ML Project - Image Classification: the Fashion-MNIST Challenge

Leonardo Schiavo

September 2022

1 Introduction

The fashion-MNIST Challenge is a Zalando's Machine Learning project. Its aim is to implement a classification algorithm of clothes pictures. In particular, we want an algorithm that, given an image, tells us which type of clothes is pictured. We studied many algorithms and got some results on the accuracy they achieve.

2 Dresses Dataset

We were provided with a database of 60,000 pictures of clothes. Each sample is a 28x28 pixel grayscale image associated with a label from 10 classes. The classes are: Ankle boot, T-shirt, Dress, Pullover, Sneaker, Sandal, Trouser, Coat, Bag, Shirt. The ten classes are perfectly balanced. Here follows an example from every class in the same order as listed before.

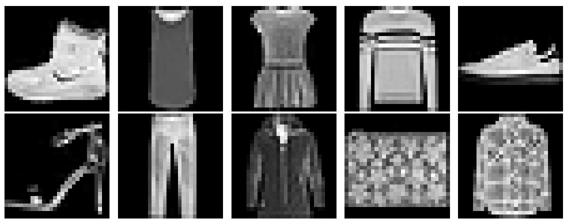


Figure 1: One example per each class

Actually we used a modified database. We said that every image was a 28×28 pixel grayscale picture, where every pixel is described with a number from 0 to 255 and the smaller numbers closer to zero represent the darker shade while the larger numbers closer to 255 represents the lighter or the white shade. So each image is associated with a 28×28 matrix where each numerical entry (i, j) correspond to the grey coloration of the pixel in position (i, j). Next we reshaped every matrix into a row vector of dimension 748, which is $28 * 28$. The database we used was attached with an index where each image is labeled with its class. For our purposes we divided the original training set into a new training set of 50,000 images and a validation set of 10,000 images. The original test set of 10,000 is remained the same. This division is not as precise as we just said since we used the shuffle option in the splitting, hence the sets are still balanced but not perfectly. First of all we considered the option of scaling the training examples: we used MinMax and Standard Scalar. The first is only a projection onto $[0, 1]$ of the sample, the second distributes the sample onto a standard Gaussian distribution (mean 0 and variance 1). Here we began to considering machine learning algorithms, since our data now is in a good shape. We used different methods and gathered data by testing them. Now we will describe the theory behind our algorithms and their results. Note that all the results are compared by the accuracy since there is no difference between false positive and false negative, they are equally not good.

3 Perceptron

The simplest classificator, the Perceptron, works for binary classification. It corresponds to a two-class model in which the input vector $\mathbf{x}$ is first transformed using a fixed nonlinear transformation to give a feature vector $\phi(\mathbf{x})$, and this is then used to construct a generalized linear model of the form

$$y(\mathbf{x}) = f(w^T \phi(\mathbf{x}))$$

where the nonlinear activation function $f()$ is given by a step function of the form

$$f(a = \begin{cases} 1 & \text{if } a \geq 0, \\ -1 & \text{if } a < 0 \end{cases}$$

We consider an alternative error function known as the *perceptron criterion*. To derive this, we note that we are seeking a weight vector w such that patterns $\mathbf{x}_n$ in class $\mathcal{C}_1$ will have $\mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}_n) > 0$, whereas patterns $\mathbf{x}_n$ in class $\mathcal{C}_2$ have $\mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}_n) < 0$. Using the $t \in \{-1, +1\}$ target coding scheme it follows that we would like all patterns to satisfy $\mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}_n) t_n > 0$. The perceptron criterion associates zero error with any pattern that is correctly classified, whereas for a misclassified pattern $\mathbf{x}_n$ it tries to minimize the quantity $-\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) t_n$. The perceptron criterion is therefore given by

$$E_{\mathrm{P}}(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}_n t_n$$

where $\mathcal{M}$ denotes the set of all misclassified patterns. The contribution to the error associated with a particular misclassified pattern is a linear function of w in regions of w space where the pattern is misclassified and zero in regions where it is correctly classified. We now apply the stochastic gradient descent algorithm to this error function and obtain the classification. However our problem is a 10 classes exclusive classification. Hence we have to use a one versus rest or a one versus one classification method. The first one is quite not convenient since the "rest" class is going to be too much bigger with respect to the separated class. Thus we decided to use only the one vs one approach: let K be the number of classes, we train $K(K1)/2$ different 2-class perceptrons on all possible pairs of classes, and then to classify test points according to which class has the highest number of 'votes'. In practice we implemented the perceptron one versus one, trained it on the new shuffled *small* training set of 10,000 images and used the *small* validation set (2,000 examples) to choose the better learning rate and preprocessing technique. After the validation step we trained a perceptron classifier with the hyperparameter found in the previous step on the small dataset. This method allows faster computation and more freedom in the grid search. The question is if the hyperparameter found on the small dataset are going to generalize well on the big dataset, hopefully yes but sometimes we are going to check it. This procedure is repeated throughout the analysis of the algorithms in the next sections. The third step is the test of the chosen perceptron onto the test set, the big one provided by Zalando.
The optimal learning rate on the small dataset was $\eta_s = 0.01$ and the preprocessing with better accuracy was MinMax. The accuracy results are in the following table.

Dataset	Small	Big
Train	0.8896	0.7973
Validation	0.8170	0.7765
Test	*	0.7705

The results are not exciting: there is quite a gap between the accuracy on the small training set and on the small validation set. On the contrary the accuracies on the big dataset are all near, but really too low since none of the reache 80%. The problem is that probably the perceptron is too simple as a model, the architecture is too naive to perceive the complexity of the classification problem, hence there could be a bias problem. Both accuracies on training ad validation set are low, high variance could be a problem, too. Just for curiosity we tried a grid search of hyperparameter on the big dataset to see if the problem lies in the not good generalizing behaviour of the hyperparameter. The grid search on the big dataset suggested a different learning rate of $\eta_b = 0.001$ but the same preprocessing technique. With those choices we trained a perceptron classifier witch gave the following accuracies: train accuracy: 0.84272, validation accuracy: 0.825 and test accuracy: 0.8148. The improvement is circa of the 2%, this is the cut we have to pay for use the smaller dataset to search for the optimal hyperparameter. The results is not satisfying, this leads us to the conviction that the perceptron is indeed not the right model.

4 k-Nearest Neighbours

In k-Nearest Neighbour an object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours. The figure gives us a better idea of what's happening.

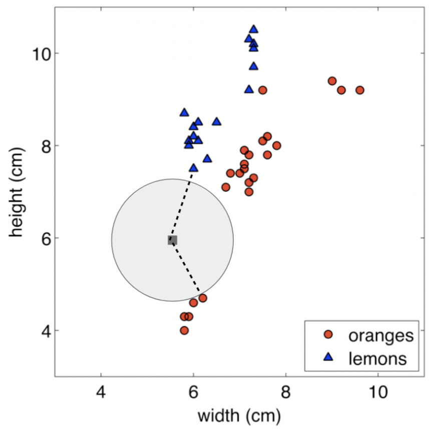


Figure 2: K-NN Model Example

Here we used Sklearn library and the function KNeighborsClassifier. As before we performed a grid search on the small dataset. The results are that the best k is 5 and the best preprocessing is the standard scaler.

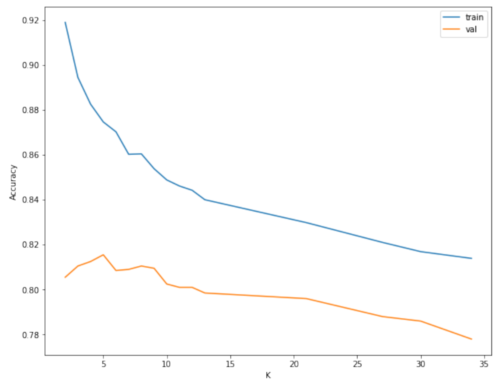


Figure 3: K-NN: Accuracy at different k

As usual we developed a knn algorithm on the big dataset with the following results:

Dataset	Small	Big
Train	0.8746	0.89976
Validation	0.8155	0.85040
Test	*	0.85230

The results are good enough but, lead by curiosity and suspiciousness we tried a grid search on the big dataset. This was definitively quite time expensive since the computation on the 3 type of preprocessing ad k ranging from 2 to 10 required 3 hours. The choose for 10 was both empirical and theoretical: we saw that as k increases as the accuracy decreases. This fact is clear by the theory since if we take k equal to the total number of traning examples than every example is going to fall in the same class leading to an accuracy of 10%. The results changed a bit the situation: k optimal now is 4, not 5 anymore, and the preprocessing chose was not preprocessing at all, not standard scaler anymore. The results are in the following table: the first column indicates the results on the big dataset with the hyperparameters searched on the small dataset; the second column instead is with the hyperparameters searched in the big dataset itself.

Dataset	Big (small hp)	Big (Big hp)
Train	0.89976	0.90768
Validation	0.85040	0.85430
Test	0.85230	0.85360

The improvement is insignificant. Is it so curious that the optimal k is changed? I would say no, since if we count by majority vote between 5 or 4 the result should be the same if the classes are quite separated. We will see that class 0 (t-shirt/top) and class 6 (shirt) are not so separated by the point of view of decision trees and random forests. Are those classes similar for all the algorithms we are investigating? I would say yes, probably it is an intrinsic property of the dataset and to solve this problem we should amplify the features that differentiate those classes.

5 Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Here we provide a simple example of binary classification of fruits.

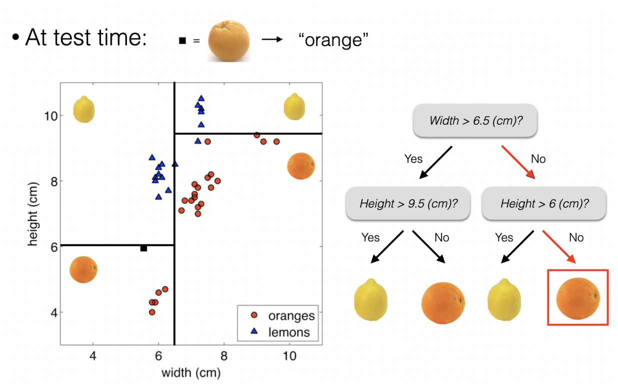


Figure 4: Structure Decision Tree

In our case we used the Sklearn library and the functions DecisionTreeClassifier and GridSearchCV. We used entropy metric as criterion and the GridSeachCV to find the best tree combining our parameters. The results on the small dataset were: max depth: 11, min samples leaf: 5, min samples split: 18. We saw that the best preprocessing was not preprocess at all and we reached the following results on both small and big dataset:

Dataset	Small	Big
Train	0.8829	0.85984
Validation	0.7735	0.8117
Test	0.776	0.8071

Even here the results are not great, we are not yer at the threshold of 90%. The usual question we tried to answer: was the grid search on the small dataset good enough? We could do a grid search on the big dataset in a bit more than 1 hour. The results of the grid search on the big dataset were max depth: 11, min samples leaf: 8, min samples split: 20 and still no preprocessing is better. The results are in the following table: the first column indicates the results on the big dataset with the hyperparameters searched on the small dataset; the second column instead is with the hyperparameters searched in the big dataset itself.

Dataset	Big (small hp)	Big (Big hp)
Train	0.85984	0.85702
Validation	0.8117	0.8107
Test	0.8071	0.807

The improvement, as in the section before, are insignificant.

6 Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and to control over-fitting. Hence it uses bagging and feature randomness to combine trees, this way it reduces the variance of the "original" classifier (decision tree) and then makes an ensemble out of it. We can see an example here.

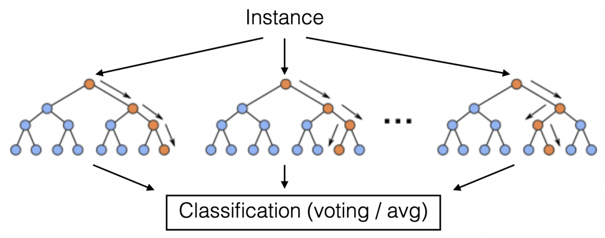


Figure 5: Structure Random Forest

The grid search granted us the hyperparameters max depth: 11, min samples leaf: 5, min samples split: 18. Our results, with the usual procedure over the MinMax preprocessed small datasets and over the not preprocessed big datasets are:

Dataset	Small	Big
Train	1.0	1.0
Validation	0.866	0.8815
Test	0.8545	0.8766

Note that the choice of MinMax for the small datasets and of the original datasets for the big leads to the best accuracies among the accuracies we found. Here it is clear that the algorithm has high variance, hence it is overfitting since the error in the training set is negligible but the error in the validation set is 12% to 14%. How can we solve this problem? One way is to tune the hyperparameter max depth to be small, for example between 5 and 15 and see how overfitting grows as max depth grows. The plot in 6 shows how the accuracy of the training set grows as the max depth grows provoking overfitting.

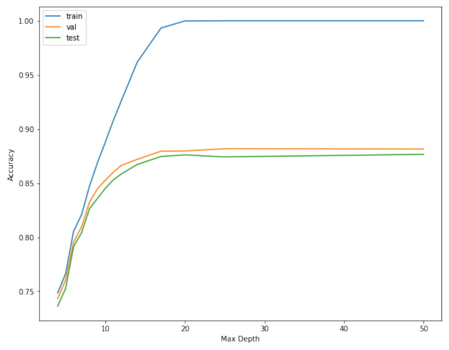
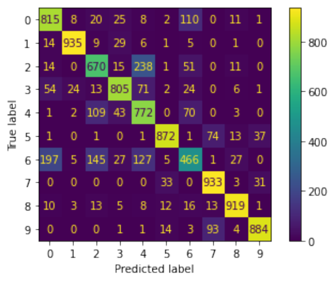


Figure 6: Overfitting of the Random Forest

In this case the grid search performed on the big dataset gives the same results as the grid search performed on the small dataset, so with some time spent we are safe to say this time hyperparameters generalize perfectly.

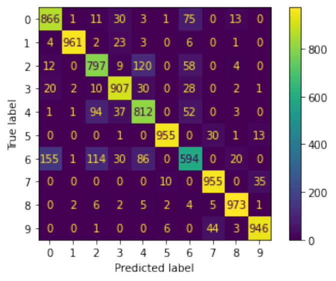
7 Detour on misclassified examples by Decision Tree and Random Forest

One important question in where an algorithm produces the error, so which examples are misclassified by the algorithm? Just for the Decision Tree and the Random Forest alorithm we decided to answare this question. Both, as we said before, have some trouble distinguishing between class 0: t-shirt/top and class 6: shirt.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	815	8	20	25	8	2	110	0	11	1
1	14	935	9	29	6	1	5	0	1	0
2	14	0	670	15	238	1	51	0	11	0
3	54	24	13	805	71	2	24	0	6	1
4	1	2	109	43	772	0	70	0	3	0
5	1	0	1	0	1	872	1	74	13	37
6	197	5	145	27	127	5	466	1	27	0
7	0	0	0	0	0	33	0	933	3	31
8	10	3	13	5	8	12	16	13	919	1
9	0	0	0	1	1	14	3	93	4	884

Figure 7: Decision Tree Confusion Matrix



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	866	1	11	30	3	1	75	0	13	0
1	4	961	2	23	3	0	6	0	1	0
2	12	0	797	9	120	0	58	0	4	0
3	20	2	10	907	30	0	28	0	2	1
4	1	1	94	37	812	0	52	0	3	0
5	0	0	0	1	0	955	0	30	1	13
6	155	1	114	30	86	0	594	0	20	0
7	0	0	0	0	0	10	0	955	0	35
8	0	2	6	2	5	2	4	5	973	1
9	0	0	1	0	0	6	0	44	3	946

Figure 8: Decision Tree Confusion Matrix

8 Support Vector Classification

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. Those methods are based on Vapnik-Chervonenkis theory (VC theory). Here we used the Sklearn library, too. As usual we proceded with a grid search over the small dataset and we found the following best hyperparameters: C: 10 and kernel: rbf. The radial basis function kernel is:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$$

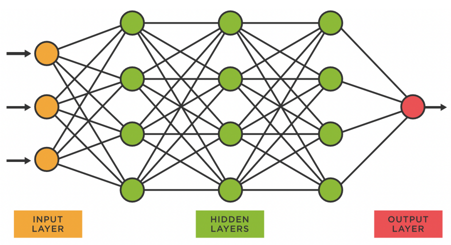
Kernel Function is used to transform n-dimensional input to m-dimensional input, where m is much higher than n then find the dot product in higher dimensional efficiently. The main idea to use kernel is: A linear classifier or regression curve in higher dimensions becomes a Non-linear classifier or regression curve in lower dimensions. The results we got on the small dataset and the big dataset both without preprocessing and with the just fixed hyperparameters are:

Dataset	Small	Big
Train	0.9836	0.97226
Validation	0.867	0.9028
Test	0.8655	0.8968

Unfortunately here suspiciousness have had to be put at bay: the grid search on the big dataset was computationally not possible. Maybe with more time and interest in this algorithm some idea can be developed: one grid search on the small dataset took 10 minutes, maybe one can try to partition the training set in small datasets and grid search onto them and after take a sort of mean, so via cross validation. The results here suffer from high variance and there is clearly an overfitting problem. How can we solve it? Here the computation become costly so we could not create a plot as we did in the Random Forest case, but we think the problem resides in the not good tuning of C and the kernel function: maybe better hyperparameters can reach better accuracies.

9 Neural Network

A neural network is a structure that tries to copy human brain. A single (artificial) neuron is exemplified with many input wires, the dendrites, one cell body, and an output wire, the axon. For every artificial neuron is defined an activation function $h_\theta(x) = f(\theta^T x)$, that decides if the neuron is activated or not, i.e. if the neurons in our brain has to activate. An (artificial) neural network is just a set of neuron connected to each other. Here we provide an example in the figure below.



With the aim to build a neural network we rely on Keras library. Note that in order that the Neural Network works with Keras data have to be scaled, so we are not using unpreprocessed data. As usual we tried a grid search to find the best hyperparameters. The best hyperparameters we found are epochs: 100, architecture: [784, 382, 382, 10], patience: 5, optimizer: Adam, dropout: 0.2, mini batch size: 256, preprocessing: MinMax. We proceded to train a neural network with a bit more of patience (15) and the same hyperparameters. We got the following results:

Dataset	Small	Big
Train	0.97230	0.97252
Validation	0.89450	0.89810
Test	0.89400	0.89740

As we already know from the previous analysis this results are affected by an high variance problem, since the training error is low but the validation error is high. Thus, the neural network is overfitting. How can we solve this problem? We can try a grid search on the big dataset, but this is quite expensive, we can try to adjust and tune the many hyperparameters we have: for example we can try to increase the dropout probability. The dropout probability is the probability that a neuron "dies" and so do not contributes in the propagation anymore for that epoch. The dropout technique is a regularization tool, other regularization tool can be useful to solve this problem.

10 Conclusion

We studied the Zalando's classification problem Fashion MNIST Challenge with various algorithm such as Perceptrom, Decision Tree, Random Forest, SVM, Neural Network and k-NN.

Algorithm	Val acc	Test Acc
Perc	0.825	0.814
k-NN	0.85430	0.85360
DT	0.8107	0.807
RF	0.8815	0.8766
SVM	0.9028	0.8968
NN	0.89810	0.89740

The best algorithm on the validation set is SVM and it is the only algorithm that reaches the 90% theshold accuracy mark on the validation set. The results are not quite satisfactory, we could not reach more than 90%, probably to advance we tune better our hyperparameters or to choose different architectures like Convolutional Neural Network.