Machine Learning Project - Image Recognition

Francesco Ferretto Riccardo Mazzieri Giuliano Squarcina

Abstract

In this work we explain, discuss and solve an image recognition problem in Machine Learning in a different number of ways. We start with simpler methods like Logistic Regression or k-NN, then made some experiments with decision trees and random forests and finally use some more complex models like Support Vector Machines and Convolutional Neural Networks. We also report our results, and try to explain and interpret them. All the presented models were implemented using `scikit-learn` *and* `keras` *modules for Python.*

1. Introduction

Image Recognition is a classic and very studied problem in Machine Learning, that can be solved in a variety of ways. Specifically, this project tackles a **multi-class image recognition problem**:

Given a training set $\mathcal{T} = \{(\boldsymbol{x}_i, y_i), i = 1, \dots, N\}$, where:

- *N is the number of training samples, each one associated with a class $y_i \in \{1, \dots, c\}$;*
- *c is the number of classes;*

Determine a function h s.t. $h(\boldsymbol{x}_i) \approx y_i$ for all $(\boldsymbol{x}_i, y_i) \in \mathcal{T}$

2. Dataset

The Dataset of choice for this problem was the **Fashion MNIST** dataset, which contains 70'000 Zalando's article images - 60'000 examples for the training set and 10'000 for test set. Specifically, we partitioned the whole training set into 50'000 samples used for actual training and 10000 used as a validation set. For this project the labels for the test set are unknown, and our predictions for them will be submitted as part of the evaluation. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Each class has the same number of images, so the dataset is balanced. We made sure to normalize the data, in such a way that all the pixel values fit in the range $[0, 1]$, by dividing everything by 255.

3. Method

To have a rough idea of the complexity of the task, we started by using the simplest models we studied during the course (Logistic Regression, k-NN) and used those performances as a baseline from where we could try to improve.

3.1. Logistic Regression

The first and simpler model we tested is ***Logistic Regression***. In literature, Logistic Regression model is meant for binary classification: specifically, the output is the probability of the sample being in class 1 instead of class 0. Logistic Regression can be extended to solve multi class classification problems by using the *"**one vs rest**" scheme*, or, with Softmax Regression, via the following *cross-entropy loss function*:

$$L(\boldsymbol{w}) = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{c} y_{i,j} log(p_{i,j})$$

Where:

- M is the number of samples;
- $y_{i,j} = 1$ if class label j is a correct classification for the i-th sample. Equal to 0 otherwise;
- $p_{i,j} = \frac{exp(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle)}{\sum_{k=1}^{c} exp(\langle \boldsymbol{w}_k, \boldsymbol{x} \rangle)}$ is the predicted probability that the i-th sample is of class j.

Notice that when $c = 2$ then the loss function is equivalent to the logistic regression's one (for the case "*one-vs-rest*").

3.2. k-Nearest Neighbors

k-Nearest Neighbors is another very simple model, where each sample is classified based on the class of the closest points in $\mathcal{T}$ to it. This learning algorithms is a *non-parametric* one, since it doesn't restrict the functional space to a specific class function $\mathcal{H}_{\boldsymbol{w}}$, i.e. without learning specific weights parameters.

One of the main problems of this method in our case is that the images have a lot of features (784 `pixels`) and we know that points in high dimensional spaces tend to be equally far from each other (this is often referred to as the "*curse of dimensionality*"), thus hindering the main underlying assumption that close points should belong to the same class.

To reduce the dimensionality of the data, we performed Principal Component Analysis before fitting the k-NN model, and then compared the result to see how the results in terms of CV accuracy changed.

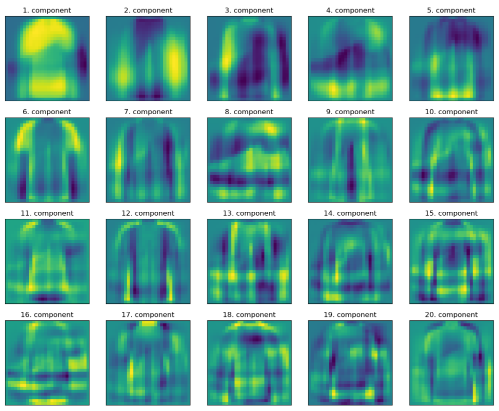

Figure 1. We tried also to plot the first 20 PCs extracted from the original features, to try to give an interpretation. Unfortunately, even if they help reducing a lot the number of features, they still do not look very interpretable.

3.3. Random Forest

Random Forest is a powerful method based on decision trees, used for both binary and multiclass classification. Contrarily to decision trees, RF mitigate a lot the problem of *overfitting* by training multiple decision trees on *random subsets of features at the time of each split and then taking the average of all the tree outputs*, reducing the variances associated with the estimates by paying in terms of biasness and yielding to a model with better generalization capabilities. One of the biggest advantages of Random Forests is the *interpretability* of the results but in our case the features of our data are just the pixel intensity values, which cannot yield significant and intuitive interpretations.

3.4. Support Vector Machines

The ***Support Vector Machine*** (*SVM*) model is a widely used method mainly used for binary classification, but also extendible to multiclass and even regression. The power of this method is that it can be used also for *non-linearly separable data*, by exploiting the so called *"kernel trick"*: data points are mapped into a higher dimensional space where a separating hyperplane can be found. In our case, we used the *RBF (Radial Basis Function)* kernel.

3.5. Convolutional Neural Network

Convolutional Neural Networks (*CNNs*) are a class of Neural Networks mainly specialized in image recognition tasks. The key idea that makes CNNs really good for image analysis is that, instead of preprocessing the data to derive features like textures and shapes, they take just the image's raw pixel data as input and "learn" how to extract these features during the training process, thanks to the convolution operation.

4. Experiments

4.1. Logistic Regression

We made *grid-search* (on 20% of training data for computing time reasons) over the following grid:

- `penalty = ('L1', 'L2');`
- `solver = ('liblinear', 'saga');`
- `C = (0.0001, 0.001. 0.01, ..., 1000)`

using statified k-fold cross validation with 5 splits and data shuffle; the best CV accuracy was $84,30\%$. Best Parameterers: `C=0.1, penalty=l2, solver='saga'`. Then, we trained the model with the best parameters on the full training set. The final accuracy on Validation Set was $85,59\%$.

4.2. k-NN

We made grid search on the full training set over the following range of values for $k : (1, 2, ..., 20)$. We did not look for the best value of p in the L_p metrics, because theorical results have proved that the best metric is L_1 (see [1] and [2]). We implemented a statified k-fold with 5 splits and data shuffle; the Best CV accuracy was 85,96%. The best number of neighbors found was $k = 5$. Then, we trained the model with the best parameter on the full training set. Final accuracy on Validation Set was $85,70\%$. We also tested empirically if L_1 was really the best metric, repeating the entire above procedure using L_2 metric. The Final Accuracy on Validation Set $(85,47\%)$, confirms the theoretical results. Finally we computed the CV Accuracy w.r.t. the number of considered Principal Component (PC) in the range [0,100] (keeping $k = 5$ fixed). We found that for number of PCs $>$ 20 the CV Accuracy stayed the same (see Figure 2), meaning that with only 20 features (versus the 784 from the original data) we can almost describe the full complexity of our data. Therefore, we decided to use only the first 20 PCs to train again our k-NN model with $k = 5$. The Final Accuracy on Validation Set: was 84.82%. Even though the performance was slightly worse than before, we thought that this might be an interesting result, given the fact that we reduced the total number of features by a factor of ≈ 40.

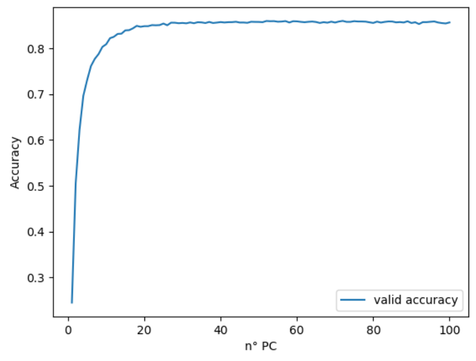

Figure 2.

4.3. Random Forest

Also in this case, we used only the first 20 PCs to train the model. We made grid search on the full training set over the following grid:

- `max_depth = (1, 2, ..., 30);`
- `max_features = (1, 2, ..., 21).`

Where `max_depth` is the maximum depth of the trees inside the forest, and `max_features` is the maximum number of features considered at each split of the forest trees. Again, we also validated each parameter combination with a stratified k-fold cross validation with 5 splits and data shuffle. The Best CV accuracy was $85,60\%$ and the best found parameters were: `max_depth = 24, max_features = 7`. Then, we trained the model with the best parameters on the full training set. The accuracy of this final model on the Validation Set was $85,08\%$.

4.4. Support Vector Machine

Also in this case, to select the optimal parameters for our model we performed a grid search, but for computing time reasons, we performed it on 20% of the training set. We used the following grid:

- `C = (1, 2, 4, 8);`
- `gamma = (0.125, 0.0625, 0.03125);`

Where `C` is the inverse of the regularization parameter, while `gamma` is the parameter inside the RBF kernel (which is the one we used) and defines how much influence a single training example will have. The best found parameters were `C = 4, gamma = 0,03125`, and the best CV accuracy was $87,25\%$. Afterwards, we trained the final model, with those best parameters, on the full training set, achieving an accuracy of $90,39\%$.

4.5. Convolutional Neural Network

For this class of models we adopted a slightly different procedure. We tested different architectures, and diagnosed their performances and generalization capabilities using plots of the training and validation accuracy with respect to the number of epochs. In such a way, we were able to see if the model was overfitting the training data too much. We started by building a CNN with the simplest possible architecture, which means basically just one convolutional layer, one pooling layer and a single, fully-connected, output layer for classification. This model had $90,52\%$ accuracy on the validation set, which is already a really good starting point. However, we noticed that the validation accuracy was still considerably worse than the training accuracy, reason for which we tried to add some regularization to our network by adding a Dropout layer. With this second model, we obtained $90,67\%$ accuracy on the validation set. To find our best model, we experimented with deeper structure and also tried to study architectures of other famous CNNs; we borrowed the main idea behind the architecture from [3] and got a final accuracy of $92,84\%$.

5. Final Reflections

Logistic Regression, despite the relative semplicity has good perfomance (globally speaking). k-NN, even though it relies on strong assumptions, has comparable performance with LR, but it's not preferable wrt to the latter (due to testing time performances). PCA was very effective, allowing k-NN to be applied commercially. SVM and CNN provided the best results, being a range of the dataset cardinality where this methods compete. Anyway, we are aware that alternative configurations of the CNN are surely possible, since in literature better perfomances have being achieved; nonetheless, this is the best result we managed to achieve with our current knowledge.

Here are summed all the best results obtained from all different used approaches.

Method	Accuracy
Logistic Regression	$85,59\%$
k-NN	$85,70\%$
k-NN (with 20 PCs)	$84,82\%$
Random Forest	$85,08\%$
SVM	$90,39\%$
CNN (best)	$92,85\%$

Table 1. Summary of the main results we obtained.

References

[1] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. On the surprising behavior of distance metric in high-dimensional space. *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)*, 02 2002.

[2] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? *ICDT 1999. LNCS*, 1540, 12 1997.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.