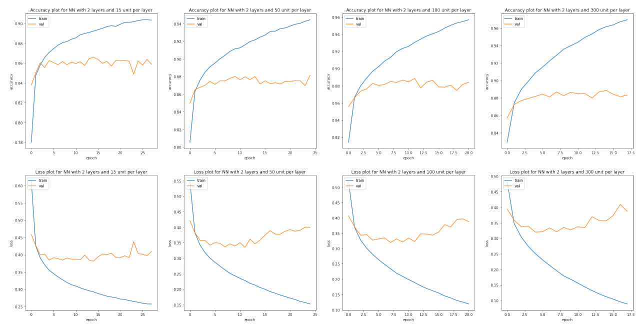


Figure 6: Accuracy and loss plots for NN with 2 layers

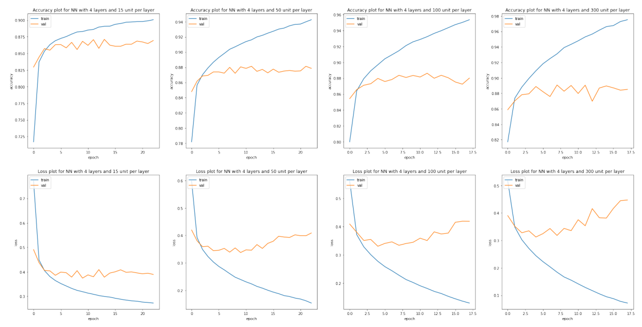


Figure 7: Accuracy and loss plots for NN with 4 layers

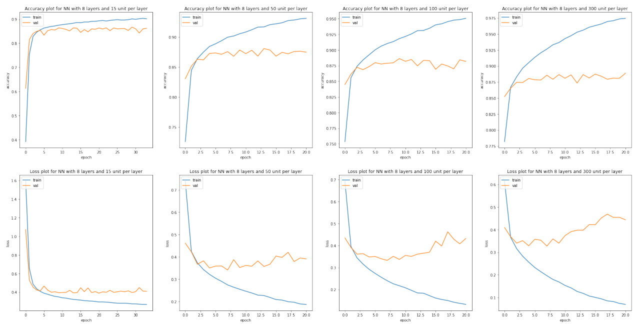
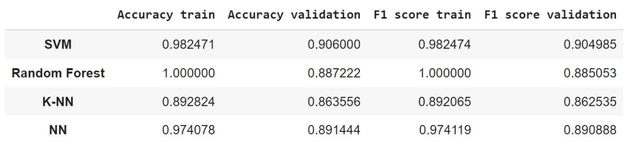


Figure 8: Accuracy and loss plots for NN with 8 layers

4. Performance evaluation

After defining and fitting our models, we determined the best among them by comparing the accuracy and f1 score, on the validation set.



	Accuracy train	Accuracy validation	F1 score train	F1 score validation
SVM	0.982471	0.906000	0.982474	0.904985
Random Forest	1.000000	0.887222	1.000000	0.885053
K-NN	0.892824	0.863556	0.892065	0.862535
NN	0.974078	0.891444	0.974119	0.890888

Figure 9: Comparison on train and val sets

K-NN's results were the worst, probably was the worst model because the distance in high dimension is less meaningful than lower, but we tried to go in deep and analyse better the mistake. So we decided to create the confusion matrices of the K-NN and Random Forest on the validation set, to see which samples were the most misclassified and whether such errors were common in both (see Figure 10).

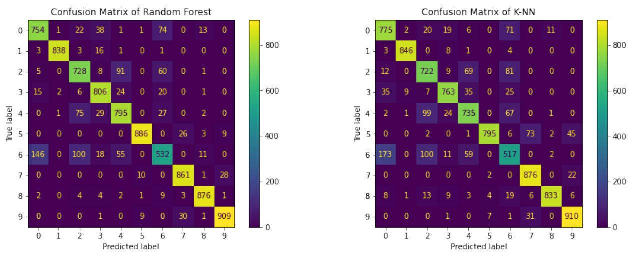


Figure 10: Confusion matrices of RF and K-NN on val set

Both Random Forest and K-NN make most of the mistakes by classifying Shirt (class 6) as T-shirt/top (class 0). If we look at the images of these misclassified clothes we realize that these samples seem look more like T-shrirt/top than Shirt. We reported in the following Figure 11 two misclassfied samples to have an idea (see the script for more examples). We can also notice that our models classify correctly bags, shoes and trousers, while they have difficulty

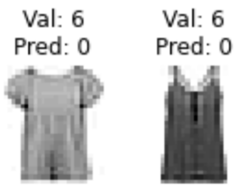


Figure 11: Two misclassfied samples

recognizing t-shirts, shirts, dresses, coats and pullovers among themselves. Then, we thought that these errors could be due to a rappresentative problem of the dataset. For this reason we decided to analyze also the errors of our best model, i.e. SVM, on the validation set (see Figure 12). Also the SVM made the same mistake, although with less frequency.

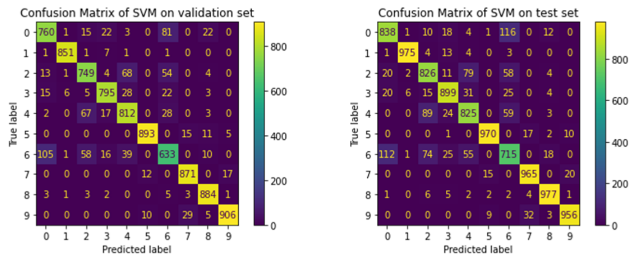


Figure 12: Confusion matrix of SVM on val and test set

This problem could help us to interpret the behavior of the NN graphs in Figures 6, 7, 8. While in the training set the accuracy increases with the passing of the epochs, the validation's accuracy grows in the early epochs and then it stabilizes. So the gap between the training and the validation accuracy is probably due to the fact that the difference between some samples of the t-shirts, shirts, dresses, coats and pullovers classes are not so evident.

The results obtained with the Neural Network are slightly lower than those of the SVM but still definitely good. Moreover, it is important to underline that the Neural Network is much more time efficient than the SVM.

4.1. Test performance

We evaluated our best model (i.e. the SVM) on the test set and we obtained these results:

- accuracy on test set: 0.8946
- f1 score on test set: 0.8944689382163171

As we can see, they are slightly worse than the validation ones. Also for the test set we created the confusion matrix to see which samples were misclassified (see Figure 12).