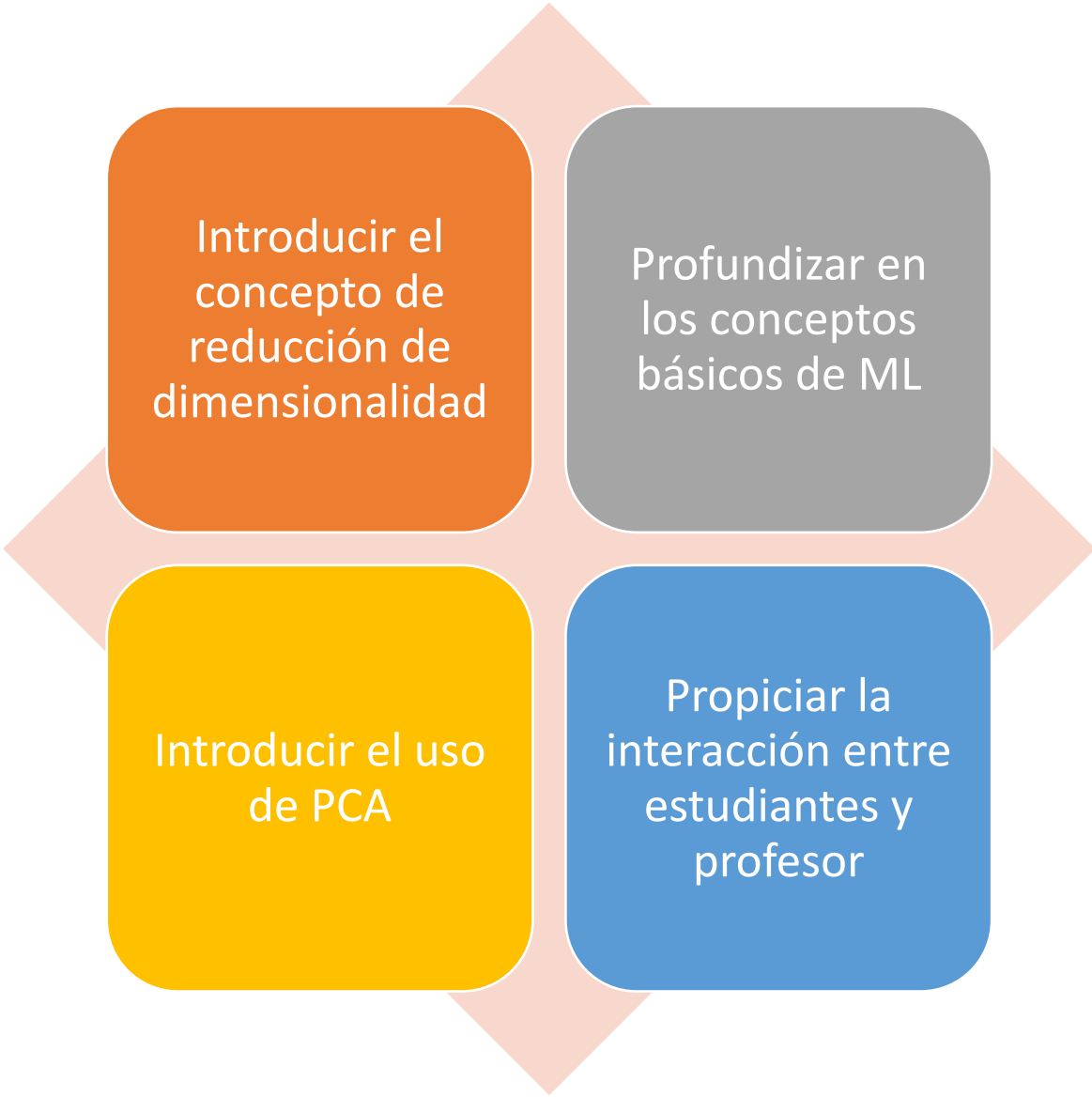
Inteligencia Artificial LFIS419

Clase 6.2: Reducción de dimensionalidad

Profesor: Jorge Arevalo (jorge.arevalo@uv.cl jab@meteo.uv.cl)

Martes 2 de mayo de 2023

Objetivos de la sesión



Problema

- Análisis de datos multivariados es importante
- Espacios con más de 3 dimensiones son difíciles de visualizar

- Se busca representar los datos de una manera que facilite el análisis

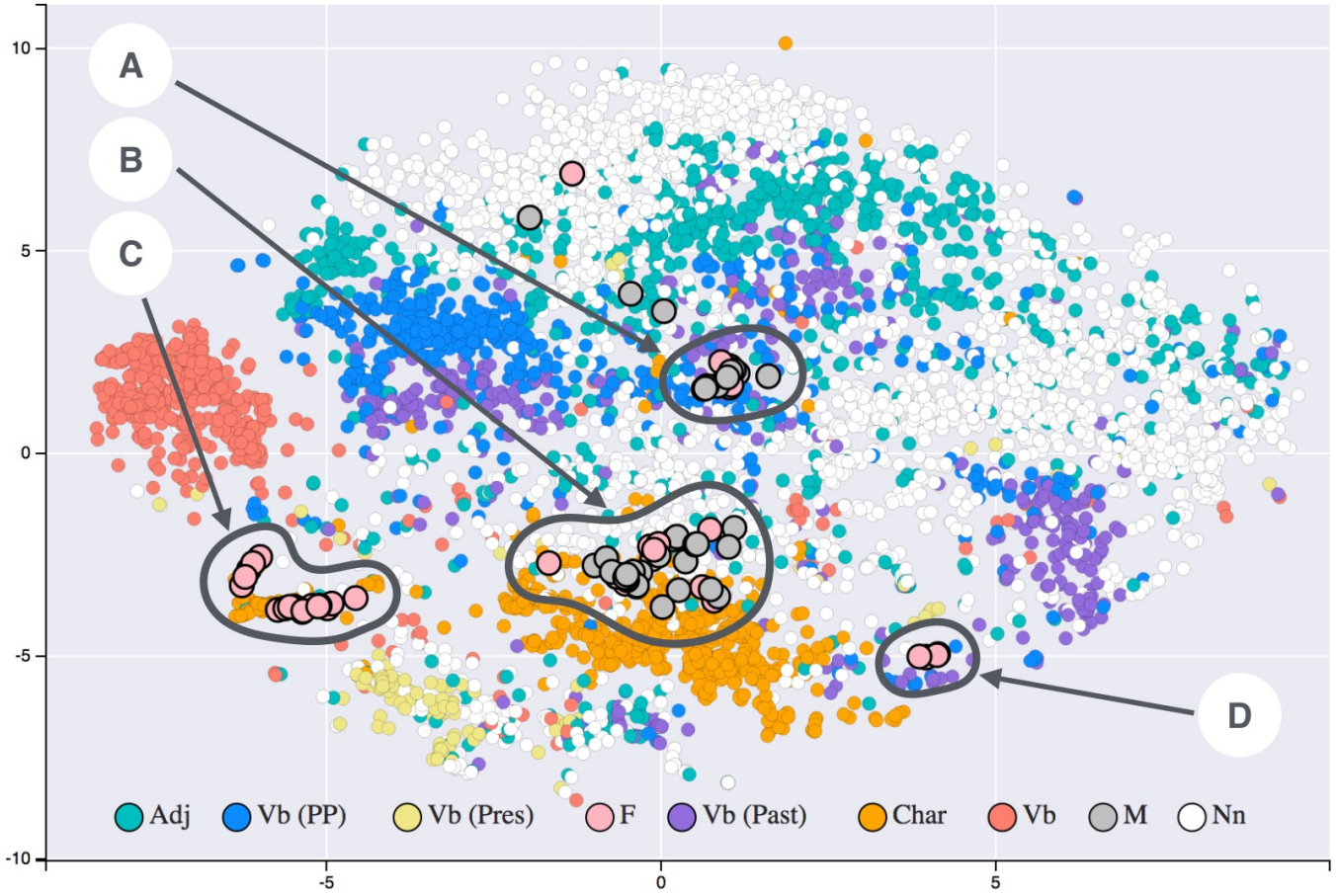
Beneficios

- Se reduce la carga computacional en las siguientes etapas
- Podría reducirse el ruido
- Proyectar los datos en un espacio de dimension pequeña es útil para visualizar los datos

t-SNE

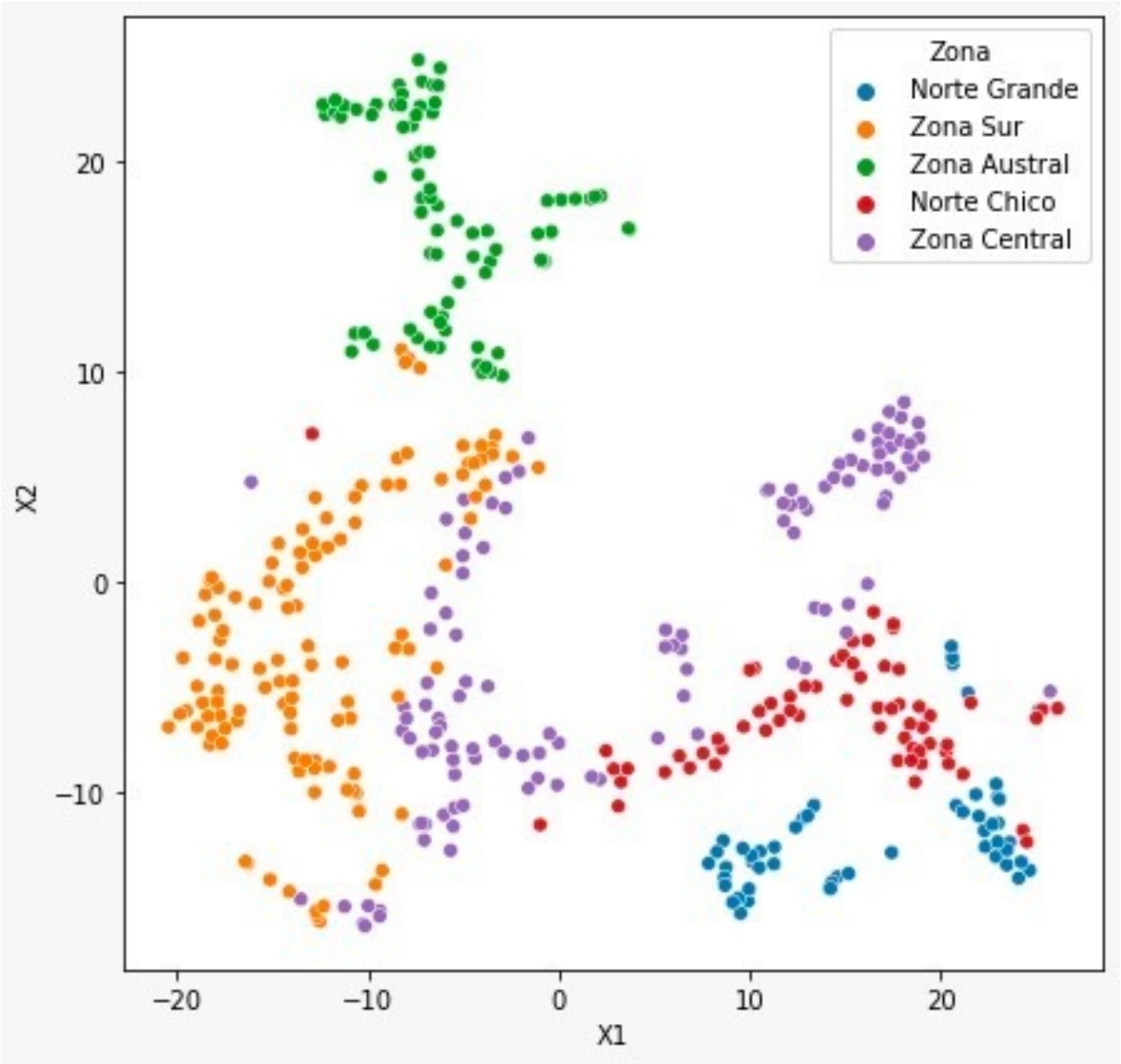
- t-distributed stochastic neighbor embedding
- 2D o 3D
- Primero calcula una distribución de probabilidad para pares de objetos en la dimensión original, donde objetos similares (distancia euclidiana) tienen alta probabilidad y objetos diferentes tienen baja probabilidad
- Luego genera una nueva distribución en los pares de objetos de baja dimensión (2D o 3D) y minimiza la diferencia entre ambas distribuciones.

t-SNE



https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

t-SNE


Zona
Norte Grande
Zona Sur
Zona Austral
Norte Chico
Zona Central
20
10
0
−10
X2
−20
−10
0
10
20
X1

t-SNE

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal width','target'])

df.head()
X = df.iloc[:,0:4]
y = df.iloc[:,4]
X = StandardScaler().fit_transform(X)
```

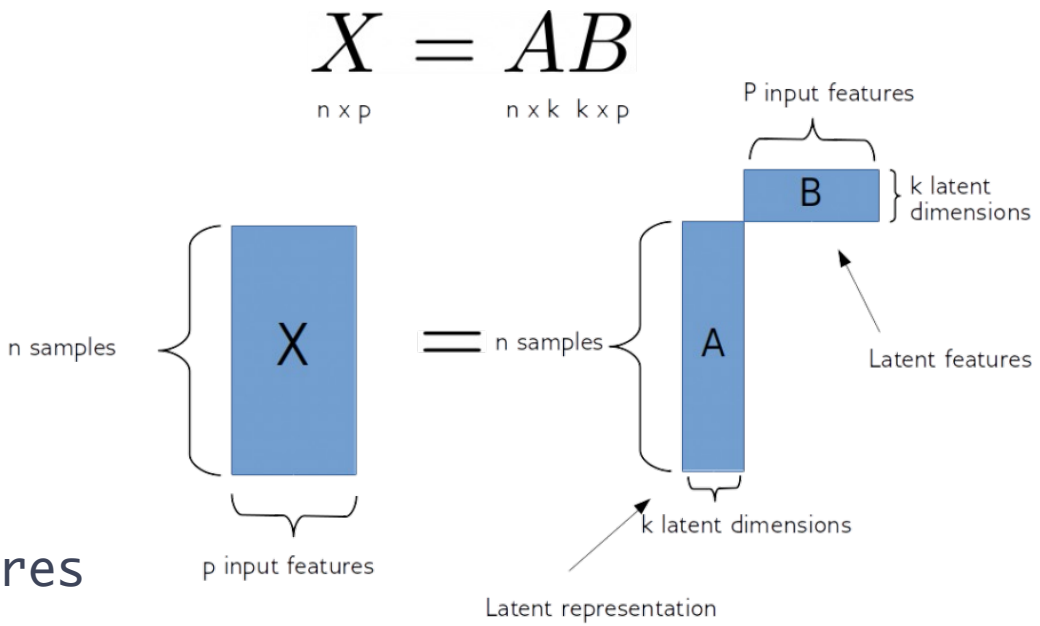
```
# the two components
tSNE =
pd.DataFrame(TSNE(n_components=2).fit_transform(X),
columns = ['tSNE1', 'tSNE2'])

# add the target
tSNE['target'] = y

sns.scatterplot(x='tSNE1', y='tSNE2', data=tSNE,
hue='target')
```

https://predictivehacks.com/dimension-reduction-in-python/

Non-Negative Matrix Factorization (NMF)

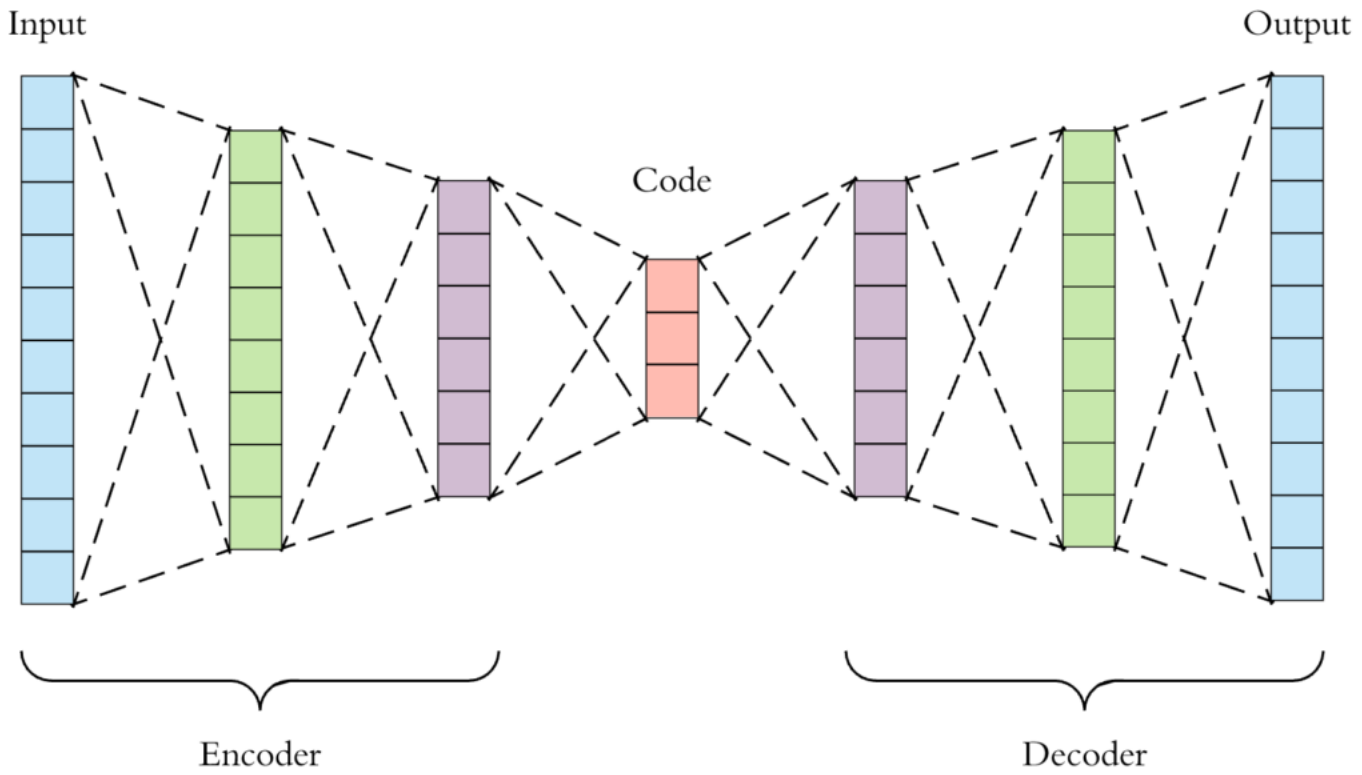
- Similar a PCA, pero coeficientes son positivos
- Útil cuando se requiere determinar factores aditivos

```
# Import NMF
from sklearn.decomposition import NMF

# Create an NMF instance: model
model = NMF(n_components=2)

# Fit the model to televote_Rank
model.fit(televote_Rank)

# Transform the televote_Rank: nmf_features
nmf_features =
model.transform(televote_Rank)
```



https://predictivehacks.com/non-negative-matrix-factorization-for-dimensionality-reduction/

Autoencoders



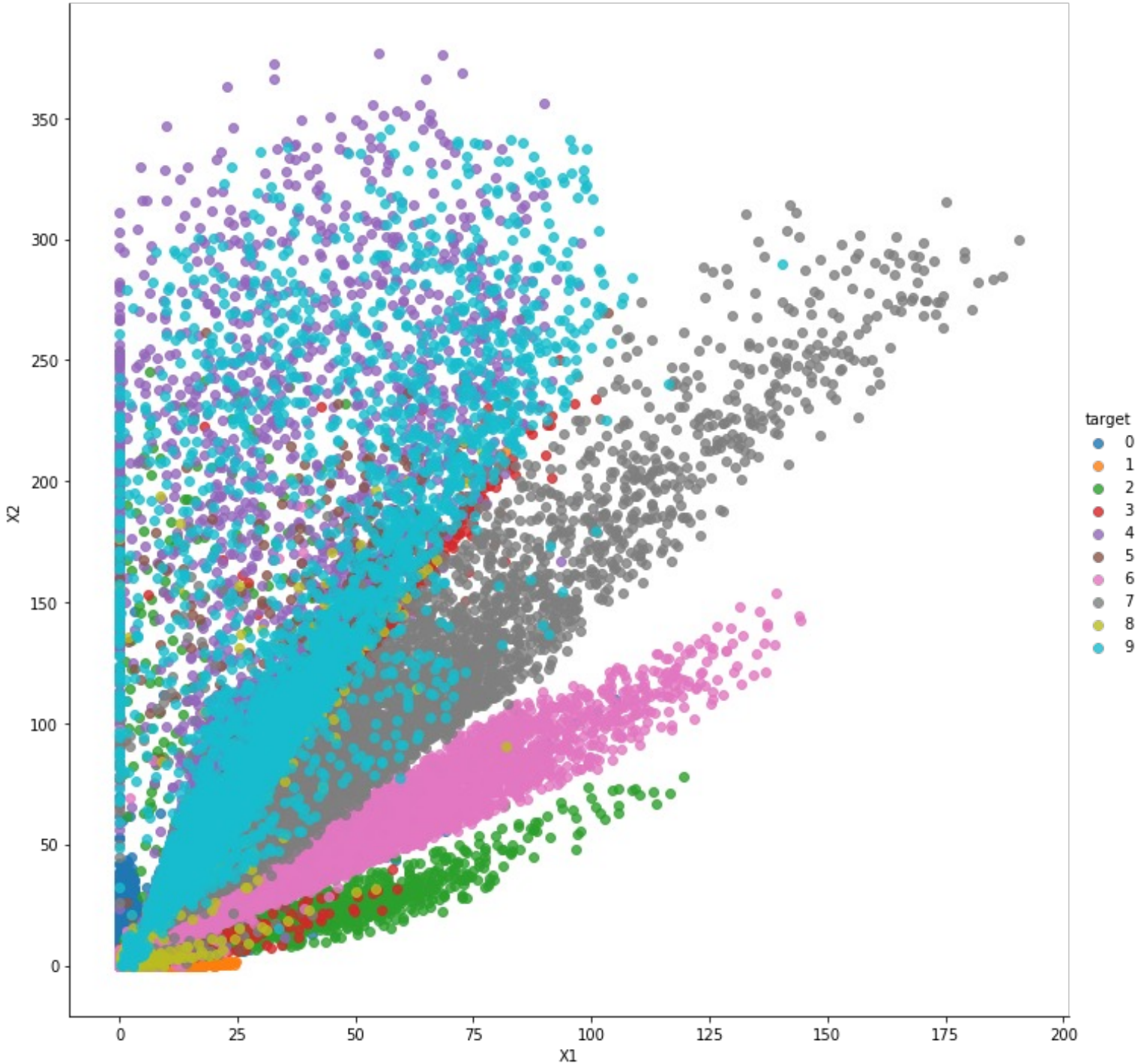
https://predictivehacks.com/autoencoders-for-dimensionality-reduction/

Autoencoders

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten,Reshape
from tensorflow.keras.optimizers import SGD


from tensorflow.keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train/255.0
X_test = X_test/255.0
```

```
### Encoder
encoder = Sequential()
encoder.add(Flatten(input_shape=[28,28]))
encoder.add(Dense(400,activation="relu"))
encoder.add(Dense(200,activation="relu"))
encoder.add(Dense(100,activation="relu"))
encoder.add(Dense(50,activation="relu"))
encoder.add(Dense(2,activation="relu"))


### Decoder
decoder = Sequential()
decoder.add(Dense(50,input_shape=[2],activation='relu')
)
decoder.add(Dense(100,activation='relu'))
decoder.add(Dense(200,activation='relu'))
decoder.add(Dense(400,activation='relu'))
decoder.add(Dense(28 * 28, activation="relu"))
decoder.add(Reshape([28, 28]))

### Autoencoder
autoencoder = Sequential([encoder,decoder])
autoencoder.compile(loss="mse")
autoencoder.fit(X_train,X_train,epochs=50)
```

```
encoded_2dim = encoder.predict(X_train)


# The 2D
AE = pd.DataFrame(encoded_2dim, columns = ['X1', 'X2'])

AE['target'] = y_train

sns.lmplot(x='X1', y='X2', data=AE, hue='target', fit_reg=False,
size=10)
```


X2
X1
target
0
1
2
3
4
5
6
7
8
9
0
50
100
150
200
250
300
350
0
25
50
75
100
125
150
175
200