	Travail fait par EL JANOUSSI Aya, EL KHOURI Tonia et SGHYAR Fatima Az-Zahrae 1. The datasets: MNIST 1.1. Description The MNIST database (Modified National Institute of Standards and Technology database) is a very popular Machine Learning dataset comosed of large collection of handwritten digits. It has 70 000 black-and-white 28x28 pixels images with a handwritten digit ('0' to '9'). The goal is to classify the black-and-white 28x28 pixels images among these 10 classes. 1.2. Load the dataset
In [1]:	<pre>import tensorflow as tf import numpy as np import pandas as pd import matplotlib.pyplot as plt X = np.load('MNIST_X_28x28.npy') y = np.load('MNIST_y.npy') 1.4. Works / Questions Q1/ What are the shape of the data? Display samples from the dataset</pre>
<pre>In [2]: Out[2]: In [3]: Out[3]: In [4]: Out[4]:</pre>	np.shape(X) (70000, 28, 28) np.shape(y) (70000,) np.size(y)
<pre>In [5]: Out[5]: In [6]:</pre>	plt.imshow(X[210]) <pre> <matplotlib.image.axesimage 0x1b8410b1310="" at=""></matplotlib.image.axesimage></pre>
In [7]:	fig, sxes = plt.subplots(3,5, figsise=(10,10))
<pre>In [8]: In [9]: Out[9]: In [10]:</pre>	<pre>irom sklearn.model_selection import train_test_split (X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size = 0.2) np.shape(X_train) (56000, 28, 28)</pre>
Out[10]: In [11]: Out[11]: In [12]: Out[12]:	(56000,) np.shape(X_test) (14000, 28, 28) np.shape(y_test)
In [13]: In [14]:	Q3/ Are the train and test sets well balanced (distribution of labels)? Why is it important for supervised Machine Learning? from collections import Counter class_frequencies = Counter (y_train) print (class_frequencies) Counter({1: 6287, 7: 5813, 3: 5722, 2: 5622, 0: 5586, 9: 5547, 6: 5499, 4: 5447, 8: 5418, 5: 5059})
	plt.title("Labels distribution in Train Set") plt.show(fig) Labels distribution in Train Set 6000 4000 5000 4000 1000
In [15]:	plt.xlabel("Count") plt.title("Labels distribution in Test Set") plt.show(fig) Labels distribution in Test Set 1600 1200 1000 1000 1000 1000 1000 100
	We notice that our train and test sets are well balanced indeed. On average we got approximately 5000 pictures of each class for the train set. And approximately 1350 pictures for each class for the test set. It is important to have well balanced train and test sets to get more accurate results and prevent biases! 2. Unsupervised Machine Learning 2.1. Dimensionality reduction Works / Questions
<pre>In [16]: In [17]: In [18]: Out[18]:</pre>	<pre>nsamples, nx, ny = X_train.shape d2_X_train = X_train.reshape((nsamples,nx*ny)) d2_X_train.shape (56000, 784)</pre>
In [19]:	<pre>pca = FcA(n_components=704) pca.fit(d2_X_train) plt.plot(pca.explained_variance_ratio_) plt.xlabel('Number of components') plt.ylabel('Explained variance ratio') plt.title('Case : n_components = 784') plt.show()</pre> Case:n_components = 784
In [20]:	n_components = 80 pca = PCA(n_components=80) pca.fit(d2_X_train) plt.plot(pca.explained_variance_ratio_)
	plt.xlabel('Number of components') plt.ylabel('Explained variance ratio') plt.title('Case : n_components = 80') plt.show() Case:n_components = 80 0.10 0.08 0.00 0.
In [21]:	n_components = 220 pca = PCA(n_components=220) pca.fit(d2_X_train) plt.plot(pca.explained_variance_ratio_) plt.xlabel('Number of components') plt.ylabel('Explained variance ratio') plt.title('Case : n_components = 220') plt.show() Case : n_components = 220
	0.10 0.08 0.00
In [22]:	pca.fit(d2_X_train) print(pca.explained_variance_ratio_) [9.78607007e-02 7.11299751e-02 6.11378609e-02 5.41981991e-02 4.89496870e-02 4.31231201e-02 3.27951510e-02 2.90182057e-02 2.76258219e-02 2.34585115e-02 2.10717984e-02 2.04504830e-02 1.71078101e-02 1.68594016e-02 1.58299768e-02 1.48260997e-02 1.31903165e-02 1.28183569e-02 1.18559912e-02 1.14479059e-02 1.06499355e-02 1.00639049e-02 9.62296362e-03 9.13030891e-03 8.86813806e-03 8.39141963e-03 8.09569577e-03 7.82375632e-03 7.41106888e-03 6.87086798e-03 6.57261235e-03 6.46733510e-03 6.03135688e-03 5.89644695e-03 5.65226372e-03 5.43846166e-03 5.03783987e-03 4.88290486e-03 4.78461280e-03 4.69136079e-03
	4.88290486e-03 4.4892512e-03 4.78461280e-03 4.69138079e-03 3.83828877e-03 3.75857479e-03 3.60180744e-03 3.49855619e-03 3.38283193e-03 3.19590800e-03 3.15901657e-03 3.10544990e-03 2.97461599e-03 2.88044549e-03 2.82623573e-03 2.69479392e-03 2.67369393e-03 2.56823549e-03 2.52467808e-03 2.20929933e-03 2.38932937e-03 2.38023518e-03 2.28727408e-03 2.20929933e-03 2.12083662e-03 2.06483648e-03 2.03079132e-03 1.95997955e-03 1.91411999e-03 1.87280453e-03 1.86299035e-03 1.80499560e-03 1.76921453e-03 1.73707047e-03 1.64906379e-03 1.63621276e-03 1.61540519e-03 1.53348446e-03 1.47643045e-03 1.42582766e-03 1.41476386e-03 1.40334528e-03 1.3964736e-03 1.34493647e-03 1.32257211e-03 1.31582548e-03 1.29886430e-03 1.25817893e-03 1.22827258e-03 1.19484972e-03 1.17021104e-03 1.14331141e-03 1.11859845e-03 1.09540747e-03 1.07634518e-03 1.07193154e-03 1.03925945e-03 1.03124284e-03 1.00422206e-03 9.95519537e-04 9.76746589e-04 9.42057996e-04 9.38114088e-04 9.11803106e-04 8.96136494e-04 8.90288543e-04 8.63872149e-04 8.47554285e-04 8.37035080e-04 8.18236118e-04 7.86368012e-04 7.79117863e-04
	8.37035080e-04 8.18236118e-04 7.85368012e-04 7.79117863e-04 7.75718310e-04 7.63165255e-04 7.59824228e-04 7.49314892e-04 7.33069031e-04 7.24347487e-04 7.15581159e-04 7.04966704e-04 6.91584931e-04 6.87077805e-04 6.78622440e-04 6.67748405e-04 6.57775896e-04 6.43014843e-04 6.31592502e-04 6.29663294e-04 6.20439460e-04 6.01349840e-04 5.98920543e-04 5.95405862e-04 5.85135768e-04 5.82670495e-04 5.78989388e-04 5.72696309e-04 5.61561599e-04 5.50458764e-04 5.34566471e-04 5.23441498e-04 5.20895701e-04 5.09464044e-04 5.00663764e-04 4.97942559e-04 4.94434042e-04 4.92067097e-04 4.82009779e-04 4.80708337e-04 4.72099315e-04 4.68811847e-04 4.64106261e-04 4.60629586e-04 4.38086422e-04 4.24328529e-04 4.20426384e-04 4.18371370e-04 4.38086422e-04 4.07688826e-04 4.00684552e-04 3.93160662e-04 3.90831564e-04 3.88838207e-04 3.83616511e-04 3.79721585e-04 3.77171652e-04 3.5617540e-04 3.53144797e-04 3.51865241e-04 3.45865309e-04 3.45294742e-04 3.41600093e-04 3.38835574e-04 3.45865309e-04 3.45294742e-04 3.41600093e-04 3.23081859e-04 3.345294742e-04 3.42987710e-04 3.23081859e-04 3.34902170e-04 3.28909857e-04 3.27987710e-04 3.23081859e-04
	3.24902170e-04 3.28909857e-04 3.27987710e-04 3.23081859e-04 3.11106127e-04 3.08981756e-04 3.05480043e-04 3.03926638e-04 3.02205017e-04 2.98949916e-04 2.95341705e-04 2.93765188e-04 2.92113417e-04 2.90661596e-04 2.87124716e-04 2.83413783e-04 2.80419567e-04 2.77232771e-04 2.3148563e-04 2.70626834e-04 2.68359935e-04 2.65496216e-04 2.65233979e-04 2.64261728e-04 2.68359935e-04 2.58619754e-04 2.56982886e-04 2.56684625e-04 2.55509336e-04 2.51975428e-04 2.56982886e-04 2.56684625e-04 2.46537650e-04 2.44410212e-04 2.41904436e-04 2.41778541e-04 2.39973482e-04 2.38442059e-04 2.38201548e-04 2.35887569e-04 2.25622101e-04 2.23779376e-04 2.29210613e-04 2.27207376e-04 2.25622101e-04 2.23779376e-04 2.20738462e-04 2.18696355e-04 2.12032082e-04 2.16104461e-04 2.14153686e-04 2.18249388e-04 2.05453927e-04 2.02310018e-04 2.09418205e-04 2.00948252e-04 1.98658179e-04 1.97150554e-04 1.96021933e-04 1.88557200e-04 1.87090798e-04 1.86586269e-04 1.88463901e-04 1.88436164e-04
	1.83083680e-04 1.81787807e-04 1.80731915e-04 1.78958007e-04 1.77110006e-04 1.76802799e-04 1.74634233e-04 1.73282136e-04 1.72311108e-04 1.71480272e-04 1.70645003e-04 1.70301434e-04 1.67970982e-04 1.6793346e-04 1.66094717e-04 1.65812227e-04 1.63344843e-04 1.62814938e-04 1.61646870e-04 1.61068750e-04 1.59840735e-04 1.58900134e-04 1.58245531e-04 1.57239483e-04 1.55641484e-04 1.53924957e-04 1.58245531e-04 1.52713055e-04 1.50971202e-04 1.49535167e-04 1.48670952e-04 1.48210619e-04 1.47228449e-04 1.45677627e-04 1.44277035e-04 1.4292944e-04 1.42750020e-04 1.41430061e-04 1.40806759e-04 1.39940367e-04 1.37619685e-04 1.37273773e-04 1.36637858e-04 1.35116127e-04 1.34891132e-04 1.33964784e-04 1.3359312e-04 1.31459088e-04 1.30937892e-04 1.28734265e-04 1.28222351e-04 1.28047366e-04 1.26989641e-04 1.26283503e-04 1.26016915e-04 1.25645760e-04 1.24353211e-04 1.23277338e-04 1.28119751e-04 1.21623855e-04 1.20572911e-04 1.19261131e-04 1.18545252e-04 1.18102191e-04 1.17498681e-04 1.16242009e-04 1.15898648e-04 1.1510259e-04 1.11835464e-04 1.1510259e-04 1.11835464e-04 1.1510259e-04 1.11835464e-04 1.151026207e-04
	1.10166064e-04 1.09802974e-04 1.09155756e-04 1.08642918e-04 1.08303212e-04 1.07887824e-04 1.06296592e-04 1.06086776e-04 1.05684095e-04 1.04501668e-04 1.03683418e-04 1.02003225e-04 1.01356269e-04 1.00173113e-04 9.98019415e-05 9.94036971e-05 9.85407331e-05 9.80219165e-05 9.73641226e-05 9.72891567e-05 9.52596126e-05 9.40111178e-05 9.34285648e-05 9.30068322e-05 9.23130990e-05 9.11787432e-05 9.05948424e-05 9.02103370e-05 8.94092500e-05 8.86256511e-05 8.82515638e-05 8.73409361e-05 8.68492332e-05 8.62843298e-05 8.55573623e-05 8.49122309e-05 8.39546395e-05 8.34410294e-05 8.25893200e-05 8.21980825e-05 8.18583353e-05 8.10486480e-05 8.06840410e-05 7.93739285e-05 7.90673310e-05 7.81651907e-05 7.73583570e-05 7.67193354e-05 7.56925854e-05 7.56322885e-05 7.43560588e-05 7.37456062e-05 7.30893834e-05 7.29249784e-05 7.21320000e-05 7.18588666e-05 7.10599939e-05 6.99827832e-05 6.94304380e-05 6.88518510e-05 6.79042413e-05 6.75018655e-05 6.64670254e-05 6.58469398e-05 6.46975953e-05 6.35639628e-05 6.31882643e-05 6.29601274e-05 6.16351895e-05 6.08625117e-05 5.97047371e-05 5.92048540e-05
	5.88247835e-05 5.82312485e-05 5.79970591e-05 5.68243523e-05 5.65161784e-05 5.57894908e-05 5.53683744e-05 5.45105825e-05 5.44361434e-05 5.38797532e-05 5.27804324e-05 5.24317154e-05 5.22078734e-05 5.13375153e-05 5.10732335e-05 5.06013562e-05 4.91506938e-05 4.90453039e-05 4.85317124e-05 4.80964290e-05 4.72288977e-05 4.67782438e-05 4.63245730e-05 4.57936865e-05 4.48333273e-05 4.45519228e-05 4.31415490e-05 4.23397838e-05 4.20913725e-05 4.10479524e-05 4.06453294e-05 4.05705015e-05 4.01054488e-05 3.98869915e-05 3.87508713e-05 3.81516790e-05 3.64021519e-05 3.77774691e-05 3.71033360e-05 3.47724412e-05 3.42388241e-05 3.49578005e-05 3.48906877e-05 3.27102671e-05 3.2463706e-05 3.29959135e-05 3.18154356e-05 3.13821137e-05 3.10523610e-05 2.97209641e-05 2.97478577e-05 2.89035033e-05 2.85509051e-05 2.8391039e-05 2.79438387e-05 2.76946744e-05 2.75664662e-05 2.72719823e-05 2.66050693e
	2.43222435e-05
	7.47441432e-06 7.44384308e-06 7.28164377e-06 7.24036365e-06 7.01712492e-06 6.87231624e-06 6.68138297e-06 6.65680423e-06 6.33237431e-06 6.22545526e-06 5.98734764e-06 5.76208260e-06 5.61119879e-06 5.51874134e-06 5.39811744e-06 5.37375292e-06 5.34230879e-06 5.16413775e-06 4.80075938e-06 4.71660802e-06 4.69100420e-06 4.63760776e-06 4.63381465e-06 4.55162770e-06 4.53469726e-06 4.48540952e-06 4.27569218e-06 4.17672808e-06 3.93623095e-06 3.92233314e-06 3.88595467e-06 3.71684674e-06 3.48326278e-06 3.32677724e-06 3.32393536e-06 3.28312195e-06 3.24960487e-06 3.21099345e-06 3.15006494e-06 3.08226370e-06 2.97763687e-06 2.95724125e-06 2.75606379e-06 2.66480141e-06 2.61797585e-06 2.45092288e-06 2.43922915e-06 2.29900186e-06 2.24143716e-06 2.03283578e-06 2.16182401e-06 2.15975946e-06 2.13581030e-06 2.08283578e-06 2.06961623e-06 1.90971521e-06 1.90072033e-06 1.81541363e-06 1.75710261e-06 1.73036717e-06 1.69783497e-06 1.53221644e-06 1.45022581e-06 1.41440588e-06 1.37775729e-06 1.25410945e-06 1.24441149e-06 1.22222144e-06 1.137775729e-06 1.25410945e-06 1.2242141e-06 1.22222144e-06
	1.17342344e-06 1.13278019e-06 1.12735970e-06 1.12050132e-06 1.11054695e-06 1.02462239e-06 1.01677430e-06 1.00168265e-06 9.68192458e-07 9.43568962e-07 8.63823320e-07 8.47670688e-07 8.09000646e-07 8.07361430e-07 7.31528960e-07 7.08286105e-07 7.05210569e-07 7.02928991e-07 6.37913857e-07 6.25792299e-07 5.88588757e-07 5.43634396e-07 5.41779604e-07 5.03763430e-07 5.01244669e-07 4.85115160e-07 4.58401692e-07 4.46589654e-07 4.44930215e-07 4.33668495e-07 3.90630652e-07 3.90250576e-07 3.70278954e-07 3.62065630e-07 3.27855520e-07 3.22308334e-07 3.10804178e-07 3.05057489e-07 2.79725143e-07 2.67699771e-07 2.62025608e-07 2.44944598e-07 2.23720023e-07 2.11866573e-07 2.10997638e-07 2.04720012e-07 1.86891376e-07 1.84946887e-07 1.75634597e-07 1.75204111e-07 1.71106744e-07 1.65785385e-07 1.29821180e-07 1.00647490e-07 9.56544662e-08 8.86864160e-08 8.78909980e-08 6.56824110e-08 5.80921343e-08 3.49977466e-08 3.47003296e-08 3.22193163e-08 3.15321270e-08 2.93581487e-08
	2.84977490e-08 2.57249319e-08 2.14035365e-08 1.78014677e-08 1.45524698e-08 1.33259627e-08 1.25514430e-08 1.15385663e-08 1.10320615e-08 9.44583228e-09 9.23267025e-09 7.40043724e-09 6.6836623ae-09 5.55422622e-09 5.31952566e-09 4.91745374e-09 3.87154658e-09 3.45785135e-09 3.14931119e-09 2.01813092e-09 9.74068563e-10 7.48700012e-10 2.70667318e-10 2.30783652e-10 1.27380571e-10 1.93093177e-32 3.50024334e-33 2.78318195e-33 2.51582509e-33 1.83657451e-33 1.79671628e-33 1.56949102e-33 1.16868592e-33 1.14136303e-33 7.37391561e-34 6.77463566e-34 5.60999342e-34 3.78064865e-34 3.59935733e-34
In [23]:	3.59935733e-34 3.59935739e-04 4.18692247e-04 4.18692247e-04 4.18692247e-04 4.1869
	3.1010007e+04 3.06190219e+04 3.03092773e+04 3.00124604e+04 2.95568964e+04 2.9196876de+04 2.83028918e+04 2.76119026e+04 2.71469476e+04 2.68635733e+04 2.62973768e+04 2.59177090e+04 2.54854551e+04 2.47713296e+04 2.46279427e+04 2.44182450e+04 2.6572934e+04 2.35170051e+04 2.32946599e+04 2.27465199e+04 2.26572934e+04 2.22059626e+04 2.20168502e+04 2.17046569e+04 2.1485563e+04 2.13777561e+04 2.09561398e+04 2.05958378e+04 2.01792857e+04 1.99110890e+04 1.97462592e+04 1.93989379e+04 1.91706466e+04 1.8262529e+04 1.89128721e+04 1.86161666e+04 1.84307255e+04 1.82625283e+04 1.77938904e+04 1.77244216e+04 1.76113610e+04 1.71589960e+04 1.68367661e+04 1.65457212e+04 1.59353594e+04 1.58946632e+04 1.57918886e+04 1.55425899e+04 1.53567588e+04 1.51463792e+04 1.49894010e+04 1.48161188e+04 1.46551170e+04 1.45024051e+04 1.43756658e+04 1.43461612e+04 1.41258365e+04 1.40712492e+04 1.334208497e+04 1.38253800e+04 1.331171434e+04 1.30742738e+04 1.28788451e+04 1.27566296e+04 1.31171434e+04 1.30742738e+04 1.28788451e+04 1.27566296e+04 1.31171434e+04 1.30742738e+04 1.28788451e+04 1.27566296e+04 1.27566296e+04 1.31171434e+04 1.30742738e+04 1.28788451e+04 1.27566296e+04
	1.26772196e+04 1.25340524e+04 1.22875445e+04 1.22307690e+04 1.22040564e+04 1.21049076e+04 1.20783817e+04 1.19945612e+04 1.18638219e+04 1.17930370e+04 1.17214579e+04 1.163411989e+04 1.15232489e+04 1.14856385e+04 1.14147470e+04 1.13229245e+04 1.12380552e+04 1.11112439e+04 1.10121132e+04 1.09952820e+04 1.09144508e+04 1.07452317e+04 1.07235057e+04 1.06919947e+04 1.05993811e+04 1.05770290e+04 1.05435651e+04 1.04861092e+04 1.03836702e+04 1.02805081e+04 1.01310169e+04 1.00250433e+04 1.00006348e+04 9.89028832e+03 9.80449576e+03 9.77781478e+03 9.74330652e+03 9.71995705e+03 9.62011154e+03 9.60711546e+03 9.52069972e+03 9.48749308e+03 9.493975878e+03 9.40433513e+03 9.36532320e+03 9.28567577e+03 9.26408362e+03 9.20517355e+03 9.37132491e+03 9.02616564e+03 8.98456726e+03 8.96258241e+03 8.88626416e+03 8.84741884e+03 8.77108828e+03 8.68834808e+03 8.66257481e+03 8.44248381e+03 8.59436187e+03 8.53856375e+03 8.50984607e+03 8.44248381e+03 8.39436187e+03 8.37495437e+03 8.35913766e+03 8.26985016e+03 8.23433609e+03 8.21940474e+03 8.35913766e+03 8.26985016e+03 8.23433609e+03 8.21940474e+03 8.14902565e+03 8.14230124e+03 8.09862282e+03 8.06578577e+03
	8.01883291e+03 7.94676970e+03 7.93562192e+03 7.87605009e+03 7.85349826e+03 7.84602854e+03 7.80358152e+03 7.77627356e+03 7.72870027e+03 7.70226756e+03 7.65849802e+03 7.63900098e+03 7.61733433e+03 7.57619948e+03 7.53033977e+03 7.51021462e+03 7.48907078e+03 7.47043704e+03 7.42484638e+03 7.37670925e+03 7.33763898e+03 7.29582597e+03 7.24188532e+03 7.20837898e+03 7.17812509e+03 7.13972280e+03 7.13619589e+03 7.12310454e+03 7.08314909e+03 7.04665533e+03 7.02431993e+03 7.02024244e+03 7.00415212e+03 6.95554683e+03 6.94069840e+03 6.91932610e+03 6.88008509e+03 6.85033574e+03 6.81512928e+03 6.81335566e+03 6.78787458e+03 6.76618106e+03 6.76276775e+03 6.72983955e+03 6.67614866e+03 6.64397950e+03 6.63390952e+03 6.60485665e+03 6.58177455e+03 6.55484177e+03 6.51015291e+03 6.47996940e+03 6.47212899e+03 6.44145607e+03 6.41231664e+03 6.39876380e+03 6.38047442e+03 6.35491458e+03 6.34102403e+03 6.29594832e+03 6.28071962e+03 6.23247977e+03 6.22144006e+03 6.29594832e+03 6.06578690e+03 6.05173145e+03 6.04006284e+03 6.08394021e+03 6.06578690e+03 6.05173145e+03 6.04006284e+03 6.01691315e+03
	5.99347084e+03 5.98538406e+03 5.96735504e+03 5.95079821e+03 5.92893923e+03 5.90791935e+03 5.89073665e+03 5.86175618e+03 5.82635243e+03 5.79051077e+03 5.76805083e+03 5.75186680e+03 5.73798309e+03 5.72399138e+03 5.71822626e+03 5.67896658e+03 5.67799218e+03 5.64715999e+03 5.64235567e+03 5.53982335e+03 5.52349944e+03 5.51211044e+03 5.49456083e+03 5.46656938e+03 5.52349944e+03 5.51211044e+03 5.49456083e+03 5.38392795e+03 5.5826086e+03 5.34275482e+03 5.38392795e+03 5.3826086e+03 5.34275482e+03 5.31677218e+03 5.28869611e+03 5.26321107e+03 5.23973697e+03 5.23528431e+03 5.21102369e+03 5.19952816e+03 5.18350698e+03 5.14034731e+03 5.13388302e+03 5.12197796e+03 5.09337647e+03 5.09337647e+03 5.01400606e+03 4.97163528e+03 4.96174055e+03 4.95835374e+03 4.93783224e+03 4.92408445e+03 4.91888426e+03 4.91163518e+03 4.88630627e+03 4.86512281e+03 4.88422692e+03 4.88238543e+03 4.88630627e+03 4.86512281e+03 4.84222692e+03 4.83238543e+03 4.81146196e+03 4.772425940e+03 4.77083356e+03 4.76190975e+03 4.68349773e+03 4.67518878e+03 4.63384982e+03 4.62079495e+03 4.59913438e+03 4.59154912e+03 4.57799699e+03 4.56723012e+03 4.59913438e+03 4.59154912e+03 4.57799699e+03 4.56723012e+03
	2.06620560e+03 2.03658276e+03 2.00400383e+03 1.99586901e+03 1.98832867e+03 1.98253154e+03 1.97320348e+03 1.95159839e+03 1.93768386e+03 1.92629043e+03 1.92544553e+03 1.91803489e+03 1.89797087e+03 1.88130124e+03 1.86334948e+03 1.85589152e+03 1.81406935e+03 1.80833401e+03 1.77650603e+03 1.75838062e+03 1.75584564e+03 1.74079566e+03 1.72643677e+03 1.71861457e+03 1.69041444e+03 1.67605817e+03 1.67042015e+03 1.66890982e+03 1.61043944e+03 1.59367239e+03 1.57885447e+03 1.51204438e+03 1.6143944e+03 1.59367239e+03 1.48754776e+03 1.48445299e+03 1.54338026e+03 1.52692740e+03 1.48754776e+03 1.48445299e+03 1.47961364e+03 1.47830317e+03 1.47594596e+03 1.47384758e+03 1.46808449e+03 1.44916142e+03 1.4436896e+03 1.43073552e+03 1.39805777e+03 1.38849282e+03 1.38674639e+03 1.33944256e+03 1.33474693e+03 1.32475900e+03 1.31295819e+03 1.29905115e+03 1.28304071e+03 1.25992410e+03 1.24677685e+03 1.23549985e+03 1.23025376e+03 1.22339419e+03 1.21836413e+03 1.21016584e+03 1.19795572e+03 1.19550332e+03 1.18240674e+03 1.17905041e+03 1.16073154e+03 1.14869241e+03 1.13262297e+03 1.13053777e+03 1.10264439e+03 1.09329595e+03 1.07218424e+03 1.05182121e+03
	1.03795855e+03 1.02937165e+03 1.01805994e+03 1.01575982e+03 1.01278364e+03 9.95751781e+02 9.60079403e+02 9.51627701e+02 9.49041259e+02 9.43624460e+02 9.43238485e+02 9.34836259e+02 9.33096010e+02 9.28011233e+02 9.06056747e+02 8.95509678e+02 8.69345617e+02 8.67809543e+02 8.63775831e+02 8.4771998e+02 8.17796603e+02 8.04006157e+02 7.98874333e+02 7.93954638e+02 7.89891539e+02 7.85184817e+02 7.77699709e+02 7.69284671e+02 7.56115341e+02 7.53521347e+02 7.27439393e+02 7.15294042e+02 7.58981659e+02 6.85988770e+02 6.44261755e+02 6.43954045e+02 6.50411076e+02 6.44261755e+02 6.43954045e+02 6.05531146e+02 6.50411076e+02 6.30372260e+02 6.05531146e+02 6.40373750e+02 5.90391340e+02 5.80832281e+02 5.76396475e+02 5.70952422e+02 5.60469119e+02 5.56655426e+02 5.4492760e+02 5.42390709e+02 5.27679226e+02 5.44192760e+02 6.46338899e+02 6.48803368e+02 4.84425651e+02 6.474656642e+02 4.66363987e+02 4.88803368e+02 4.84425651e+02 4.74656642e+02 4.66363987e+02 4.88803368e+02 4.84425651e+02 4.74656642e+02 4.66363987e+02 4.88803368e+02 4.84425651e+02 4.31154607e+02 4.25636640e+02 4.07253331e+02 4.3848126e+02 4.31154607e+02 4.25636640e+02 4.07253331e+02 4.03427748e+02 4.31154607e+02 4.25636640e+02 4.07253331e+02 4.03427748e+02 3.94118315e+02 3.93718826e+02 3.74772698e+02 3.68770816e+02 3.94118315e+02 3.93718826e+02 3.74772698e+02 3.68770816e+02
	2.72643048e+01 2.57664948e+01 2.45900817e+01 1.96846116e+01 1.36756175e+01 1.19896389e+01 7.20891812e+00 6.65663438e+00 4.94542681e+00 6.08885521e-11 2.59239576e-11 2.31165481e-11 2.19782144e-11 1.87783080e-11 1.85734224e-11 1.73592757e-11 1.49796301e-11 1.48034890e-11 1.18987479e-11 1.14049965e-11 1.03784706e-11 8.51991657e-12 8.31313178e-12
In [24]: In [25]:	8.31313178e-12 8.31313178e-12 8.26238443e-12 7.25195395e-12 5.89863788e-12 5.46074232e-12 2.82532465e-12 1.88227097e-12] PCA.explained_variance ratio is used to get the ration of variance (eigenvalue / total eigenvalues). If we sum up the array's values it will be equal to 1 which means that if we gather all the principal components it will indeed explain the total of the variance of the data. Let's try to verify that using the following line of code: print (np.sum (pca.explained_variance_ratio_)) 0.999999999999998
In [26]:	This means that having 154 components will enable us to explain 95% of the variance of the data. Furthermore, We can also find it approximately by plotting the graph, using the following code: plt.plot(cumulative_explained_variance) plt.xlabel('Number of components') plt.ylabel('Cumulative explained variance %') plt.title('Cumulative explained variance Graph') plt.grid() plt.show() Cumulative explained variance Graph
	Indeed, we notice that having 154 components give us a cumulative explained variance of 95% Q3/ Display some MNIST pictures with different values of n_components.
In [27]:	<pre># Initialize the PCA model n_components = [25, 64, 100, 169, 225] for n in n_components: pca = PCA(n_components=n) reduced_d2_X = pca.fit_transform(d2_X_train) X_pca = pca.inverse_transform(reduced_d2_X) X_pca_d2_approx = X_pca.reshape(nsamples, nx, ny) # Plot some MNIST pictures with different values of n_components for i in range(5): plt.figure(figsize=(4,4)) plt.imshow(X_pca_d2_approx[i]) plt.title("n_components: {}".format(n)) plt.show()</pre>

5 10 15 20 25 25 n_components: 25 0 5 10 20 25 0 10 15 20 25 n_components: 25 0 -5 10 20 25 15 10 20 25 n_components: 25 0 5 10 20 25 10 15 25 n_components: 25 5 10 15 20 25 10 25 n_components: 64 0 5 10 15 20 25 25 0 10 15 20 n_components: 64 0 -5 10 20 25 10 15 20 25 n_components: 64 0 5 10 15 20 25 10 15 25 n_components: 64 5 10 15 20 25 n components: 64 0 5 10 15 20 25 0 10 15 25 n_components: 100 0 -5 10 15 20 25 10 15 20 25 n_components: 100 0 5 10 15 20 25 10 25 n_components: 100 0 5 10 15 20 25 10 15 20 25 n_components: 100 0 5 10 20 25 25 0 10 n_components: 100 0 -5 10 20 25 10 15 20 25 n_components: 169 0 5 10 20 25 n_components: 169 5 10 15 20 25 10 15 20 25 n components: 169 0 5 10 15 20 0 n_components: 169 0 -5 10 15 20 25 10 20 25 n_components: 169 0 5 10 15 20 25 n_components: 225 5 10 15 20 25 10 15 25 n_components: 225 0 5 10 15 20 25 n_components: 225 0 -5 · 15 20 25 15 20 25 n_components: 225 5 10 20 25 n_components: 225 5 10 15 20 25 10 15 20 25 2.2. Data clustering **Questions / Works** Q1/ You already split X (and y) in a train and test sets with the sklearn method: split_train_test. With sklearn, perform K-MEANS. Play with the parameter K as well as the initialization (KMEANS++, random, or fixed array) In [28]: from sklearn.cluster import KMeans In [29]: kmeans = KMeans(n clusters=2, random state=0, n init=1).fit(d2 X train) In [30]: kmeans.labels Out[30]: array([1, 0, 1, ..., 1, 0, 1]) In [31]: kmeans.predict(d2_X_train) Out[31]: array([1, 0, 1, ..., 1, 0, 1]) kmeans.cluster centers Out[32]: array([[0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.]]) In [33]: d2 X train.shape Out[33]: (56000, 784) k-means++ initialization In [34]: kmeans = KMeans(n clusters=5, init='k-means++').fit(d2 X train) In [35]: kmeans.predict(d2_X_train) Out[35]: array([2, 2, 3, ..., 3, 0, 0]) In [36]: kmeans.cluster centers Out[36]: array([[0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.], $[0., 0., 0., \ldots, 0., 0., 0.],$ [0., 0., 0., ..., 0., 0., 0.]]) Random initialization In [37]: kmeans = KMeans(n_clusters=10, init='random').fit(d2_X_train) In [38]: kmeans.predict(d2 X train) Out[38]: array([4, 3, 7, ..., 9, 1, 1]) In [39]: kmeans.cluster centers [0., 0., 0., ..., 0., 0., 0.], [0., 0., 0., ..., 0., 0., 0.],[0., 0., 0., ..., 0., 0., 0.]]) When using the kmeans ++ initialization, we find the most distant initial centroids, whereas with a random initialization those centroids are chosen randomly. On the other hand, the fixed array initialization chose the initial centroids from the fixed array provided in init but it's annoying to do this type of initialization since we will need to create an array of shape (3,784), "3" being the number of clusters and "784" the number of features of the data. Q2/ For the correct K (K=10), evaluate how good is this partition (with the knowledge of y) In [40]: from sklearn.metrics.cluster import homogeneity score from sklearn.metrics.cluster import completeness score from sklearn.metrics.cluster import v measure score from sklearn.metrics import adjusted rand score res model = [] res param = [] res train_acc = [] res valid_acc = [] res test acc = [] In [41]: nsamples, nx, ny = X_test.shape d2_X_test = X_test.reshape((nsamples,nx*ny)) ncluster = 10 kmeans = KMeans(n_clusters=ncluster, init='k-means++').fit(d2 X train) y_pred = kmeans.predict(d2_X_test) print("Evaluation of the clustering (K-Means, all information):") print("Homoegneity within clusters:", homogeneity_score(y_test, y_pred)) print("Completeness score:", completeness score(y test, y pred)) print("Adjusted Rand Score:", adjusted_rand_score(y_test, y_pred)) print("V-measure: ", v_measure_score(y_test, y_pred)) res_model.append("K-means") res param.append("10 clusters \nAll data") res_train_acc.append("-") res_valid_acc.append("-") res test acc.append(v measure score(y test, y pred)) Evaluation of the clustering (K-Means, all information): Homoegneity within clusters: 0.49695622802035133 Completeness score: 0.5053233300704322 Adjusted Rand Score: 0.3618521352209539 V-measure: 0.5011048544596556 homogeneity_score: gives the degree to which all of the clusters contain only data points which are members of a single class. • completeness_score: gives the degree to which all the data points that are members of a given class are assigned to the same cluster. adjusted_rand_score: gives a normalized measure of similarity between two label assignments. v_measure_score: gives a combination of homogeneity and completeness score. We notice that the results are not good thus it's not reliable. Q3/ Using the PCA performed in section 2 apply K-MEANS with K=10 and $n_components = 2$. Display the partition and comment. In [42]: pca = PCA(n_components=2) reduced_d2_X_train = pca.fit_transform(d2_X_train) reduced_d2_X_test = pca.transform(d2_X_test) kmeans = KMeans(n_clusters=10, init='k-means++').fit(reduced_d2_X_train) y_pred = kmeans.predict(reduced_d2_X_test) print("Evaluation of the clustering (K-Means, 2 principal components):") print("Homoegneity within clusters:", homogeneity score(y test, y pred)) print("Completeness score:", completeness_score(y_test, y_pred)) print("V-measure: ", v_measure_score(y_test, y_pred)) plt.suptitle('KMeans in 2 dimensions compared with ground truth') plt.subplot(1,2,1)plt.title('KMeans with 2 principle components') for i in range (0,10): lbl = 'Class ' + str(i) plt.scatter(reduced d2 X test[(y pred==i),0], reduced d2 X test[(y pred==i),1], s=1, label=lbl) plt.legend(prop={'size': 8}) plt.subplot(1,2,2)plt.title('Ground truth') for i in range (0,10): lbl = 'Class ' + str(i) plt.scatter(reduced_d2_X_test[(y_test==i),0], reduced_d2_X_test[(y_test==i),1], s=1, label=lbl) plt.legend(prop={'size': 8}) plt.show() res_model.append("K-means") res_param.append("10 clusters \n2 principle components") res_train_acc.append("-") res_valid_acc.append("-") res_test_acc.append(v_measure_score(y_test, y_pred)) Evaluation of the clustering (K-Means, 2 principal components): Homoegneity within clusters: 0.3550043587286218 Completeness score: 0.35963802661256083 V-measure: 0.3573061705570845 KMeans in 2 dimensions compared with ground truth KMeans with 2 principle components Ground truth Class 0 Class 1 Class 1 Class 2 1000 Class 2 1000 Class 3 Class 3 Class 4 Class 4 500 500 Class 5 Class 5 Class 6 Class 6 Class 7 Class 7 0 0 Class 8 Class 9 Class 9 -500500 -10001000 -1500-1000 1000 The homogeneity score and the completeness score indicates that the k-means model is not doing performing well to separate the samples into the correct clusters. The v-measure score is also relatively low. This suggests that this k-means model is not a good fit for this dataset. We will need to consider using another clustering algorithm or just change the parameters of the model. Q4/ Do the same job with the EM-clustering using the good K parameter (10 for MNIST). Comment your results. In [43]: from sklearn import mixture emgm = mixture.GaussianMixture(n components=10,covariance type='full').fit(reduced d2 X train) y pred = emgm.predict(reduced d2 X test) print("Evaluation of the clustering (EMGM, 2 principal components):") print("Homoegneity within clusters:", homogeneity_score(y_test, y_pred)) print("Completeness score:", completeness_score(y_test, y_pred)) print("V-measure: ", v_measure_score(y_test, y_pred)) plt.figure(figsize=(10,5)) plt.suptitle('Expectation-Maximization with Gaussian Mixture in 2 dimensions compared with ground truth') plt.subplot(1,2,1)plt.title('EMGM with 2 principle components') for i in range (0,10): lbl = 'Class ' + str(i) plt.scatter(reduced_d2_X_test[(y_pred==i),0], reduced_d2_X_test[(y_pred==i),1], s=1, label=lbl) plt.legend(prop={'size': 8}) plt.subplot(1,2,2)plt.title('Ground truth') for i in range (0,10): lbl = 'Class ' + str(i) plt.scatter(reduced_d2_X_test[(y_test==i),0], reduced_d2_X_test[(y_test==i),1], s=1, label=lbl) plt.legend(prop={'size': 8}) plt.show() res model.append("EM Gaussian Mix") res param.append("10 clusters \n2 principle components") res_train_acc.append("-") res_valid_acc.append("-") res_test_acc.append(v_measure_score(y_test, y_pred)) Evaluation of the clustering (EMGM, 2 principal components): Homoegneity within clusters: 0.36034412909630886 Completeness score: 0.3695990552084314 V-measure: 0.3649129206973976 Expectation-Maximization with Gaussian Mixture in 2 dimensions compared with ground truth EMGM with 2 principle components Ground truth 1500 1500 Class 0 Class 1 Class 1 Class 2 Class 2 1000 1000 Class 3 Class 3 Class 5 Class 5 Class 6 Class 6 500 500 Class 7 Class 7 Class 8 Class 9 0 0 -500-500-1000-1000500 1000 1500 2000 2500 500 1000 1500 2000 2500 -1000 -500 In this case, this second k-means model is a little better than the previous one. However, we feel like the model is still not able to consider the complexity of the data and it still confuses some classes especially that we only considered two principal components. 3. Supervised Machine Learning 3.1. Decision Tree, SVM and Logistic Regression **Questions / Works** Q1/ What is the major difference between Naïve Bayes Classifier and Support Vector Machine (or Logistic Regression)? (if you forgot the course #2, a clue: what are we trying to predict?) Naive Bayes classifier is a generative model that evaluates P(X|y) to know P(y|X) whereas SVM and logistic regression are discriminative models that estimate P(y|X). Q2/ With sklearn, perform a classification using your favorite methods. With the documentation, check how to modify the parameters and comment how it influences the results. For example, if you chose SVM, change the kernel between 'linear' and 'rbf' (Gaussian kernel). You can also play with 'C' parameter to switch from hard ☐ margin SVM to soft-margin **SVM2...** Q2/1. Decision Tree: In [44]: from sklearn import tree from sklearn.tree import DecisionTreeClassifier tree1 = DecisionTreeClassifier() tree1.fit(d2_X_train, y_train) print("Model tree classifier: unrestricted model") print("Depth: ", tree1.get_depth(), ", Number of leaves: ", tree1.get_n_leaves()) print("Accuracy on the training set: ", tree1.score(d2 X train, y train)) print("Accuracy on the test set: ", tree1.score(d2 X test, y test)) res model.append("Decision tree") res param.append("Unrestricted max depth \nAll information") res train acc.append(tree1.score(d2 X train, y train)) res valid acc.append("-") res_test_acc.append(tree1.score(d2_X_test, y_test)) depths = range(5, 21) #evaluation for each depth of tree nleaves = len(depths)*[None] accuracies_train = len(depths)*[None] accuracies_test = len(depths)*[None] for depth in depths: tree2 = DecisionTreeClassifier(max depth=depth) model = tree2.fit(d2_X_train, y_train) nleaves[depth-depths[0]] = tree2.get_n_leaves() accuracies_train[depth-depths[0]] = tree2.score(d2_X_train, y_train) accuracies_test[depth-depths[0]] = tree2.score(d2_X_test, y_test) plt.figure(figsize=(10,5)) plt.subplot(1,2,1)plt.title('Number of leaves plotted against max depth') plt.plot(depths, np.log2(nleaves), label='restricted model', color='b') plt.axhline(np.log2(tree1.get n leaves()), 0, depths[-1], color='darkblue', linestyle='--', label='log2 of n le plt.xlabel("Depth") plt.ylabel("Log base 2 of the number of leaves") plt.legend() plt.subplot(1,2,2)plt.title('Accuracy plotted against max depth') plt.plot(depths, accuracies train, label='Accuracy on train set, restricted model', color='r') plt.axhline(tree1.score(d2_X_train, y_train), 0, depths[-1], color='darkred', linestyle=' plt.plot(depths, accuracies_test, label='Accuracy on test set, restricted model', color='b') plt.axhline(tree1.score(d2_X_test, y_test), 0, depths[-1], color='darkblue', linestyle='--', label='Accuracy or plt.xlabel("Depth") plt.ylabel("Accuracy") plt.legend() plt.suptitle("Comparison of several depths of tree") res model.append("Decision tree") res param.append("Max depth: 15 \nAll information") res train acc.append(accuracies train[10]) res valid acc.append("-") res_test_acc.append(accuracies_test[10]) Model tree classifier: unrestricted model Depth: 50 , Number of leaves: 3742 Accuracy on the training set: 1.0 Accuracy on the test set: 0.8699285714285714 Comparison of several depths of tree Number of leaves plotted against max depth Accuracy plotted against max depth 11 Log base 2 of the number of leaves 10 0.90 Accuracy 0.80 0.75 Accuracy on train set, restricted model ———— Accuracy on train set, unrestricted model 0.70 restricted model Accuracy on test set, restricted model --- Accuracy on test set, unrestricted model log2 of n_leaves for the unrestricted model 0.65 10.0 12.5 12.5 15.0 17.5 15.0 17.5 20.0 Depth Depth Q2/2. SVM: In []: from sklearn import svm nbfit = 10 #number of svm to be fitted, must have nbfit%4 = 0 kernels = nbfit//4*['rbf'] + nbfit//4*['linear'] + nbfit//4*['poly'] + nbfit//4*['sigmoid']#the first batch of Cs = nbfit//4*[0.1, 0.5, 1, 2] #we test the same Cs with the two types of kernel accuracies = nbfit*[None] #table of accuracies pca = PCA(n components=pca.n components) #PCA to reduce dimensionality reduced d2 X train = pca.fit transform(d2 X train) reduced d2 X test = pca.transform(d2 X test) nsamples = reduced d2 X train.shape[0] #number of samples in the total train set for k in range(0, nbfit): cur svm = svm.SVC(kernel=kernels[k], C=Cs[k], gamma='auto') #selects the parameters of the current SVM indexes train = list(range(k*nsamples//nbfit, (k+1)*nsamples//nbfit)) #indexes used to train the current indexes valid = list(range(0, k*nsamples//nbfit)) + list(range((k+1)*nsamples//nbfit, nsamples)) #index reduced d2 X train svm = reduced d2 X train[indexes train] reduced d2 X valid svm = reduced d2 X train[indexes valid] y train svm = y train[indexes train] y valid svm = y train[indexes valid] cur svm.fit(reduced d2 X train svm, y train svm) #training of the current svm print("SVM fitted: ", k+1, "/", nbfit) accuracies[k] = cur_svm.score(reduced_d2_X_valid_svm, y_valid_svm) #accuracy of the current SVM on its valid_svm index bestSVM = np.argmax(accuracies) print("Best SVM: kernel ", kernels[index bestSVM], ", C=", Cs[index bestSVM], ", Accuracy on validation set: ", bestSVM = svm.SVC(kernel=kernels[index bestSVM], C=Cs[index bestSVM], gamma='auto') indexes train = list(range(index bestSVM*nsamples//nbfit, (index bestSVM+1)*nsamples//nbfit)) reduced d2 X train svm = reduced d2 X train[indexes train] y train svm = y train[indexes train] bestSVM.fit(reduced d2 X train svm, y train svm) print("Accuracy on test set: ", bestSVM.score(reduced d2 X test, y test)) res model.append("SVM") res param string = "K-cross validation: 20 folds \n154 principle components \nkernel" + str(kernels[index bests res param.append(res param string) res train acc.append("-") res valid acc.append(accuracies[index bestSVM]) res test acc.append(bestSVM.score(reduced d2 X test, y test)) SVM fitted: 1 / 10 SVM fitted: 2 / 10 SVM fitted: 3 / 10 SVM fitted: 4 / 10 Q2/3. Naive Bayes: In [45]: from sklearn.naive bayes import GaussianNB naivebayes = GaussianNB() naivebayes.fit(d2 X train, y train) print("Naive Bayes Classifier: ") print("Accuracy on train set: ", naivebayes.score(d2 X train, y train)) print("Accuracy on test set: ", naivebayes.score(d2_X_test, y_test)) res model.append("Gaussian Naive Bayes") res param.append("-") res train acc.append(naivebayes.score(d2 X train, y train)) res valid acc.append("-") res test acc.append(naivebayes.score(d2 X test, y test)) Naive Bayes Classifier: Accuracy on train set: 0.56475 Accuracy on test set: 0.5612857142857143 Q2/4 Logistic Regression: In [46]: from sklearn.linear model import LogisticRegression #from sklearn.preprocessing import StandardScaler # create StandardScaler instance #scaler = StandardScaler() # fit and transform the data #X scaled = scaler.fit transform(d2 X train) # use the scaled data for logistic regression lr2 = LogisticRegression(penalty='12', dual=False, C=1, solver='lbfgs', multi class='auto', max iter=1000) lr2.fit(d2 X train, y train) lr1 = LogisticRegression(penalty='11', dual=False, C=1, solver='liblinear', multi class='auto', max iter=1000) lr1.fit(d2 X train, y train) print("Logistic regression (penalty L1):") print("Accuracy on train set: ", lr1.score(d2 X train, y train), "Accuracy on test set: ", lr1.score(d2 X test, print("Logistic regression (penalty L2):") print("Accuracy on train set: ", lr2.score(d2 X train, y train), "Accuracy on test set: ", lr2.score(d2 X test, res model.append("Logistic regression") res param.append("Penalty L1 \nAll information") res train acc.append(lr1.score(d2 X train, y train)) res valid acc.append("-") res_test_acc.append(lr1.score(d2_X_test, y_test)) res model.append("Logistic regression") res param.append("Penalty L2 \nAll information") res train acc.append(lr2.score(d2_X_train, y_train)) res valid acc.append("-") res_test_acc.append(lr2.score(d2_X_test, y_test))

n_components: 25

0

cy on train set: 0.56 cy on test set: 0.56 or test set: 0.56 or the data points in the train set and again 56% on the test se es model is not really appropriate to classify our dataset's images.
c regression we got: c regression (penalty L1): ccy on train set: 0.93 ccy on test set: 0.92 c regression (penalty L2): ccy on train set: 0.94
on: Whether we use L1 or L2 norm, we get quite the same result. Although, the lbfgs didn't converge, we still have an the test sets because convergence is not a sufficient condition for a good model, however, it is a necessary condition optimal solution. Section 2.1. you applied a PCA to X so that the projected set – hereafter Xred –lied" space. Among the supervised methods you chose, select one method and a de to Xred. Does the PCA influence the performance of the classification (accordinately of the reduction)?
<pre>ettytable import PrettyTable [0.66, 0.8, 0.9, 0.95] = len(info)*[None] = len(info)*[None] ies_train = len(info)*[None] ies_test = len(info)*[None] ex in range(0, len(info)): m = info[index] = PCA(elem) ies_ded_d2_X_train = pca.fit_transform(d2_X_train) ies_ded_d2_X_test = pca.transform(d2_X_test) ies_ded_d2_X_test = pca.transfo</pre>
el.append("Decision tree") am.append("Unrestricted max depth \n66% of information") in_acc.append(accuracies_train[0]) id_acc.append("-")
t_acc.append("-") t_acc.append(accuracies_test[0]) t_acc.append(accuracie
Deep Learning ayer Perceptron (MLP) fons / Works at is the size of the input tensor? What is the size of the output layer? ensor is (samples_number × 784 × 1) The size of the output tensor is 10 since we have ten neurons (ten outcomes). w many epochs do you use? What does it mean? What is the batch_size? What
och is the number of passes a training dataset takes around a machine learning algorithm. In our case, since we have with a lot of neurons per layer, we will then need less epochs to get a great performance of the model (approximately s). Size is the number of training examples utilized in one iteration. So in our case, the batch_size = samples_number. By do we define a validation set (for example: validation_split=0.2)? Sion set is used to evaluate the performance of the training after every epoch. By the most important parameters you have to set with the compile and the fit recomplain why they are important parameters, i.e. they influence the training process.
eters are the loss (sparse categorical crossentropy), the optimizer(Adam optimizer), the metrics we want to evaluate the accuracy in our case, epochs and the batch_size. Closs measures the difference between the predicted outputs and the targeted outputs. The choice of the loss function of the model parameters during the solve and its evaluation metrics. The optimizer is used to update the model parameters during nitor the model's performance during training and the choice of metrics to use depends again on the problem we are model train for a very long time. And finally, the batch_size sets the number of samples to be processed in one for eass. Unlike epochs, having a larger batch size makes the training faster but it will need more memory. Insorflow import keras Insorflow.keras.layers import Dense, Flatten, Conv2D, Dropout, MaxPooling2D, Flatten
<pre>insorflow.keras import Model = 10 = keras.Input(shape=(d2_X_train.shape[1],)) se(32, activation='relu') (inputs) = Dense(10, activation='softmax') (x) odel(inputs, outputs, name="MLPO") pile(loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True), optimizer=keras.opt. t = mlp.fit(d2_X_train, y_train, epochs=epochs, validation_split=0.2, verbose=0) re = mlp.evaluate(d2_X_test, y_test, verbose=0) mary() Activation function: ReLu (Softmax for output), number of epochs:", epochs) are(figsize=(9, 4.5)) are(figsize=(9, 4.5))</pre>
<pre>plot(1,2,1) title("History of the trainning for MLPO") t(mlp_hist.epoch, mlp_hist.history['accuracy'], 'b', label='Train') t(mlp_hist.epoch, mlp_hist.history['val_accuracy'], 'b', label='Validation') plet('Epochs') plet('Accuracy') le('Accuracy on train and validation set') end() plot(1,2,2) t(mlp_hist.epoch, mlp_hist.history['loss'], 'b', label='Train') t(mlp_hist.epoch, mlp_hist.history['val_loss'], 'b', label='Validation') plet('Epochs') plet('Loss') le('Loss on train and validation set') end() w()</pre>
Ip.name, ": Test loss: ", mlp_score[0], "Test accuracy: ", mlp_score[1]) el.append("MLP") am_string = "1 hidden layer," + str(mlp.count_params()) + " parameters \n" + str(mlp_hist.ep. am.append(res_param_string) in_acc.append(mlp_hist.history['accuracy'][-1]) id_acc.append(mlp_hist.history['val_accuracy'][-1]) t_acc.append(mlp_score[1]) \ayael\anaconda3\lib\site-packages\tensorflow\python\util\dispatch.py:1082: UserWarning: "`s rossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoi tion and thus does not represent logits. Was this intended?" dispatch_target(*args, **kwargs) MLPO" type) Output Shape Param #
(InputLayer) [(None, 784)] 0 4 (Dense) (None, 32) 25120 5 (Dense) (None, 10) 330 rams: 25,450 e params: 25,450 nable params: 0 on function: ReLu (Softmax for output), number of epochs: 10 History of the trainning for MLP0 suracy on train and validation set Loss on train and validation set
2.0 - Train Validation
Epochs est loss: 0.34652629494667053 Test accuracy: 0.9241428375244141 mment the training results 6 accuracy on the test set which is quite good. We are not underfitting nor overfitting. The MLPO model is not really one using a relatively simple dataset. For other complex data, it could be a sign of overfitting. here any overfitting? Why? If yes, what could be the causes? How to fix this issue not observe overfitting, how can you make your model overfit? Try and demorality.
<pre>= [0, 50, 10, 50] #epoch[0] is not to be used = keras.Input(shape=(d2_X_train.shape[1],)) se(32, activation='relu') (inputs) = Dense(10, activation='softmax') (x) Model(inputs, outputs, name="MLP1") = keras.Input(shape=(d2_X_train.shape[1],)) se(128, activation='relu') (inputs) se(128, activation='relu') (x) dodel(inputs, outputs, name="MLP2") = keras.Input(shape=(784,))</pre>
<pre>se(128, activation='relu') (inputs) se(128, activation='relu') (x) se(128, activation='r</pre>
ore = mlp3.evaluate(d2_X_test, y_test, verbose=0) are(figsize=(9, 4.5)) plot(1,2,1) title("History of the training for MLP1, MLP2, MLP3") t(mlp1_hist.epoch, mlp1_hist.history['accuracy'], 'b', label='MLP1, train') t(mlp1_hist.epoch, mlp1_hist.history['val_accuracy'], 'b', label='MLP1 validation') t(mlp2_hist.epoch, mlp2_hist.history['accuracy'], 'r', label='MLP2, train') t(mlp2_hist.epoch, mlp2_hist.history['val_accuracy'], 'r', label='MLP2 validation') t(mlp3_hist.epoch, mlp3_hist.history['val_accuracy'], 'g', label='MLP3, train') t(mlp3_hist.epoch, mlp3_hist.history['val_accuracy'], 'g', label='MLP3 validation') bel('Epochs') bel('Accuracy') le('Accuracy on train and validation set')
<pre>le('Accuracy on train and validation set') end() plot(1,2,2) t(mlp1_hist.epoch, mlp1_hist.history['loss'], 'b', label='MLP1, train') t(mlp1_hist.epoch, mlp1_hist.history['val_loss'], 'b', label='MLP1 validation') t(mlp2_hist.epoch, mlp2_hist.history['loss'], 'r', label='MLP2, train') t(mlp2_hist.epoch, mlp2_hist.history['val_loss'], 'r', label='MLP2 validation') t(mlp3_hist.epoch, mlp3_hist.history['loss'], 'g', label='MLP3, train') t(mlp3_hist.epoch, mlp3_hist.history['val_loss'], 'g', label='MLP3 validation') bel('Epochs') bel('Loss') le('Loss on train and validation set') end() w() lp1.name,": Test loss:", mlp1_score[0], "Test accuracy:", mlp1_score[1]) lp2.name,": Test loss:", mlp2_score[0], "Test accuracy:", mlp2_score[1]) lp3.name,": Test loss:", mlp3_score[0], "Test accuracy:", mlp3_score[1])</pre>
<pre>lp3.name,": Test loss:", mlp3_score[0], "Test accuracy:", mlp3_score[1]) el.append("MLP") am_string = "1 hidden layer," + str(mlp1.count_params()) + " parameters \n" + str(mlp1_hist. am.append(res_param_string) in_acc.append(mlp1_hist.history['accuracy'][-1]) id_acc.append(mlp1_hist.history['val_accuracy'][-1]) t_acc.append(mlp1_score[1]) el.append("MLP") am_string = "4 hidden layers," + str(mlp2.count_params()) + " parameters \n" + str(mlp2_hist am.append(res_param_string) in_acc.append(mlp2_hist.history['accuracy'][-1]) id_acc.append(mlp2_hist.history['val_accuracy'][-1]) t_acc.append(mlp2_score[1]) el.append("MLP") am_string = "4 hidden layers," + str(mlp3.count_params()) + " parameters \n" + str(mlp3_hist am.append(res_param_string)</pre>
am_string = "4 hidden layers," + str(mlp3.count_params()) + " parameters \n" + str(mlp3_hist am.append(res_param_string) in_acc.append(mlp3_hist.history['accuracy'][-1]) id_acc.append(mlp3_hist.history['val_accuracy'][-1]) t_acc.append(mlp3_score[1]) \ayael\anaconda3\lib\site-packages\tensorflow\python\util\dispatch.py:1082: UserWarning: "`s rossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoi tion and thus does not represent logits. Was this intended?" dispatch_target(*args, **kwargs) History of the training for MLP1, MLP3 turacy on train and validation set Loss on train and validation
2.0
<pre>e keras.Input(shape=(d2_X_train.shape[1],)) se(128, activation='relu') (inputs) se(64, activation='relu') (x) se(32, activation='relu') (x) = Dense(10, activation='softmax') (x) Model(inputs, outputs, name="MLP4") = keras.Input(shape=(d2_X_train.shape[1],)) se(128, activation='relu') (inputs) cout(0.15) (x) se(64, activation='relu') (x) cout(0.15) (x) se(32, activation='relu') (x) se(32, activation='relu') (x) = Dense(10, activation='softmax') (x)</pre>
(InputLayer)
1.75 1.50 1.25 1.75 1.50 1.75 1.50 1.25 1.75 1.50 1.75 1.50 1.75 1.50 1.75 1.50 1.75 1.50 1.75 1.50 1.75 1.50 1.50 1.50 1.50 1.50 1.50 1.50 1.5
arly overfitting since its training loss << validation loss and its validation loss keeps increasing while the train loss keeps increasing while t
eal with an image classification problem. mment the training results. Is it better than your MLP? Faster? Lighter? = 15 = keras.Input(shape=(28, 28, 1)) v2D(filters=64, kernel_size=3, strides=1, padding="valid", activation='relu', input_shape=(3 tten() (x)) se(64, activation='relu') (x) = Dense(10, activation='softmax') (x) odel(inputs, outputs, name="CNN0")
<pre>pile(loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True), optimizer=keras.opt t = cnn.fit(X_train, y_train, epochs=epochs, validation_split=0.2, verbose=0) re = cnn.evaluate(X_test, y_test, verbose=0) pre(figsize=(9, 4.5)) plot(1,2,1) title("History of the trainning for CNNO") t(cnn_hist.epoch, cnn_hist.history['accuracy'], 'b', label='train') t(cnn_hist.epoch, cnn_hist.history['val_accuracy'], 'b', label='validation') pel('Epochs') pel('Accuracy') le('Accuracy on train and validation set') end() plot(1,2,2) t(cnn_hist.epoch, cnn_hist.history['loss'], 'b', label='train')</pre>
t(cnn hist.epoch, cnn hist.history['val_loss'], 'b', label='validation') pel('Epochs') pel('Loss') le('Loss on train and validation set') end() w() CNN: Test loss:", cnn_score[0], "Test accuracy:", cnn_score[1]) el.append("CNN") am_string = "1 convolutionnal layer," + str(cnn.count_params()) + " parameters \n" + str(cnn am.append(res_param_string) in_acc.append(cnn_hist.history['accuracy'][-1]) id_acc.append(cnn_hist.history['val_accuracy'][-1]) t_acc.append(cnn_score[1])
CNNO" type)
CNNO" type) Output Shape Param # (InputLayer) [(None, 28, 28, 1)] 0 (Conv2D) (None, 26, 26, 64) 640 (Flatten) (None, 43264) 0 0 (Dense) (None, 64) 2768960 1 (Dense) (None, 10) 650
CNNO" (InputLayer) [(None, 28, 28, 1)] 0 (Conv2D) (None, 26, 26, 64) 640 (Flatten) (None, 43264) 0 (InputLayer) (None, 64) 2768960 1 (Dense) (None, 10) 650 Tams: 2,770,250 e params: 2,770,250 e params: 2,770,250 mable params: 0 History of the trainning for CNNO couracy on train and validation set Loss on train and validation set 175 150 075 075 075 075 075 075
CONNUT (InputLayer) (None, 28, 28, 1)] 0 (Conv27) (None, 26, 26, 68) 640 (Flatten) (None, 43264) 0 (Conv27) (None, 43264) 0 (Dense) (None, 10) 650 Frams: 2,770,250 8 params: 0 History of the trainning for CNNO curacy on train and validation set 175 150 125 100 125 125
Community (None, 28, 28, 1)] (Innotity (None, 28, 28, 1)] (Community) (Community
Comparison (Name, 28, 28, 13) 0 Comparison (Name, 28, 28, 13) 0 Comparison (Name, 43264) 0 Comparison (Name, 43264) 0 Comparison (Name, 43264) 0 Comparison (Name, 10) 650 Compar
Clause Canada (Clause 282, 283, 111 0 (Clause 282, 283, 111 0 (Clause 282) (Clause 282, 283, 283, 283, 283, 283, 283, 283,
October Shape Faces Frequency
The plan agent of the first performance, change the architecture of the CNN(add/remail change and remails and separate performance and remails and separate performance and remails and separate performance and remails and r
On the Property of the Common
To the contract of the contrac