École Polytechnique Fédérale de Lausanne

Introduction to Machine Learning CS-233(a)

Milestone 2 Report

Ann-Kristin Bergmann, 362802

Caspar Martin Schön, 362091

Adrian Zvizdenco, 362739

December 20, 2022

1 Principal Component Analysis

The **PCA** technique was used to reduce the dimensionality of the *H36M* dataset from $D = 3300$ features to only $d = 200$, preserving the maximum amount of information in less dimensions and eventually reducing the elapsed time in training all the different models. The **explained variance** of the eigenvectors considered in the **PCA** is 99.43%, which would not increase significantly with higher d as it is pretty high, already. The measured **elapsed time** for the Principal Component reduction itself on the *H36M* dataset is ~ 4.4 seconds.

2 Ridge & Logistic Regression with PCA feature

Model	Time Speedup	Raw Metrics	PCA Metrics
Logistic Regression	41	81.23% acc.; 0.79 F1	83% acc.; 0.80 F1
Linear Regression	147	540 MSE	2.96 MSE
Ridge Regression	126	0.367 MSE	2.92 MSE

Table 1: Re-modelling with PCA technique

The PCA dataset drastically improved the elapsed time for all models. We computed the $speedup = time_{noPCA}/time_{PCA}$, with results presented in the Table 1. We also observed that the accuracies of different **hyper-parameters** λ get closer to each other in the cross-validation plots 1 and 2. For linear regression the **MSE loss** decreased as the data is better parameterized with reduced dimensionality and becomes more compact. In the case of ridge regression the PCA doesn't improve the results, because λ-**regularization** tends to improve the **numerical stability** of the training, which is also improved by **reducing dimensionality** with PCA. With PCA, the **MSE loss** of the ridge regression increases compared to previous results by ~ 2.55, meaning that reduced dimensionality allows less flexibility in the regression task.

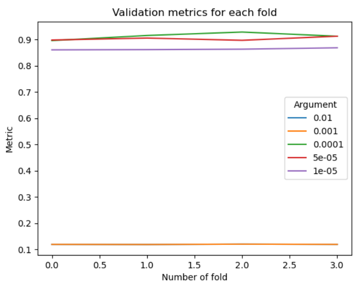

Figure 1: Logistic PCA

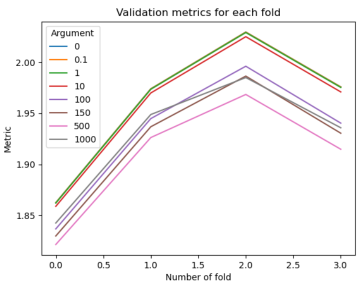

Figure 2: Ridge PCA

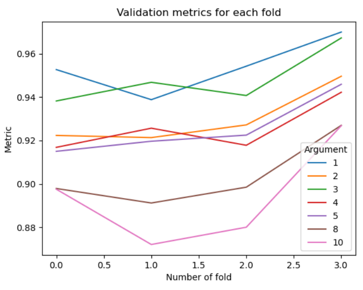

Figure 3: KNN PCA

3 K-Nearest Neighbors

In order to find a suitable value for **hyper-parameter** k, we performed **4-fold cross-validation** over the list {1,2,3,5,6,7,8,10,20} and observed the classification accuracy over all folds. We found $k = 1$ as the **hyper-parameter** with the **highest average accuracy** over the 4 folds as observed in the Figure 5, meaning that the label of the **closest neighbor** is assigned as the prediction of the test sample.

Model	Time Speedup	Raw Metrics	PCA Metrics
kNN	20	89.8% acc.; 0.79 F1	89.8% acc.; 0.79 F1

With PCA we found the same **hyper-parameter** $k = 1$ and exactly same **macro F1-score** and classification **accuracy**, hence due to the good representation (explained variance) of the data via PCA there was no decline in accuracy. Reducing the dimensions, the dataset becomes more compact (compressed), which affects the cross entropy loss function, improving the model's behaviour. As distances are smaller in a lower dimensional space, the optimizer is better at minimizing the loss and finding the better suitable neighbor.

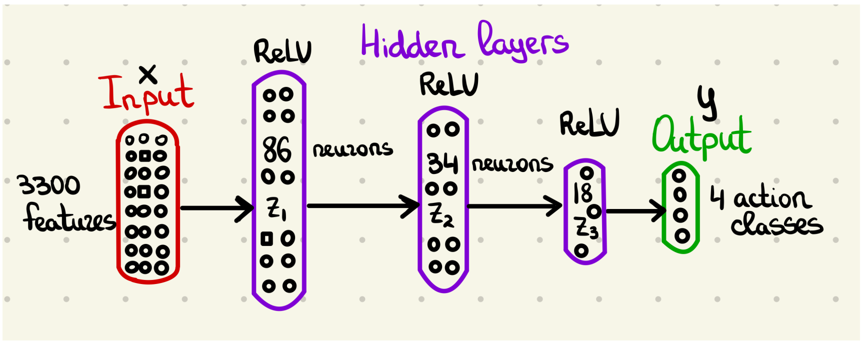

Figure 4: Neural Network Architecture

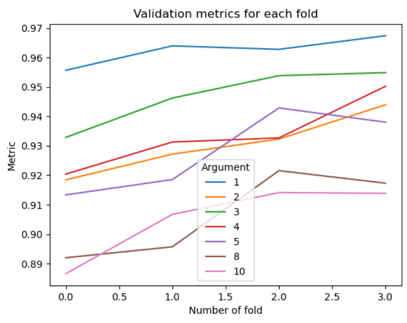

Figure 5: KNN w/o PCA

4 Neural Network

With the default layout using a hidden layer of 32 neurons, we reached 76.04% classification accuracy. We added 2 more hidden layers and tweaked the parameters, to find that with 3 **layers** of $86, 34$ and 10 **neurons** respectively, a classification **accuracy** of 91.6% and **macro F1-score** 0.90 can be achieved. Choosing too many hidden layers and too many dimensions causes **over-fitting**, due to the higher degree-of-freedom and learning the data instead of making good predictions for the test data. We limited the maximum epochs to ~ 250 and learning rate $1e-3$ to stop the learning earlier and prevent over-fitting as well. Besides that, we changed the number of iterations after which the **stochastic gradient descent** reduces the **learning rate**, making it **adaptive** every 50 epochs in the algorithm, instead of 100 previously.
Ultimately, to further improve the Neural Network, we added a couple of $1D$ **convolutional** layers (`kernel` $= 5$, $1 \rightarrow 6 \rightarrow 24$ channels) with max-pooling (`kernel`=2), followed by a **dropout** layer (p $= 0.6$) and optimized the model with **ADAM** to get a classification accuracy of 94.5% and macro F1-score 0.94.

Parameters	# Epochs	Hidden Layers	Initial LR	Optimizer	Metrics
Default	1000	1 (32)	1e-4	SGD	0.74 F1-score, 76.04% acc.
Trial	300	3 (84, 32, 10)	1e-4	SGD	0.88 F1-score, 90.2% acc.
Trial	250	3 (86, 34, 18)	1e-3	SGD	0.90 F1-score, 91.6% acc.
Best Setup	50	3 (120, 68, 36)	1e-4	ADAM	0.94 F1-score, 94.5% acc.

Table 2: Neural Network Parameters

Some of the console outputs we obtained are screenshot and saved in the **output** folder of the project