Dimensionality reduction of large datasets with t-SNE and PCA

Ondrej Spetko, Anna Lunterova
Aalborg University Copenhagen

Introduction

The initial project idea is to use unsupervised machine learning in Python to analyze and cluster a big dataset, describing correlation between dimensions of high-dimensionality unlabeled data by visualizing it into low dimensional space. Two most common dimensionality reduction algorithms, PCA and t-SNE will be used, and compared. Primarily Scikit, Numpy, Matplotlib and Pandas data science libraries will be used for implementation. The final t-SNE data points will be visualized and coloured on a 3D scatter map for further exploration of dataset. Additionally, different parameters for t-SNE were compared and analyzed based on the resulting visualization. Firstly, the purpose behind used implementation technique and the two algorithms will be explained in the analysis. Then the structure of implementation will be described, and lastly, the results of this process will be shown.

Analysis

Dataset

Before starting, the chosen data collection needs to be gathered. The chosen was already collected dataset from USDA National Nutrient 2017 database, that consists of around 8400 data points, with 41 dimensions/features. The features consists of the calorie amount, amount of proteins, carbohydrates, fats, fiber, and of different minerals and vitamins, and recommended amount of those different minerals and vitamins per day. This data was downloaded in an already processed form of an excel sheet. The process of data cleaning was decided based on the purpose of the visualization and type of communicated information.

Python

Python is flexible, free, supported by large community and with a lot of libraries for scientific computing, data science and machine learning. It is a high-level programming language that allows to quickly write and prototype, however with slower performance compared to C++. Usually for the reason of higher performance Scikit library is used for machine learning computing. Among famous data science libraries belong Scikit, Numpy, Pandas and Matplotlib. These allow easier handling, analyzing, and visualizing data implementation. Famous environment for using Python is Jupyter Notebook. It is an open-source web application that allows to create and share Python execution environment with outputs in one document. Jupyter Notebook can be run using Anaconda environment, which includes commonly used tools and languages for scientific computing and data science. Next chapters explain the specific two data mining algorithms that will be used.

PCA

The most common algorithm for reducing dimension of a dataset is using of principal component analysis (PCA) developed already back in 1933. PCA is linear algorithm that uses an orthogonal transformation to transform a set of observations of likely similar variables into a so called principal components, what is a set of values of linearly uncorrelated variables. Problem with linear dimension reduction algorithms is that the dissimilar data points are being placed far apart in lower dimensional representations. Linear algorithms won't be able to describe complex polynomial relations between features. Unlike other famous non-linear dimensionality reduction algorithm t-SNE that uses probability distribution with element of randomness instead to find the relations in the data. This allows t-SNE model distances between points in the low-dimensional map

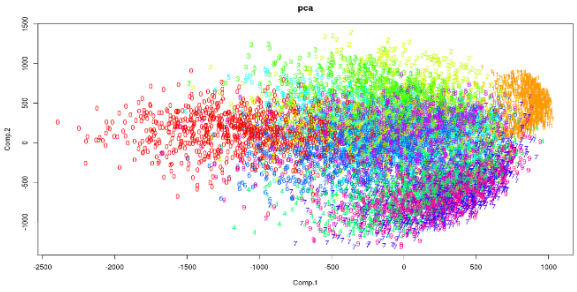

Figure 1. PCA dimensionality reduction visualization

t-SNE

t-SNE is a relatively new technique (2008) developed by Laurens van der Maaten that reduces amount of dimensions in high-dimensional data by assigning relative distance value based on the relative correlations to each datapoint resulting in a two or three-dimensional map. The strength of t-SNE is in preserving the local distances of the high-dimensional data in mapping to low-dimensional data. Even though the e-SNE results in obvious clustering of the similar data points in the final map, the algorithm is not a clustering algorithm and is used only as exploratory or visualization tool because original dimensions are mapped to lower dimensions without preserving any link to their original values. Because of that one can not make definite assumptions based only on t-SNE output. However, output of the t-SNE can be used in process of classification or clustering as input into further classification or clustering algorithms. t-SNE algorithm is quite heavy on the system resources because it compares the relations pairwise

with goal of minimizing the sum of the difference of the probabilities in higher and lower dimensions. The visualizations produced by t-SNE are found to be significantly more accurate compared to Principal Component Analysis (PCA), Sammon mapping, Isomap, Locally Linear Embedding and other visualization techniques algorithms.

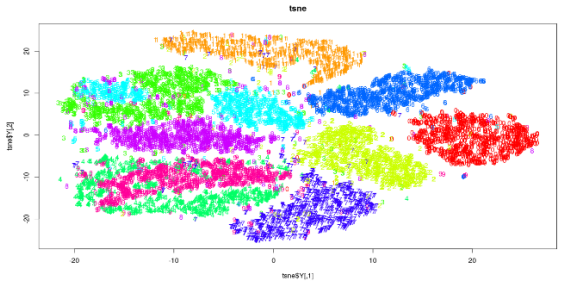

Figure 2. t-SNE dimensionality reduction visualization

When implementing t-SNE it is important to be aware of the parameters that are tuning the algorithm output. Below is table of the parameters that are available when implementing t-SNE with Python programming language.

n_components : int, optional (default: 2)	Dimension of the embedded space.
perplexity : float, optional (default: 30)	The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selcting a value between 5 and 50. The choice is notn extremely critical since t-SNE is quite insensitive to this parameter.
early_exaggeration : float, optional (default: 4.0)	Controls how tight natural clusters in the original space are in the embedded space and how much space will be between them. For larger values, the space between natural clusters will be larger in the embedded space. Again, the choice of this parameter is not very critical. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high.
learning_rate : float, optional (default: 1000)	The learning rate can be a critical parameter. It should be between 100 and 1000. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high. If the cost function gets stuck in a bad local minimum increasing the learning rate helps sometimes.
n_iter : int, optional (default: 1000)	Maximum number of iterations for the optimization. Should be at least 200.
metric : string or callable, (default: "euclidean")	The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter, or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS If metric is "precomputed", X is assumed to be a distance matrix. Alternatively, if

Figure 3. t-SNE parameters

Structure and design

The structure of the code was divided into main data analysis steps: * Pre-processing * Loading * Cleaning * Dividing of the dataset * Data analysis * t-SNE execution and modelling * Data visualization * Scatter plot

Jupyter notebook allows to write code in separate cells, that can be executed in any order, and display the results below the executed cell(See figure below of an example of separated reading cell)

Data reading

Firstly Numpy and Pandas libraries were loaded. Dataset in form of csv file was loaded, and saved as data frame. The list of columns it is composed from was printed together with

```
import numpy as np
import pylab
import pandas as pd
food=pd.read_csv("acro.csv")
food.columns
food=food[['ID', 'FoodGroup', 'Descrip',
       'Energy_kcal', 'Protein_g', 'Fat_g', 'Carb_g',
       'Sugar_g', 'Fiber_g', 'VitA_mcg', 'VitB6_mg', 'VitB12_mcg', 'VitC_mg',
       'VitE_mg', 'Folate_mcg', 'Niacin_mg', 'Riboflavin_mg', 'Thiamin_mg',
       'Calcium_mg', 'Copper_mcg', 'Iron_mg', 'Magnesium_mg', 'Manganese_mg',
       'Phosphorus_mg', 'Selenium_mcg', 'Zinc_mg', 'VitA_USRDA', 'VitB6_USRDA',
       'VitB12_USRDA', 'VitC_USRDA', 'VitE_USRDA', 'Folate_USRDA',
       'Niacin_USRDA', 'Riboflavin_USRDA', 'Thiamin_USRDA', 'Calcium_USRDA',
       'Copper_USRDA', 'Magnesium_USRDA', 'Phosphorus_USRDA', 'Selenium_USRDA',
       'Zinc_USRDA']]

food.shape
food.dropna()
#food.dtypes
#food.describe()
```

Figure 4. Example of a cell in Jupyter Notebook

the head and tail of the document (first and last 10 rows), to understand its dimensions better,see figure below.

	ID	FoodGroup	Descrip	Energy_kcal	Protein_g	Fat_g	Carb_g	Sugar_g	Fiber_g	VitA_mcg	...	Folate_USRDA	Niacin_USRDA	Riboflavin_USRD
0	1001	Dairy and Egg Products	Butter, salted	717.0	0.85	81.11	0.06	0.06	0.0	684.0	...	0.0075	0.002625	0.02615
1	1002	Dairy and Egg Products	Butter, whipped, with salt	717.0	0.85	81.11	0.06	0.06	0.0	684.0	...	0.0075	0.002625	0.02615
2	1003	Dairy and Egg Products	Butter oil, anhydrous	876.0	0.28	99.48	0.00	0.00	0.0	840.0	...	0.0000	0.000188	0.00384
3	1004	Dairy and Egg Products	Cheese, blue	353.0	21.40	28.74	2.34	0.50	0.0	198.0	...	0.0900	0.063500	0.29384
4	1005	Dairy and Egg Products	Cheese, brick	371.0	23.24	29.68	2.79	0.51	0.0	292.0	...	0.0500	0.007375	0.27002
		Dairy and												

Figure 5. Dataset structure of original document

Additionally, types of each columns were printed (float,int,objects,strings..), and the size of the data frame matrix.

Data cleaning

Then, from the column of food category, certain categories were decided not to be used, as they were causing noise in the resulting visualization mostly because these categories were complete meals. This is part of the "condensing" cell in the code. The columns with values of nutrients, carbohydrates, proteins, fats, fibers etc., were separated from the food ID, description and food category, into separated dataframes. This was done to prepare the dataset for further analysis with t-SNE and PCA, by dividing into a dataframe of only columns with necessary values.

Data analysis

Short analysis consisted of counting number of values in each category (food_desc["FoodGroup"].value_counts()), and calculating statistics, such as mean of the values or their range. For this, both Describe and Scipy statistics package were explored. This was done to understand standard deviations of values and differences between categories better (how far outliers might be etc.).

Separately, principal component analysis (PCA) was performed in separated cells, as part of the data analysis. The steps for doing principal component analysis can be found at the end of the code. The steps were in this order: Singular Vector Decomposition, then selecting eigenvalues and ranking them from highest to lowest (figure below).

Then cumulative sum was calculated, telling us how many percentages of the data similarity each of the principal components (represented by its eigenvalue) represents. Projection matrix was calculated, to prepare for the final visualization of

```
# ranking the eigenvalues from highest to lowest in order choose the top k eigenvectors
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])
```

```
Eigenvalues in descending order:
6.4303461127309225
4.053416599986892
3.345825758072278
2.9085411898022455
2.428615666142026
2.096371109978456
1.7193368686757668
1.3861624283670798
1.2819441202840078
1.1239333686860995
1.039317949160192
1.0364047942962367
0.9896392929901093
0.9579355239660062
0.9409745830236887
0.8802344162209781
```

Figure 6. Matrix eigenvalues

points by the two main principal components. Array on the figure below shows how many percentages of the data can be represented by the first eigenvalues.

```
[ 15.3273276   24.9770176   32.94219478  39.86635866  45.64799687
 50.63868333  54.73179031  58.03173183  61.08356799  63.75923883
 66.23347156  68.70076915  71.05673532  73.33722651  75.57733996
 77.67285347  79.63261472  81.4953571   83.30671559  84.89804173
 86.41175595  87.80416742  89.09257861  90.30601074  91.43219335
 92.47388378  93.43465963  94.30022791  95.12546855  95.87846028
 96.61417072  97.30559015  97.9119951   98.49744363  99.03905109
 99.4213317   99.75796003  99.89137739  99.97555173  99.98514004
 99.99385733 100.        ]
```

Figure 7. Array of cumulative summary in percentages

Since the datasets consists of so many dimensions, and all of them are very similarly significant, we can already predict that the visualization by principal component analysis will not be that precise. The first two principal components can represent only 25% of data.

Data visualizations

Visualizations were done by running sklearn manifold library function, TSNE and by implementing PCA from scratch to break it down into small steps. The already implemented function was used for TSNE as it is more complex algorithm, with changing default parameters, and exploring the results. Creation of both 2D and 3D t-SNE maps were created. In the created program structure, firstly we run TSNE function on the created dataframe (figure below).

```
#running tsne
from sklearn import manifold, datasets
from sklearn.manifold import TSNE
#sample=food_noname[0:1000]

model = manifold.TSNE(n_components=2,perplexity=60.0, random_state=0,n_iter=1000)
Y=model.fit_transform(food_noname)
```

Figure 8. Execution of t-SNE from Sklearn library

This created as an output a matrix Y of x and y values, for each point in the space. After, Y was plotted and visualized into uniform color map, where the structure of the output points could be already seen (Figure below).

Colorization of the points in the map, or projection into colors was done in two ways. Firstly by their distance in the space by using PCA. Dimensions we compared to the median of each, and using those comparisons as bits in a 24-bit color (see the code cell bellow).

```
#plotting function
def plot_tsne(xy, colors=None, alpha=0.9, figsize=(90,90), s=70, cmap='hsv'):
    plt.figure(figsize=figsize, facecolor='white')
    plt.margins(0)
    plt.axis('off')
    fig = plt.scatter(xy[:,0], xy[:,1],
                      c=colors, # set colors of markers
                      cmap=cmap, # set color map of markers
                      alpha=alpha, # set alpha of markers
                      #marker=',', # use smallest available marker (square)
                      s=s, # set marker size. single pixel is 0.5 on retina, 1.0 otherwise
                      lw=0, # don't use edges
                      edgecolor='') # don't use edges
    # remove all axes and whitespace / borders
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
    plt.show()
```

Figure 9. Plotting the resulting Y matrix into a 2D space

```
#Colouring. PCA to 24 dimensions, comparing the dimensions to the median of each, and using those comparisons
#as bits in a 24-bit color.
from sklearn.decomposition import IncrementalPCA
def projection_to_colors(projection, bits_per_channel=8):
    basis = 2**np.arange(bits_per_channel)[::-1]
    basis = np.hstack([basis, basis, basis])
    shuffled = np.hstack([projection[:,0::3], projection[:,1::3], projection[:,2::3]])
    bits = (shuffled > np.median(shuffled, axis=0)) * basis
    # if we stacked into a 3d tensor we could do this a little more efficiently
    colors = np.vstack([bits[:,:bits_per_channel].sum(axis=1),
                        bits[:,bits_per_channel:(2*bits_per_channel)].sum(axis=1),
                        bits[:,(2*bits_per_channel):(3*bits_per_channel)].sum(axis=1)]).astype(float).T
    return colors / (2**bits_per_channel - 1)

def pack_binary_pca(data, bits_per_channel=8):
    bits_per_color = 3 * bits_per_channel
    pca = IncrementalPCA(n_components=bits_per_color)
    pca_projection = pca.fit_transform(data)
    return projection_to_colors(pca_projection, bits_per_channel)

%time colors = pack_binary_pca(food_noname, 8)
plot_tsne(Y, colors)
```

Figure 10. Projection of points into 24 colors

Secondly, the colorization was done by assigning the colors to specific food groups, to see if there is any correlation between points in the same food category to be close to each other. This was chosen at the end as final way of coloring, and the result will be seen and analyzed in the behaviour chapter. The figure below shows labels for the assigned colors to specific food groups.

Figure 11. Legend of categories

Lastly, 3D interactive map was implemented with the help of plotly library. To do that, in the t-SNE parameters the number of components was increased to 3. This way, the Y matrix had 3 dimensions. This was loaded and visualized in the scatter plot map (figure below).

The output of this is a 3D interactive map, food points colored by the categories, and with names of the categories visualized on hover. This allows us or the user to see the distribution of different categories in space, different clusters, etc. The final outputs and visualizations of those two algorithms will be described in the next chapter.

BEHAVIOUR AND RESULTS

Dimensionality reduction with PCA

After calculating the projection matrix in the PCA cells, this was visualized using seaborn library. The colors were

```
import plotly as py
import plotly.graph_objs as go
x=Y[:, 0]
y=Y[:, 1]
z=Y[:, 2]
trace1 = go.Scatter3d(
    x=x,
    y=y,
    z=z,
    text = catNames,
    hoverinfo = 'text',
    mode='markers',
    marker=dict(
        size=3,
        color=catNumbers,                # set color to an array/list of desired values
        colorscale='Viridis',   # choose a colorscale
        opacity=0.8
    )
)

data = [trace1]
layout = go.Layout(
    margin=dict(
        l=0,
        r=0,
        b=0,
        t=0))
fig = go.Figure(data=data, layout=layout)
#py.iplot(fig, filename='3d-scatter-colorscale')
py.offline.plot(fig, filename = 'trial.html')
```

Figure 12. 3D scatter plot implementation

assigned by food categories. Since the visualization is characterized by only first two principal components, that represents only 24%, we could see the points are still quite indistinguishable and clustered together (figure below).

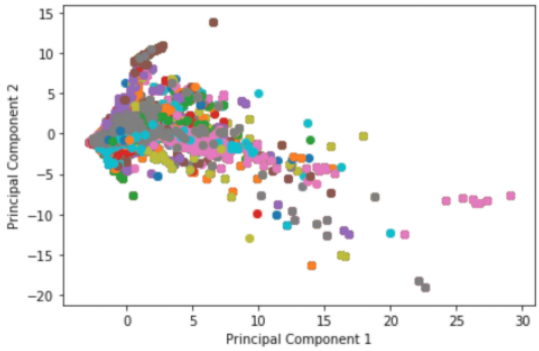

Figure 13. PCA visualization. Result of projection of main components onto the points

Dimensionality reduction with t-SNE

The t-SNE was run multiple times, with different parameters of perplexity and number of iterations. Figure below shows the perplexity of 10, 30, 50, and 90. The number of iterations is 1000 on all four samples.

Figure 14. t-SNE dimensionality reduction, visualizations comparison

The higher the number of perplexity and iterations, the longer the processing took. The higher the perplexity the more clearly divided points. The number can be between 1-100, and the closer to 100 the less changes. The number of iterations changed the look only until the number around 1000 was reached, then the changes were small. For the final 2D visualization perplexity 90 and 4000 iterations were used. The two figures below compare the outcome of a map when colorization by food categories, with PCA to 24 dimensions.

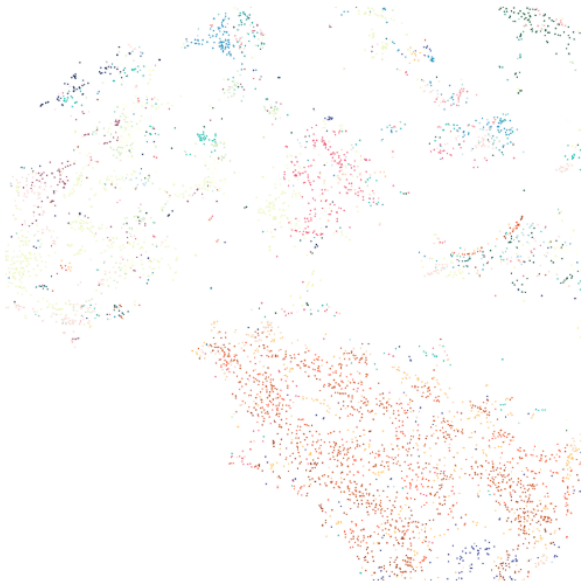

Figure 15. Final 2D t-SNE map colored by food categories

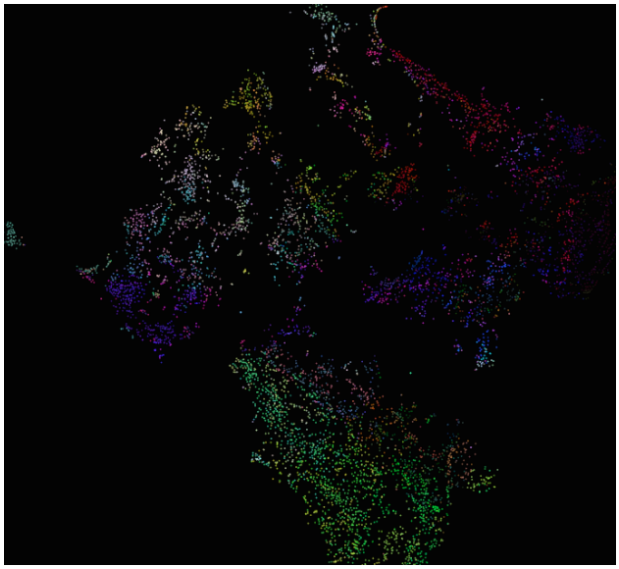

Figure 16. Final 2D t-SNE map colored by PCA distances

On both, the perplexity is 90 and 4000 iterations, only the way to color the points was different. We can see some similarities, such as on the food category the meat points are mostly clustered together on the bottom of the map, and by distance they are also colored in similar shades. Similarly, fruit is more on the left side, etc. Further, to interact with the points and see them on a 3D map, an interactive exploratory map with labels can be found here https://shourikan.github.io/TrialViso/, and figure from it seen on the figure below.

At the end Interactive 2D map, with perplexity 90 and 4000 iterations was further chosen as a best representation of the dataset. The final implementation was part of a bigger project, and can be seen here.

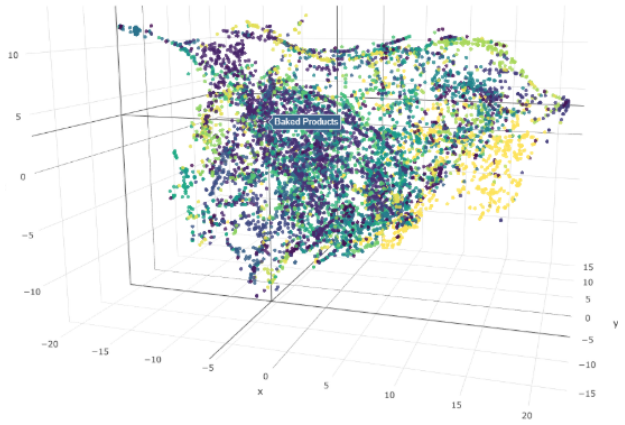

Figure 17. Interactive 3D t-SNE map

CONCLUSION

This research lead us to see the benefits of dimensionality reduction with t-SNE algorithm, however PCA served as a great tool for analyzing the principal components. In case of high correlation of only few dimensions and low correlation of others, PCA is a great and quicker tool to implement. However, in case of food database, all dimensions were more or less equally significant in localizing the point. By coloring the points we could see the closeness of points in same categories or similar categories such as beef and pork. This fulfilled the purpose of this program, and chosen visualization served as a base for further communication of the dataset in a form of an application.