Machine Learning project: analysis of the Fashion-MNIST dataset

Marcin Sidorowicz

February 2020

1 Introduction

Fashion MNIST is a dataset introduced by Zalando to have a similar role as the original MNIST dataset - as a benchmark for classificating and generative models. It consists of 28x28 black and white images, with each pixel encoded as an integer from 0 to 255. Instead of handwritten digits, the pictures show different parts of clothing. In this project, we will try to apply different classification methods learned on this course. In particular, we will compare the performance of those methods to the original MNIST, trying to explain possible differences in accuracy.

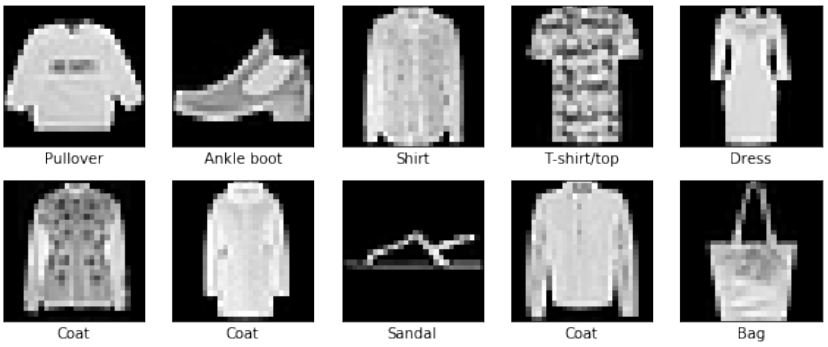

Fig. 1. Example images from the dataset

2 k-Nearest Neighbors

After testing with different distance metrics (including Euclidean distance, L1 distance and Hamming distance), we've achieved around 85% accuracy. This stands in big contrast to the original MNIST, on which this method is known to achieve accuracy of above 95%. This may be a result of the original images being perfectly cropped, thresholded and distinct among different classes - unlike in

our dataset, where for example *T-shirts, pullovers, coats, shirts* are visually very similar.

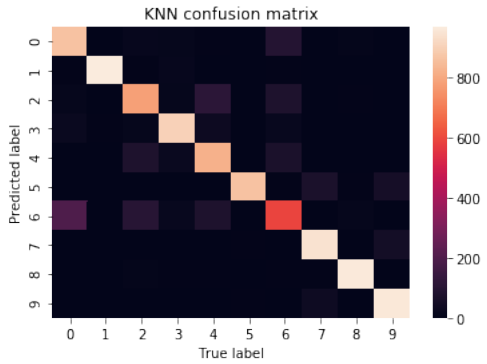

Fig. 2. Confusion matrix for KNN classifier

Note that in the above confusion matrix, we can distinguish two groups of seemingly similar classes - $\{0, 2, 4, 6\}$ and $\{5, 7, 9\}$. This pattern will continue to appear with other models.

3 SVM (and some dimensionality reduction)

This method turned out to be very time-consuming, with the SVM problem being as hard as inverting a large matrix, in addition to the problem of training multiple SVMs to classify multiple classes. While this will not have a large impact on the computing time, we can use this as an excuse to take a look at dimensionality reduction. For that, we will use PCA and try to find an optimal component number to reduce to.

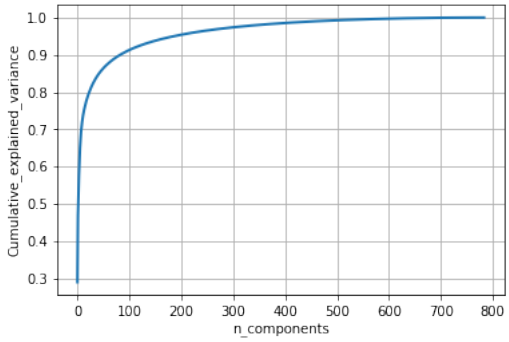

Fig. 3. Results of PCA on our dataset

We can see that taking about 100 most important components amounts to over 90% of the overall variance, therefore we will go with $n = 100$.
Still, we were looking at runtime of hours per parameter set, therefore we were

able to check only a small number of different parameters. However, we have achieved good results with polynomial and Gaussian kernels with regularization factor of 100 - both around 89%. It is worth nothing that SVM performs very well on the original MNIST, with the best tuned parameters giving over 97% accuracy.

4 Decision trees and random forests

Decision trees by themselves were prone to overfitting, with accuracy falling off very quickly as maximum depth was increased over 10. However, the random forests increased the accuracy a lot - no matter the maximum depth, the individual trees produced roughly uncorrelated errors, and the overall forest behaved very neat, achieving results of around 87%.

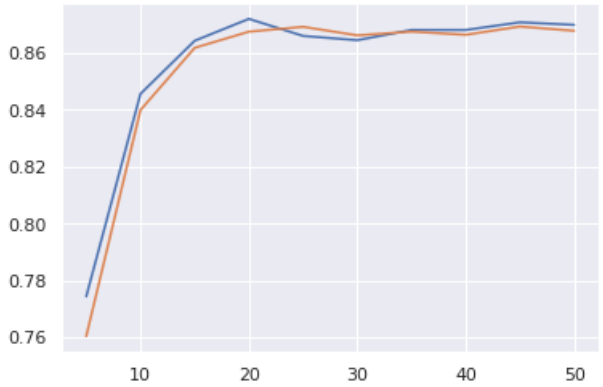

Fig. 4. Accuracy of random forests (Gini coefficient and entropy-based). X-axis: maximum depth of single tree.

5 Naive Bayes

Python's Scikit-learn offers different models based on Bayesian inference - we will use two of them, MultinomialNB and GaussianNB. The multinomial Naive Bayes model turned out to be of no use - it naively treats continuous data as discrete, which resulted in erratic behaviour and giving the same results regardless of smoothing constant and other parameters. GaussianNB model behaved better, but it still gave the worst results of all used models - this was most likely a result of the data not fitting the model assumptions - the model tried to fit a Gaussian distribution to the pixel values, which clearly was not the case.

6 Convolutional neural network

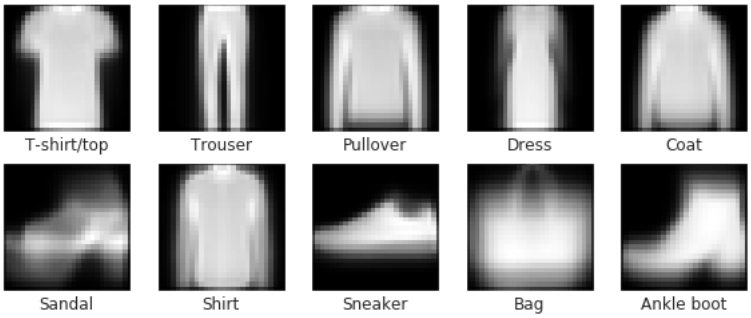

Fig. 5. Average pixel values of each class

All of the models we have used have one common weakness - they are naive, in the sense that they only consider values of individual points, not taking into account the state of neighbouring pixels. When images of different classes are similar in shape (like in the figure above), this can lead to decrease in accuracy. To demonstrate a different approach, in the last example we will use a convolutional neural network. However, we will treat this network as a black box and simply follow an online tutorial, and we only include it in the project to show how good results a "smarter" model can achieve.
Our CNN consists of two sequential layers, each in turn consisting of:

- Convolution layer with kernel of size 3*3
- Batch normalization layer
- ReLU activation function
- Max Pooling layer of size 2*2

After 3000 iterations, the results are very good - we have achieved accuracy of 89%, with some online CNN models getting results as good as 93%. Again, when used on the original MNIST dataset, neural networks are known to reach error rate below 1 percent, which again shows that our dataset is considerably harder to classify.

7 Summary of results

The table below presents best achieved results for each method on Fashion MNIST and original MNIST, respectively (the most optimal parameters of methods can vary between the two sets).

Method	Fashion MNIST	Original MNIST
kNN	0.867	0.959
Linear SVM	0.833	0.925
Poly SVM	0.85	0.97
Gaussian SVM	0.868	0.965
Decision tree	0.81	0.88
Random forest	0.871	0.951
Naive Bayes	0.6721	0.80

The results clearly show that Fashion MNIST is a lot harder to classify - the images are not resized, the edges are less clear and some classes are very similar. However, this also means that it is a more realistic benchmark for real-life problems, e.g. for image (including clothes, obviously) recognition.