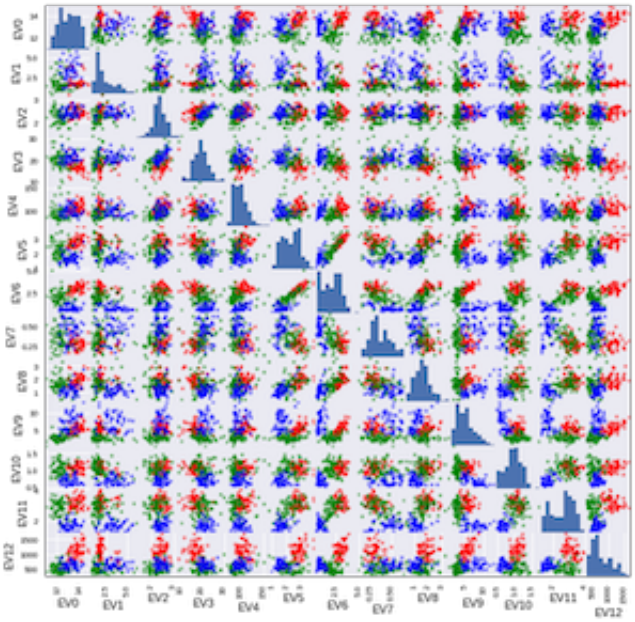
MACHINE LEARNING (T-MACHLE)

Practical work 12 - Dimensionality reduction

STUDENT: ***ROMAIN CLARET***
PROFESSOR: ***A. PEREZ-URIBE & J. HENNEBERT***
ASSISTANT: ***H. SATIZABAL***
DUE-DATE: ***MONDAY 17 DECEMBER 2018***

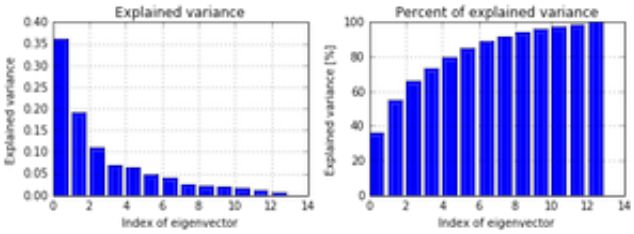
PCA

Provide the scatter matrix and select the pair of variables (by visual inspection of the scatter matrix) that appears to allow the recognition of the three classes of wine. Explain.



- Based on my observations I cannot choose a specific pair that allows a better recognition than another.
- For me: **EV6** and **EV12** *lines* are good in general

Find the smallest set of components capable of explaining at least 50% of the variance of the data.



- [837.64134503 444.46132455]
- Keeping 55.406338356935315 % of the variance

What is the percentage of the variance of the data explained by each one of the first 3 principal components ?

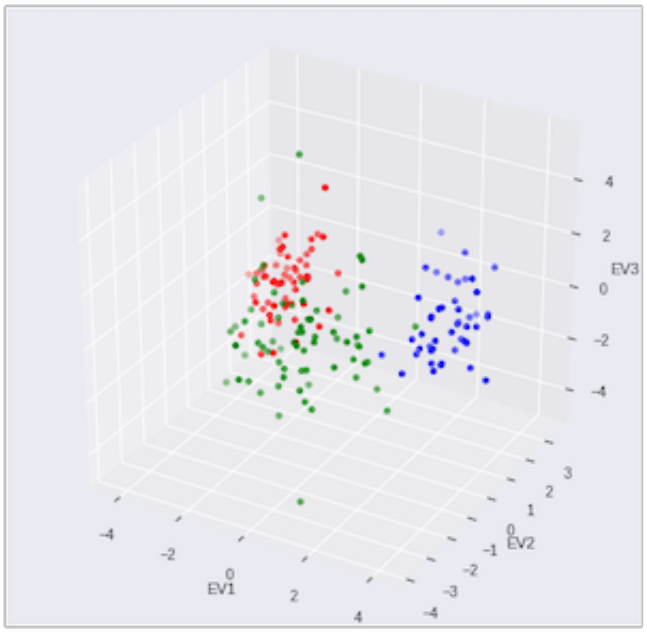
```
percent = pca.s/sum(pca.s)*100
print(percent[:3]) #or print(pca.fracs[:3]*100)
```

- [36.1988481 19.20749026 11.12363054]

Find the smallest set of components capable of explaining at least 60% of the variance of the data. How do you print the resulting eigenvectors ? print them and if only 3 components are required, provide the 3D projection of the original dataset.

```
print min_index + 1, 'eigenvectors are needed'
print(pca.s[:min_index + 1])
print 'Keeping', cummulated_variance[min_index]*100, '% of the
variance'
```

- 3 eigenvectors are needed
- [837.64134503 444.46132455 257.40081061]
- Keeping 66.52996889318527 % of the variance



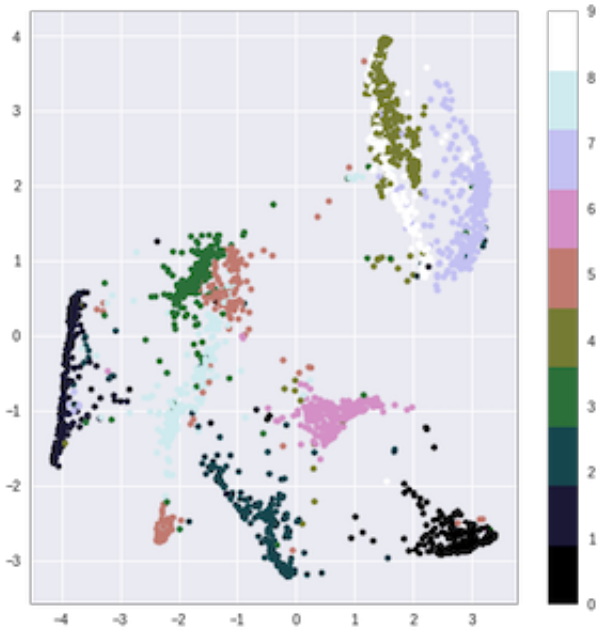
Find the smallest set of components capable of explaining at least 80% of the variance of the data.

- 5 eigenvectors are needed
- [837.64134503 444.46132455 257.40081061 163.57735843 151.87461575]
- Keeping 80.1622927555479 % of the variance

T-SNE

Run the notebook and observe the resulting 2D visualization of the dataset. Are there ten classes clearly separated ? provide that visualization.

- I couldn't make the visualization work on Colab out of the box. So I used a custom colorbar from jakevdp/discrete_cmap.py



- They are not separated clearly.
- [0, 1, 2, 6] are distinguishable
- As to be expected, the following triplets are not well distinguishable. Indeed they are similar visually. [3, 5, 8][4, 7, 9]

What is the dimensionality of the input data ?

```
print(X.shape)
```

- (2500, 784)
- 2500: components
- 784: flattened 28x28 picture

what is the final dimensionality at the output of t-SNE ?

```
print(X.shape)
```

- (2500, 2)
- 2500: components

- 2: the chosen amount of out dimension from t-SNE

What is the dimensionality of the input data being fed to t-SNE ?

- X: (2500, 784)
- pca: (2500, 50)
- x2p: (2500, 2500)

Identify the values of the parameters having been used to obtain those results: perplexity, learning rate, momentum, number of iterations.

- perplexity: 40
- learning rate: 0.01 (min_gain)
- initial_momentum: 0.5(iter < 20)
- final_momentum: 0.8 (iter >= 20)
- number of iterations: 50 (max_iter)

What is the formula being used to compute the error given every 10 iterations ?

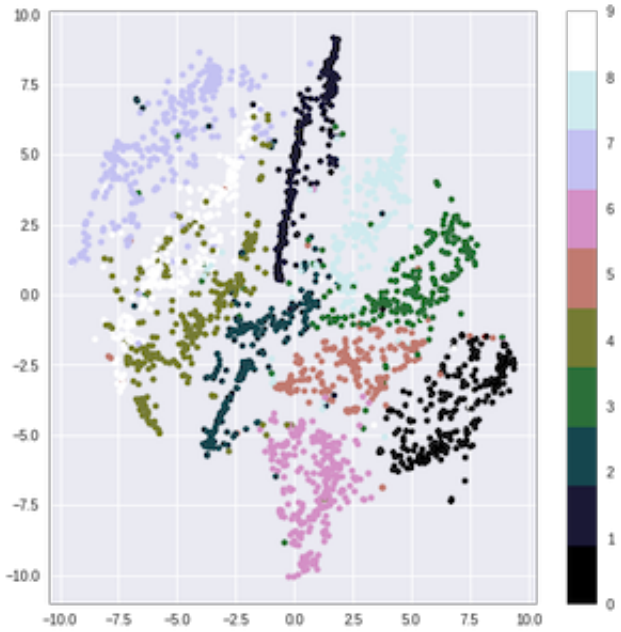
```
C = Math.sum(P * Math.log(P / Q));
```

- Kullback–Leibler divergence

What happens if you feed the original datapoint to the t-SNE algorithm directly ?

- The PCA would not work. Which would result in much noise.

Are you happy with the 2D visualization of the ten classes of datapoint ? if not, modify the parameters to get a better one. Provide the resulting visualization and then explain how did you get to it.



- no_dims: 40
- perplexity: 40
- learning rate: 0.001 (min_gain)
- initial_momentum: 0.6 (iter < 20)
- final_momentum: 0.9 (iter >= 20)
- number of iterations: 120 (max_iter)
- Reasons:
  - My goal was to split triplets
  - I noticed that starting at 110 iterations: the error is cut a lot
  - The greater was the gap between dim and perplexity the better is the error result.
  - But too extreme values are not providing nice clusters even if the error is low.
  - But finally, the same value for the perplexity and no_dims got the best results for me.
  - Momentum gaps were resulting in a higher mixing of points
  - It was many trials…

AUTOENCODERS

Provide the notebook you setup for training a convolutional autoencoder and for denoising the digits of the MNIST database.

Suggested topology

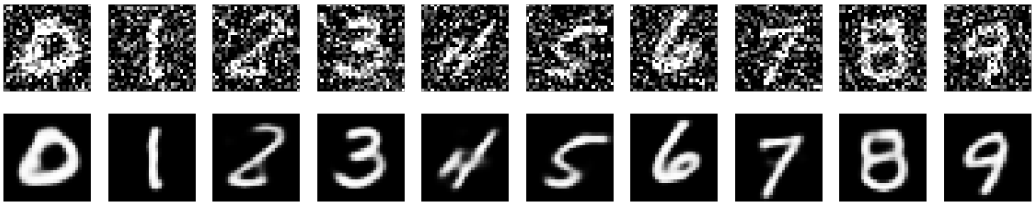
LAYER (TYPE)	OUTPUT SHAPE	PARAM #
input_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_1 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_1 (MaxPooling)	(None, 14, 14, 16)	0
conv2d_2 (Conv2D)	(None, 14, 14, 16)	2320
max_pooling2d_2 (MaxPooling)	(None, 7, 7, 16)	0
conv2d_3 (Conv2D)	(None, 7, 7, 16)	2320
up_sampling2d_1 (UpSampling)	(None, 14, 14, 16)	0
conv2d_4 (Conv2D)	(None, 14, 14, 16)	2320
up_sampling2d_2 (UpSampling)	(None, 28, 28, 16)	0
conv2d_5 (Conv2D)	(None, 28, 28, 1)	145

Code

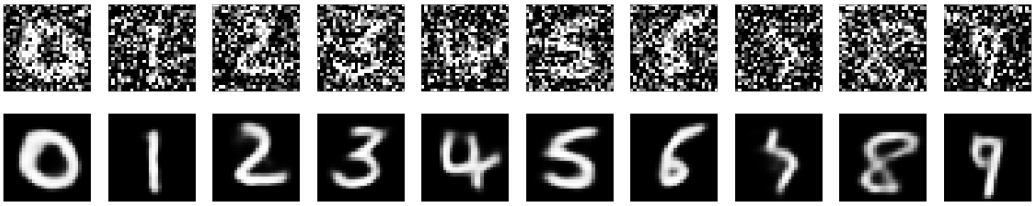
```
l1 = Conv2D(16, (28, 28), padding='same', activation='relu',
name='l1')(input_img)
l1_mp = MaxPooling2D((2, 2), name='l1_mp')(l1)

l2 = Conv2D(16, (14, 14), padding='same', activation='relu',
name='l2')(l1_mp)
l2_mp = MaxPooling2D((2, 2), name='l2_mp')(l2)

l3 = Conv2D(16, (7, 7), padding='same', activation='relu',
name='l3')(l2_mp)
l3_us = UpSampling2D((2, 2), name='l3_us')(l3)

l4 = Conv2D(16, (14, 14), padding='same', activation='relu',
name='l4')(l3_us)
l4_us = UpSampling2D((2, 2), name='l4_us')(l4)

decoded = Conv2D(1, (28, 28), activation='sigmoid', pad-
ding='same')(l4_us)
```

Provide results showing some examples of the ten digits before and after denoising for the three different levels of noise: noise factor = 0.5, 0.7, and 0.9

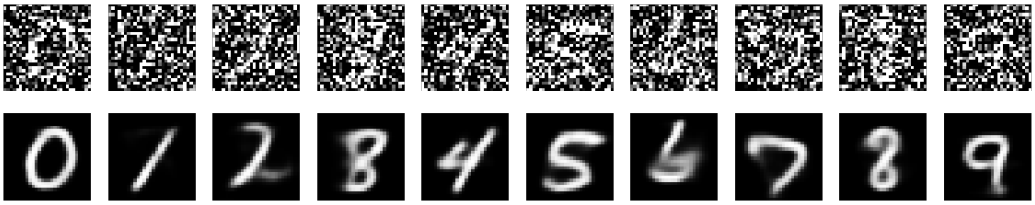
noise factor = 0.5, epochs=10



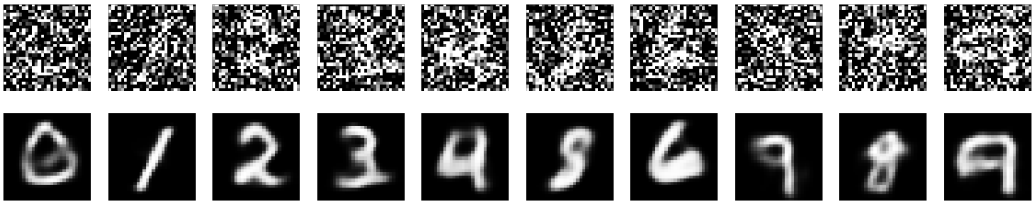
noise factor = 0.7, epochs=10



noise factor = 0.9, epochs=10

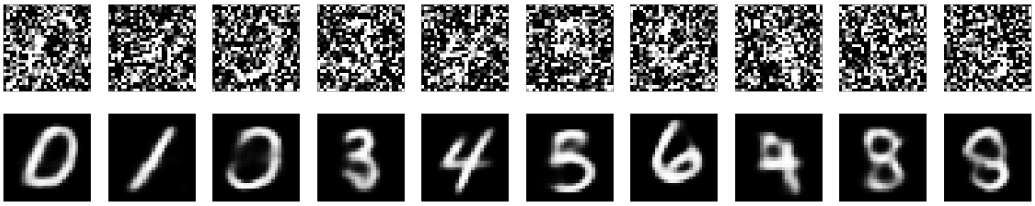


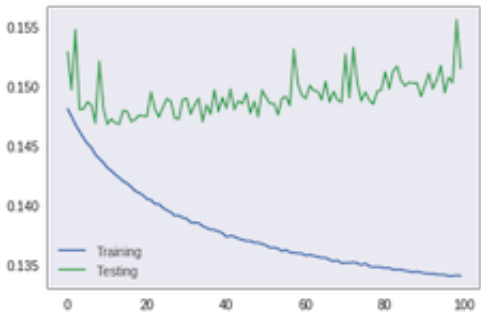
noise factor = 0.95, epochs=10



noise factor = 0.99, epochs=10

noise factor = 0.99, epochs=100





- Just for science, I tried until 0.99 for the noise factor. And the result is decent, compared to my input interpretation. In the case of the 100 epochs, I think it's a good case of overfitting.

Conclude based on the obtained results

- The higher is the noise factor, the more difficult it is for me, human, to find out the input. However, the autoencoder is doing amazingly well at filtering the noise out.