Anomaly Detection
Lab 5 - PCA, Autoencoders

Andrei Hîji

December 2024

In this lab we will use anomaly detection algorithms based on PCA and Autoencoder.

1 PCA main ideas

- PCA starts from the covariance matrix of the mean-centered data matrix $X \in^{N\times d}$
$$\Sigma = \frac{X^T X}{N} = P\Delta P^\top$$
such that $\Sigma \in R^{d\times d}$ where element Σ_{ij} is the covariance between data dimension i and j
- the EVD of $\Sigma = P\Delta P^\top$ item Δ is diagonal and contains the eigenvalues between λ_{max} and λ_{min}
- $P \in^{d\times d}$ represents the orthonormal eigenvectors of the covariance corresponding to Δ

  Algorithm that computes anomaly score based on distances along all eigenvectors:

  1. EVD: $\Sigma = P\Delta P^\top$
  2. Transform: $X' = XP$
  3. Normalize: $X' = X'\Delta^{\frac{-1}{2}}$
  4. Anomaly score: squared Euclidean distance of the transformed data point from the center of the transformed data

A few methods that will be useful to you throughout this lab session are:

- **numpy.linalg.eigh** returns the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix; its parameters include:

  - Hermitian or real symmetric matrix whose eigenvalues and eigenvectors are to be computed: a

- **numpy.random.multivariate_normal** draws random samples from a multivariate normal distribution; its parameters include:
  - mean of the N-dimensional distribution: mean
  - covariance matrix of the distribution (must be symmetric and positive-semidefinite)
  - size of the generated dataset: size
- **numpy.cumsum** returns the cumulative sum of the elements along a given axis; its parameters include:
  - input array: a
  - axis along which the cumulative sum is computed: axis - optional
- **generate_data()** is used to create a synthetic dataset where normal samples are generated using a multivariate Gaussian distribution while outliers are generated by a uniform distribution; its parameters include:
  - number of training/testing points: n_train, n_test
  - number of features: n_features
  - proportion of outliers: contamination
  - range of values: offset
- **sklearn.model_selection.train_test_split** splits datasets into random train and test subsets and include the parameters:
  - Input data: *arrays
  - Test data size and train data size: test_size=None, train_size=None
  - random_state=None
  - shuffle=True
  - stratify=None - used to split in a stratified fashion
- **sklearn.metrics.roc_auc_score** computes ROC AUC:
  - True labels: y_true
  - Obtained scores: y_scores
- **scipy.io.loadmat** loads data from a MATLAB file and includes the following parameters:
  - file_name
  - Dictionary in which to insert matfile variables: mdict=None

  - appendmat=True : to append the .mat extension to the end of the given filename

- **pyod.utils.utility.standardizer** transforms data to zero-mean and unit variance; it includes the parameters:
  - Training samples and test samples: X, X_t=None

- **numpy.quantile** computes the q-th quantile of the data along the specified axis (the value below which the specified percentage of data falls); its parameters include:
  - Input array: a
  - Probability or sequence of probabilities of the quantiles to compute. Values must be between 0 and 1 inclusive: q

- **tensorflow.clip_by_value** clips tensor values to a specified min and max; its parameters include **clip_value_min** and **clip_value_max**; any values less than **clip_value_min** are set to **clip_value_min** and any values greater than **clip_value_max** are set to **clip_value_max**

2 Exercises

2.1 Ex. 1

1. In the first exercise you will generate a 3D dataset with 500 points using **np.random.multivariate_normal** function with the mean vector **[5, 10, 2]** and the covariance matrix [**[3, 2, 2]**, **[2, 10, 1]**, **[2, 1, 2]**] and plot it (3D). Then you will perform the PCA steps from the course (center data, compute covariance matrix, EVD).

2. Plot in the same figure both the cumulative explained variance (computed with the sorted [descending] eigenvalues and the **numpy.cumsum** function) with the **pyplot.step** function and the individual variances (respecting the order) - using the **pyplot.bar** function.

3. Project the data in the new space and identify the outliers based on the deviation of the values over the dimension corresponding to the 3rd principal component (compared to the mean of all the values of the same component). Use **0.1** as contamination rate and the **numpy.quantile** function in order to find the corresponding threshold and predict the labels. Plot the dataset again (using a different color for points labeled as anomalies). Repeat the same steps for the second principal component.

4. Project the data in the new space and identify the outliers based on the normalized distance (by the corresponding standard deviation) of the data points to the centroid (in the new space) along all the principal components (follow the steps from the algorithm in the first part of the lab). Plot the dataset again (using a different color for points labeled as anomalies)

2.2 Ex. 2

1. In this exercise you will use the shuttle dataset. Split the data in a training set and a testing set (60% of data). Standardize your data and fit a **pyod.models.pca.PCA** model with the training set using the real contamination rate of the training set. Plot the cumulative explained variance and the individual variances as in the previous exercise (you can access the variances with the **explained_variance_** attribute).

2. Compute the balanced accuracy for both the train and test sets. Then fit the **pyod.models.kpca.KPCA** model with the same training data and compute the scores again.

2.3 Ex. 3

For the last 2 exercises you will need to install tensorflow using **pip install tensorflow**.

1. In this exercise you will use the shuttle dataset from ODDS. Load the data using **scipy.io.loadmat()** and use **train_test_split()** to split it into train and test subsets (use 50% of data for testing). Use min-max normalization to bring your train data in the [0-1] range.

2. Design an Autoencoder class that subclasses **keras.Model**. Use the **keras.Sequential** model to create encoder and decoder submodels that contain only **keras.layers.Dense** layers. The 2 submodels should contain layers with [8, 5, 3] and [5, 8, 9] output units. Use **relu** activation function for each layer except the last one (from the decoder), which will use **sigmoid** activation.

3. Compile your model using **adam** optimizer and **mse** loss and fit it with your training data using **100 epochs** and a batch size of **1024** (use the test data as validation data in the trainig process). Plot the training and validation loss.

4. In order to obtain the scores for the training data pass it through the autoencoder and get the reconstruction error for each sample. Compute a threshold that will be used to classify data with the **numpy.quantile** function and the contamination rate of the dataset. Compute the balanced accuracy for both the training and testing set.

2.4 Ex. 4

1. In this exercise we will use the mnist dataset from **tensorflow.keras.datasets.mnist**. After you load the dataset with **tensorflow.keras.datasets.mnist.load_data()** you will normalize it by dividing with 255. In order to simulate anomalies, you will add some noise to the images with **tensorflow.random.normal** (multiplied by a factor of 0.35). You will use **tensorflow.clip_by_value** to keep the range of the pixels [0, 1].

2. Design a Convolutional Autoencoder class that uses the **keras.Sequential** model to create encoder and decoder submodels that contain **keras.layers.Conv2D** and **keras.layers.Conv2DTranspose** layers. The encoder will contain:

   - 1 **Conv2D** layer with 8 (3 X 3) filters, relu activation, strides=2 and padding - 'same'
   - 1 **Conv2D** layer with 4 (3 X 3) filters and the rest of params as above

   The decoder will consist of:

   - 1 **Conv2DTranspose** layer with the same parameters as the last layer of the encoder
   - 1 **Conv2DTranspose** layer with the same parameters as the first layer of the encoder
   - 1 **Conv2D** layer with 1 filter with sigmoid activation that will reconstruct the original image

3. Compile your model using **adam** optimizer and **mse** loss and fit it with your training data using **10 epochs** and a batch size of **64** (use the test data as validation data in the trainig process). Use only the original train data for training. Compute the reconstruction loss for the training data and a threshold (that will be the mean of the reconstruction errors + their standard deviation). Based on the threshold and the obtained reconstruction errors classify both the original test images and the ones that have the added noise (and compute the corresponding accuracy).

4. Plot in the same figure, on four rows, 5 test images: on the first row - the original ones, on the second one - the images with the added noise, on the third, the reconstructed images obtained from the original ones and on the last row the reconstructed images obtained from the images with added noise.

5. Modify the training stage in order to obtain a Denoising Autoencoder and print the same figure again.