PROCESSAMENTO DE DADOS: Redução de dimensionalidade para melhoria da eficiência e precisão de sistemas de reconhecimento de padrões

Marcelo de Medeiros

***Abstract*—Neste artigo foi empregado métodos de redução de dimensionalidade e métodos de classificação de dados para analisar quais seriam os melhores resultados para os bancos de dados propostos, sendo eles o Digits e o Mfeat-pixel. Foi feito a redução de dimensionalidade com 7 métodos para 20D e aplicado 10 métodos de classificação para cada uma das reduções. Como resultado, foram obtidos tabelas de acurácias para reduções 20D e 2D, em que foi analisado que o melhor método de redução foi o t-SNE e o melhor método de classificação neste método foi SVM com kernel rbf.**

Keywords: Redução de Dimensionalidade, Classificação de Dados, t-SNE e Reconhecimento de Padrões.

I. Introdução

O reconhecimento de padrões é a ciência que trata de classificar e descrever dados. A classificação de padrão pode ser feita de duas maneiras, sendo elas supervisionada e não supervisionada. Na supervisionada quem está tratando dos dados já espera certas classes de entrada para determinados dados. Na não supervisionada o próprio classificador tem o trabalho de identificar e categorizar.

A correta classificação e categorização dos dados são partes essenciais no pré-processamento dos dados para uma correta tomada de decisão visando esses classificadores.

A sua aplicação é a mais abrangente possível, sendo muitas vezes indispensáveis para a utilização de inteligencia artificial ou até mesmo pequenas automações. Um exemplo que pode ser citado é os operadores de telefonia automatizados, que trabalham no reconhecimento de voz para então abrir um procedimento, sem interferência de operador humano, sendo as entradas de dados a voz em forma de onda, e o classificador a palavras faladas.

Outro exemplo é o reconhecimento de letras escritas manualmente feita por aprendizado de máquina. No qual o banco de dados deste classificador está repleto de dados para poder fazer a melhor identificação possível para os novos dados que chegam nesta máquina.

II. Métodos Para Redução de Dimensionalidade

A. Análise de Componentes Principais (PCA)

É uma técnica de estatística que busca os principais componentes que caracterizam a classe, e faz redução de dimensionalidade preservando esses elementos, sendo uma método supervisionado.

Ele inicia lendo os dados de entrada $X = \vec{x}_1, \vec{x}_2, ..., \vec{x}_n$ com $\vec{x}_i \in R^d$. Computa o vetor média μ_x e a matriz de covariância dada por:

$$\Sigma_x = \frac{1}{n}\sum_{i=1}^{n}(\vec{x}_i - \vec{\mu}_x)(\vec{x}_i - \vec{\mu}_x)^T \quad (1)$$

Para obter o autovalores e autovetores de Σ_x. Selecionando os k autovetores associados aos k maiores autovalores da matriz de covariâncias, expressa por $\vec{w}_1, \vec{w}_2, ..., \vec{w}_k$. Deste modo então definindo a matrix de transformação $W_{PCA} = [\vec{w}_1, \vec{w}_2, ..., \vec{w}_k]$. Para em seguida poder projetar a nova base de dados reduzida a partir da equação 2 :

$$\vec{y} = W_{PCA}^T \vec{x}_i \quad (2)$$

Com os autovetores nas linhas W_{PCA}^T.

B. Linear discriminant analysis (LDA)

No LDA a ideia principal é maximizar a separabilidade entre as classes, sendo este método supervisionado.

Este método começa verificando os dados de entrada $X = \vec{x}_1, \vec{x}_2, ..., \vec{x}_n$ com $\vec{x}_i \in R^d$ e computando os vetores médias μ_j de cada classe, j=1,...,C. Logo após ele fará a computação de 3 matrizes, de espalhamento de cada classe, expressa na equação 03, de espalhamento intra-classes, expressa na equação 04 e espalhamento entre-classes, expressa na equação 05:

$$S_j = \sum_{\vec{x}_i \in \omega_j} (\vec{x}_i - \vec{\mu}_x)(\vec{x}_i - \vec{\mu}_x)^T \quad (3)$$

Sendo j o numero de classes q vai de 1 até C.

$$S_w = S_1 + S_2 + ... + S_C \quad (4)$$

$$S_B = \sum_{i=1}^{C} n_i(\vec{\mu_i} - \mu)(\vec{\mu_i} - \vec{\mu})^{\vec{T}} (5)$$

Para obter os autovalores e autovetores de $S_W^{-1}S_B$, então escolhendo os k autovetores associados aos k maiores autovalores, expressos por $\vec{w}_)1, \vec{w}_2, ..., \vec{w}_k$. Para que então se possa definir a matriz de transformação $W_{LDA} = [\vec{w}_)1, \vec{w}_2, ..., \vec{w}_k]$, para aplicar a projeção dos dados na nova base, expressa por:

$$\vec{y} = W_{LDA}^T \vec{x}_i \tag{6}$$

Com autovalores nas linhas de W_{LDA}^T.

C. Kernel PCA

Este é um método supervisionado, que tem como a ideia principal, fazer as análises de componentes principais para uma redução utilizando estrutura com a presença de não linearidade.

Inicia-se com a construção de uma estruturando uma matriz de kernel k para os dados de treinamento X : $K_{i,j} = K(\vec{x}_i, \vec{x}_j)$, para criar uma matriz K' com uso da seguinte expressão:

$$K' = K - 1_n K - K 1_n + 1_n K 1_n \tag{7}$$

E com o uso da equação 7 é possivel resolver os vetores de $\vec{\alpha}_k$

$$K\vec{\alpha}_k = (\lambda_k n)\vec{\alpha}_k \tag{8}$$

Por fim, computando o principal componente kernel $y_k(\vec{x})$ para $k = 1, 2, ..., d$, através da equação 8:

$$y_k(\vec{x} = \phi(\vec{x})^T \vec{v}_k = \sum_{i=1}^{n} \alpha_{ki} \phi(\vec{x})^T \phi(\vec{x}_i) = \sum_{i=1}^{n} \alpha_{ki} K(\vec{x}, \vec{x}_i) \tag{9}$$

D. ISOMAP

Este método de redução começa construindo um grafo unindo os vizinhos mais próximos, computando os menores caminhos entre cada par de vértices, conhecendo as distâncias entre os pontos, visando encontrar um mapeamento para o plano que preserve as distâncias.

Inicia-se produzindo um grafo a partir dos dados $\vec{x}_i, y_i$ para $i = 1, 2, ..., n$ onde $\vec{x}_i$ é o vetor das características que representa a i-ésima amosta e y_i a classe deste vetor.

A matriz de distâncias ponto a ponto D para cada amostra $\vec{x_i}$ do conjunto é montada. Aplicando o algoritmo Dijkstra na intenção de obter os menores caminhos de $\vec{x_i}$ aos demais. E busca D_ij ser o valor igual ao tamanho do menor caminho entre $\vec{x_i}$ e $\vec{x_j}$.

Com a matriz D montada no subespaço Euclidiano R^k, encontra-se um conjunto de pontos, de modo que as distancias sejam preservadas.

E. Local Linear Embedding (LLE)

Este é um método de redução de dimensionalidade não supervisionado e não linear. Que inicia indiciando um grafo a partir dos dados de entrada $\vec{x_i}$, computando a distancia entre ele e seus vizinhos $\vec{x_j}$ para encontrar os k menores de distância entre eles. Deste modo pode-se expressar a geometria local por coeficientes lineares da seguinte forma:

$$\vec{x}_i \approx \sum_j w_{ij} \vec{x}_j \tag{10}$$

Com isso é possível uma reconstrução de vetor para combinação linear de vizinhos para $\vec{x_j} \in N(\vec{x_i})$. Disto se faz necessário obter os pesos W com a utilização do método a tentar minimizar a soma dos erros quadráticos desta reconstrução, expressa por:

$$E(W) = \sum_i |\vec{x_i} - \sum_j w_{ij} \vec{x_j}|^2 \tag{11}$$

Em seguida, obtém-se uma matriz Z com todos os vizinhos de $\vec{x}$ nas colunas para os subtrair de toda coluna de Z, para então computar uma covariância local $C = Z^T Z$, e resolver um sistema linear $C = \vec{w} = \vec{1}$ sendo um vetor de coluna de 1's, para zerar todos os coeficientes w_j tais que $\vec{x_i}$ não seja vizinho de $\vec{x_j}$. Então é feita a normalização dos coeficientes da seguinte forma:

$$w_j = \frac{w_j}{\sum w_j} \tag{12}$$

Agora é feita a computação das coordenadas de imersão com a criação de uma matriz $M = (I - W)^T (I - W)$ encontra-se nela os k autovetores associados aos k autovalores não nulos de M. Chegando a equação:

$$\phi(Y) = Tr[Y^T M Y] \tag{13}$$

Chega-se então a um problema de otimização, sendo essencial uma solução aplicando multiplicadores de Lagrange, derivando em relação a Y, sendo a matriz M simétrica, deste modo chegamos a equação (5):

$$\arg\min_Y \mathrm{Tr}[Y^T M Y] = \arg\min_Y \mathrm{Tr}[\lambda Y Y^T] = \tag{14}$$

$$= \arg\min_Y \mathrm{Tr}[\lambda I] = \sum_i \lambda_i \tag{15}$$

Por fim, de modo a minimizar a soma dos autovalores, deve-se escolher apara compor a matriz Y_{nXk} os k autovetores associados aos k menores autovalores não nulos da matriz M.

F. Laplacian Eigenmaps

Esta é uma técnica de redução de dimensionalidade não linear, que aproxima os pontos próximos na variedade que devem permanecer próximos no espaço Euclidiano.

No qual é dado um conjunto de amostras $\vec{y_i} \in R^d$ sendo $i = 1, 2, ..., n$ para induzir um grafo G (matriz A). Por sua vez, é feito a computação da matriz Laplaciana $L = D - A$ e normalizada $L = D^{-1} L$.

Sendo possível fazer uma decomposição espectral de L com o intuito de obter os k autovetores associados aos k menores autovalores não nulo e montar a matriz F_{nXk} com autovetores nas colunas.

Utilizando multiplicadores de Lagrange e derivando e igualando a zero, chega-se a equação:

$$L\vec{f} = \lambda D\vec{f} \tag{16}$$

Com isso, de modo a evitar o mapeamento constante, tem-se $\vec{f} = \vec{v}_{n-1}$ com $\lambda = \vec{\mu}_{n-1}$. Portanto, pode-se dizer que quanto menor μ_{n-1}, melhor o mapeamento obtido e a solução f é o *Fiedler vector* e o mínimo valor obtido para $J(\vec{f})$ é a conectividade algébrica de G.

G. T-Distributed Stochastic Neighbour Embedding (t-SNE)

É um método de redução de dados pertencente a um grupo de técnicas não paramétricas. Que começa convertendo a distancia Euclidiana entre os pontos $\vec{x} \in R^m$ para $i = 1, 2, ..., n$ em probabilidades condicionais dado uma similaridade entre dois pontos $\vec{x_i}$ e $\vec{x_j}$ cuja probabilidade condicional $p_{j|i}$ de que $\vec{x_i}$ escolheria $\vec{x_j}$ como vizinho. Da seguinte forma:

$$p_{j|i} = \frac{\exp(-||\vec{x}_i - \vec{x}_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||\vec{x}_i - \vec{x}_k||^2/2\sigma_i^2)} \tag{17}$$

Em seguida é definido os valores de p_{ij} como :

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{18}$$

Com uma amostra de solução inicial onde $Y^{(0)} = \{\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_n\}$ para $\mathcal{N}(0, 10^{-4}\mathbf{I})$ com o intuito de calcular afinidades dimensionais baixas q_{ij} da seguinte maneira:

$$q_{ij} = \frac{1 + ||\vec{y}_i - \vec{y}_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||\vec{y}_k - \vec{y}_l||^2)^{-1}} = \frac{w_{ij}^{-1}}{\sum_{k \neq l} w_{kl}^{-1}} = \frac{w_{ij}^{-1}}{Z} \tag{19}$$

Na equação seguinte é então possível calcular o gradiente:

$$\frac{\delta C}{\delta \vec{y}_i} = 4 \sum_{j=1}^{n} (p_{ij} - qij) w_{ij}^{-1} (\vec{y}_i - \vec{y}_j) = \tag{20}$$

$$= 4 \sum_{j=1}^{n} (p_{ij} - qij)(1 + ||\vec{y}_i - \vec{y}_j||^2)^{-1} (\vec{y}_i - \vec{y}_j) \tag{21}$$

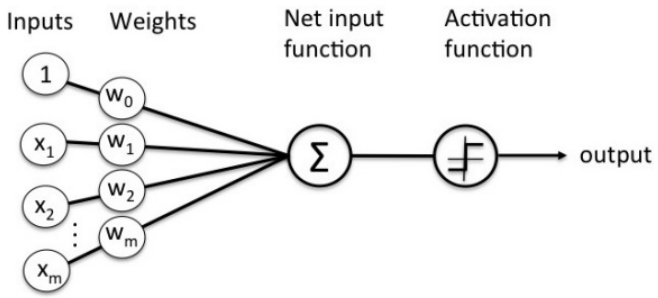
E por fim atualiza as coordenadas com a descida do gradiente por impulso:

$$Y^{(t)} = Y^{(t-1)} - \eta \frac{\delta C}{\delta Y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)}) \tag{22}$$

III. Métodos Utilizados Para Classificação

A. Perceptron

O seu proposito é que ele crie um algoritmo que aprende a cada interação com uso de pesos. Seu modo de trabalho é fazer multiplicação de cada peso pela sua respectiva entrada e acumular. Em seguida, esse valor é passado para a função de ativação, sendo ela de classificação binária. A Fig. 1 demonstra a sua arquitetura.



Schematic of Rosenblatt's perceptron.

Fig. 1: arquitetura básica de um neurônio artificial do tipo perceptron. Fonte: Introdução ao Reconhecimento de Padrões - Prof. Dr. Alexandre Luís Magalhães Levada

Como pode ser visto na Fig. 1 dada as entradas x_1,x_2...x_m e os pesos w_1,w_2...w_m é feito um calculo matemático e encontrado uma função de ativação binária para uma determinada saída. O uso do peso w_0=-θ e da entrada com valor x_0=1 que são definidos para simplificar a notação da equação (14):

$$g(z) = \begin{cases} +1, & \text{se } z \geq 0 \\ -1, & \text{se } z < 0 \end{cases} \tag{23}$$

Assim, chega-se a seguinte equação:

$$z = \sum_{i=0}^{k} w_i x_i = \overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}} \tag{24}$$

Com isso a equação para a atualização dos pesos é dada por:

$$\overrightarrow{w} = \overrightarrow{w} + \alpha(\overrightarrow{d_i} - y_i)\overrightarrow{x_i} \tag{25}$$

Onde $\alpha \in [0, 1]$ é a taxa de aprendizado, a saída do neurônio é $y_i = g(\overrightarrow{x_i})$ e d_i é a classificação esperada, sendo +1 ou -1.

Por fim, o algoritmo do perceptron modifica o hiperplano separador até que ele chegue a uma ponto de convergência. A Fig. 2 ilustra um hiperplano separador para 2 classes de entradas.

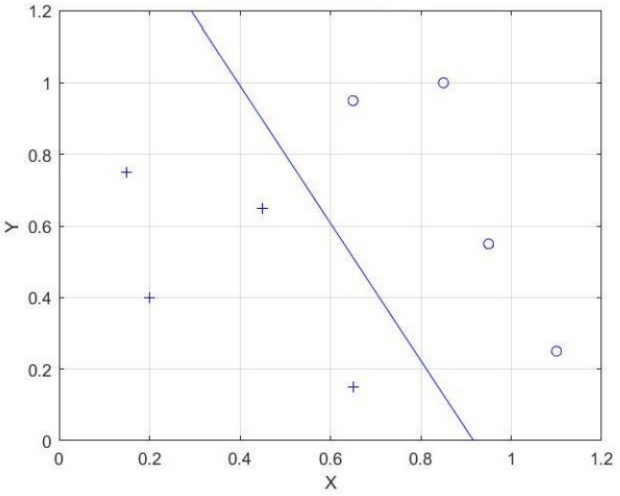


Fig. 2: Hiperplano separador. Fonte: Acervo pessoal

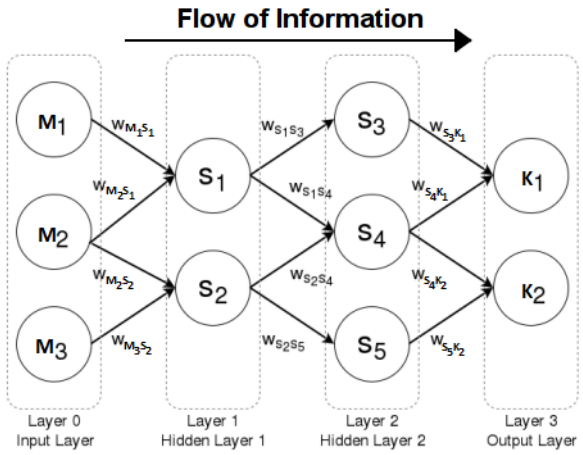


Fig. 3: Multicamadas de Perceptron do tipo feed-forward. Fonte: https://brilliant.org/wiki/feedforward-neural-networks/

B. Regressão Logística

Com o objetivo de classificar ou categorizar um vetor em observação de variáveis discreta, a regressão logística é uma adaptação da regressão linear. O seu modelo de ajuste para a saída é sempre limitado ($0 \leq h_\theta(x) \leq 1$). Para ajustes no modelo, é necessário modificar a hipótese $h_\theta(x)$ com uma função não linear, também conhecida como função sigmoide.

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{26}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{27}$$

Adotando a convecção de que $x_0 = 1$, tem-se :

$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j \tag{28}$$

Onde $g(z)$ tende a 1 quando $z \to \infty$ e g(z) tende a 0 quando $z \to -\infty$ para $g(z)$ e $h_\theta(x)$ limitados entre 0 e 1. De modo que é possível chegar a uma equação compacta de probabilidade para variáveis aleatórias binárias :

$$p(y|x;\theta) = [h_\theta(x)]^y [1 - h_\theta(x)]^{1-y} \tag{29}$$

C. Multilayer Perceptron (MLP)

Com o objetivo de generalizar o modelo do Perceptron, foi criado um modelo de rede de multicamadas. Na Fig. 3 podemos observar o modelo de multicamadas do tipo *feed-forward*.

Dado uma camada com M neurônios de entradas, uma ou mais camadas intermediarias e uma camada de saída com K neurônios. Com o conjunto de treinamento dado por:

$$T = \{\vec{x}(n), \vec{d}(n)\}_{n=1}^{N} \tag{30}$$

Onde $\vec{x}(n) \in R^M$ representa um vetor de padrões e $d(n) \in R^K$ a saída desejada de cada padrão de entrada. Sendo $y_j(n)$ o sinal na saída do neurônio j na ultima camada pela entrada $x(n)$ aplicada a camada de entrada. Logo será produzido o sinal de erro pelo neurônio de saída j dado por:

$$e_j(n) = d_j(n) - y_i(n) \tag{31}$$

Onde $d_j(n)$ é j-ésima componente do vetor de resposta desejada $d(n)$. Com isso, o erro instantâneo do neurônio j da camada de saída seguindo a terminologia adotada de Mínimos Quadrados Médios (LMS) é :

$$\epsilon_j(n) = \frac{1}{2} e_j^2(n) \tag{32}$$

No qual pode-se chegar ao erro total instantâneo com a soma de todos os erros de todos os neurônio de saída, dado por :

$$\epsilon(n) = \sum_{j \in C} \epsilon_j(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \tag{33}$$

Com isso, fazendo correções dos pesos sinápticos conectando o neurônio i ao neurônio j, $\forall i, j$ através da regra delta:

$$w_{ji}(n+1) = w_{ji}(n) + \eta \delta_j(n) y_i(n) \tag{34}$$

Onde o gradiente local $\delta_j(n)$ depende se o neurônio em questão é saída ou escondido. Se o neurônio j pertence a camada de saída, o gradiente local é o produto entre a derivada da função e sinal de erro associados a j:

$$\delta_j(n) = e_j(n) \phi_j'(v_j(n)) \tag{35}$$

Se o neurônio j pertence a camada escondida, a equação dele é dado por :

$$\delta_j(n) = \phi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{36}$$

Onde o gradiente local é igual ao produto entre a derivada da função de ativação j e a soma ponderada dos gradientes locais computados para os neurônios da camada da direita conectados a j.

D. K-Nearest Neighbors (KNN)

É um classificador não paramétrico e não linear. Com o objetivo de destinar $\vec{x}$ à classe com maior representatividade no conjunto de K amostras mais próximas.

Dada a probabilidade Θ pode ser aproximada pela proporção de amostras que caem em uma região de volume V. Se K é o número de amostras de um total de N que caem em uma região , pode-se ter uma aproximação para $p(x)$, dado por :

$$p(x) = \frac{K}{NV} \tag{37}$$

De modo não paramétrico a probabilidade condicional da classe ω_j pode ser dado por:

$$p(\vec{x}|\omega_j) = \frac{K_j}{N_j V_R} \tag{38}$$

Onde V_R indica o volume da região de interesse, N_j indica o número total de amostras da classe ω_j e K_j refere-se ao número de amostras da classe ω_j da região de interesse. Com isso a probabilidade a priori pode ser expressa como:

$$p(\omega_j) = \frac{N_j}{N} \tag{39}$$

Na qual N é número total de amostras. Logo, pelo critério *Maximum A Posteriori*(MAP), deve-se designar $\vec{x}$ a classe ω_j se:

$$p(\omega_j|x) > p(\omega_j|\vec{x}), \forall i \neq j \tag{40}$$

E. Nearest Mean Classifier

É um classificador que tem como objetivo minimizar a distancia entre $\vec{x}$ e a média de classe ω_j. Pode-se expressa da seguinte forma:

$$D_j(\vec{x}) = ||\vec{x} - \mu_j||^2, j = 1, 2, ..., c \tag{41}$$

Com isso quanto menor o valor, melhor. Tomamos o negativo de $D_j(\vec{x})$ com o termo quadrático independente de classe para escrever a função discriminante, temos:

$$d_j(\vec{x}) = \vec{x}^T \vec{\mu}_j - \frac{1}{2} \vec{\mu}_j^T \vec{\mu}_j \tag{42}$$

Desse modo a regra se torna: designar $\vec{x}$ à classe que maximiza a função $d_j(\vec{x})$.

F. Bayesiano Sob Hipótese Gaussiana

G. Support Vector Machines (SVM)

H. K-médias

I. GMM

IV. Materias

A. Plataforma Computacional

Para o desenvolvimento deste artigo, foi optado por utilizar o software Visual Studio Code versão 1.84.2.0 para a criação do código em um sistema operacional Windows 10 Pro. O hardware utilizado foi AMD Ryzen 7 5700U 1.8Ghz com 12GB de RAM.

B. Bibliotecas

Bibliotecas utilizadas foram: Pandas, Numpy, Matplotlib e Scikit-Learn.

```
from sklearn import datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Os Submódulos utilizados e as classes especificas utilizada do Scikit-Learn foram :

```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis
from sklearn.decomposition import KernelPCA
from sklearn.manifold import Isomap
from sklearn.manifold import
    LocallyLinearEmbedding
from sklearn.manifold import
    SpectralEmbedding
from sklearn.manifold import TSNE

from sklearn.linear_model import Perceptron
from sklearn.linear_model import
    LogisticRegression
from sklearn.neural_network import
    MLPClassifier
from sklearn.neighbors import
    KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette_score
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import
    pairwise_distances
```

C. Bases de Dados

A base de dados "Digits" da biblioteca do scikit-learn foi importada com o comando:

```
from sklearn import datasets.
digits = datasets.load_digits()
```

É um conjunto de dados que contém imagens de 8x8 de pixel inteiros de dígitos escritos a mão, sendo de 10 classes, ou seja, indo do numero 0 ao 9. As características deste banco de dados tem 1797 instâncias e 64 atributos.

Já a base de dados "Mfeat-pixel", que também é da biblioteca do scikit-learn foi importada com o comando:

```
from sklearn.datasets import fetch_openml
mfeat = fetch_openml(name='mfeat-pixel',
    version=1, parser='liac-arff')
```

Este é um conjunto de dados de amostra das imagens 15 x 16 pixels de de algarismos manuscritos de 0 a 9 digitalizados em 8 bits em escala de cinza. Contendo 200 instâncias por classe, sendo 10 classes,totalizando 2000 instancias. 240 (15 x 16) atributos médios de pixels em 2 x 3 janelas.

V. Experimentos e Resultados

A. Comandos em Python e funções da biblioteca scikit learn utilizadas

Foi criado funções para cada um dos métodos de classificação, priorizando um funcionamento sem advertências proveniente do compilador, generalização para todos os tipos de métodos de redução de dimensionalidade e um tempo satisfatório de processamento.

Para todos os métodos de classificação que precisavam de treinamento, foram utilizado 50% da base de dados e aplicado nos outros 50% o método treinado. Desta forma, a classe especifica para esta execução foi:

```
from sklearn.model_selection import
    train_test_split
```

De modo geral, a utilização desta ferramenta de treino foi especificada da seguinte forma:

```
X_treino, X_teste, Y_treino, Y_teste =
    train_test_split(X_reduzido, Y,
    test_size=0.5, random_state=5)
```

A partir disto, foram executados os comandos para os métodos de classificação e configuradas, quando necessário, especificações que não são padrão na ferramenta.

- Perceptron

```
clf = Perceptron(random_state=1000, tol=1e-3)
clf.fit(X_treino, Y_treino)
```

- Regressão Logística

```
clf = LogisticRegression(max_iter=5000,
    random_state=1000, tol=1e-3)
clf.fit(X_treino, Y_treino)
```

- MLP

```
clf = MLPClassifier(max_iter=1000,
    random_state=1000, tol=1e-3)
clf.fit(X_treino, Y_treino)
```

- KNN

```
clf = KNeighborsClassifier(n_neighbors=7)
clf.fit(X_treino, Y_treino)
```

- Nearest Mean Classifier

```
clf = NearestCentroid()
clf.fit(X_treino, Y_treino)
```

- Bayesiano Sob Hipótese Gaussiana

```
clf = GaussianNB()
clf.fit(X_treino, Y_treino)
```

- SVM - Kernel Linear

```
clf = SVC(kernel='linear')
clf.fit(X_treino, Y_treino)
```

- SVM - Kernel RBF

```
clf = SVC(kernel='rbf')
clf.fit(X_treino, Y_treino)
```

- K-médias

```
clf = KMeans(n_clusters=10, random_state
    =1000, n_init='auto', init='k-means++')
clf.fit(X_reduzido)
```

- GMM

```
clf = GaussianMixture(n_components=10,
    random_state=1000, max_iter=1000,
    init_params='kmeans')
clf.fit(X_reduzido)
```

É válido ressaltar, que os métodos de K-médias e GMM não fazem uso de dados de treinamento.

Uma particularidade se deu nos métodos de redução de LLE, Laplacian Eingenmaps e ao utilizar os dados brutos. Quando aplicado o método de classificação Bayesiano Sob Hipótese Gaussiana com a utilização da ferramenta "Quadratic Discriminant Analysis" da seguinte forma:

```
clf = QuadraticDiscriminantAnalysis()
clf.fit(X_treino, Y_treino)
```

O compilador informou que as variáveis são colineares, portanto se fez necessário utilizar uma outra ferramenta para a aplicação. Dentre os módulos de implementação de algoritmo do Naive Bayes o que melhor se adaptou aos dados foi o Gaussian Naive Bayes (GaussianNB).

O método de redução do ISOMAP quando aplicado o método de classificação de regressão logística, foi configurado o máximo de interações em 5000, isso foi feito para que a função convergisse.

B. Resultados obtidos

1) Base de Dados Digits: Dentre os métodos de redução propostos neste artigo, houve 2 que mostraram certos obstáculos para se fazer uma redução para 20D. Sendo eles o LDA, que impossibilita uma redução que seja maior que o número de classes -1, ou seja, como temos 10 classes, sendo os dígitos de 0 a 9, o numero mínimo a ser reduzido é para 9 componentes. O outro é o t-SNE, que para poder utilizar uma redução para 20D é necessário mudar o método de utilização na configuração da ferramenta. No código a seguir é possível ver a configuração utilizada.

```
dados_tsne = TSNE(n_components=20,
    learning_rate='auto', init='random',
    perplexity=10, method = 'exact')
```

No método foi configurado *exact*, entretanto, o tempo de processamento para essa redução aumenta consideravelmente, se comparada com o método padrão da ferramenta, *barnes_hut*. O tempo de processamento com o uso do método *exact* ficou entre 5 minutos e 17 segundos e 5 minutos e 32 segundos. O valor de acurácia média ficou em 75,7%.

Com a utilização do método *barnes_hut*, o tempo de processamento fica em torno de 6,5 segundos, entretanto sendo

obrigatório o uso de uma redução para 3 componentes. O valor de acurácia média ficou em 77,6%.

Para concluir a questão do método de redução do t-SNE, foi optado por utilizar o método *barnes_hut*, devido ao seu tempo de processamento, por mais que ele tenha uma diferença quando reduzido para 20D, essa diferença de acuracidade média é de 0,019, não impactando significativamente quando comparados com outros métodos que tiveram uma redução para 20D. Ficando da seguinte forma:

```
dados_tsne = TSNE(n_components=3,
    learning_rate='auto', init='random',
    perplexity=10, method = 'barnes_hut')
```

Portanto, a tabela I e II mostra a acurácia dos dados.

TABELA I:
Tabela de acurácia para os Dados da base "Digits" (Parte 1)

	PCA	LDA	KPCA	ISOMAP	LLE
Perceptron	0.872	0.952	0.823	0.814	0.835
Reg. Logística	0.943	0.953	0.940	0.964	0.932
MLP	0.960	0.964	0.959	0.976	0.963
KNN	0.973	0.969	0.973	0.983	0.967
Nearest Mean	0.908	0.963	0.908	0.954	0.956
Bayesiano	0.934	0.960	0.935	0.957	0.966
SVM (linear)	0.969	0.959	0.968	0.980	0.367
SVM (rbf)	0.988	0.966	0.988	0.987	0.973
K-médias	0.214	0.396	0.214	0.337	0.339
GMM	0.208	0.390	0.208	0.335	0.358
Média	0.724	0.770	0.720	0.753	0.696

TABELA II:
Tabela de acurácia para os Dados da base "Digits" (Parte 2)

	Lap. Eig.	t-SNE	Raw data
Perceptron	0.690	0.657	0.938
Reg. Logística	0.089	0.939	0.952
MLP	0.089	0.982	0.958
KNN	0.960	0.987	0.978
Nearest Mean	0.927	0.964	0.907
Bayesiano	0.921	0.973	0.858
SVM (linear)	0.089	0.970	0.974
SVM (rbf)	0.967	0.989	0.981
K-médias	0.183	0.481	0.182
GMM	0.186	0.481	0.178
Média	0.464	0.766	0.719

É possível observar na tabela I, o maior valor médio, sendo o método de redução com LDA, com 0,770. Entretanto é válido ressaltar, que este método não fez uma redução para 20D, ele fez uma redução para 9D, por conta de suas limitações. Sendo os maiores valores as reduções com t-SNE, com valor de 0,766, e ISOMAP com valor de 0,753.

Todavia, para ter uma visualização mais clara de como as classes estão sendo espalhadas, esses 3 métodos de redução foram reduzidos para 2D, plotados, e utilizado os 10 métodos de classificação para seus dados reduzidos.

Inicialmente podemos observar o espalhamento dos dados com a redução 2D do LDA na Fig. 4:

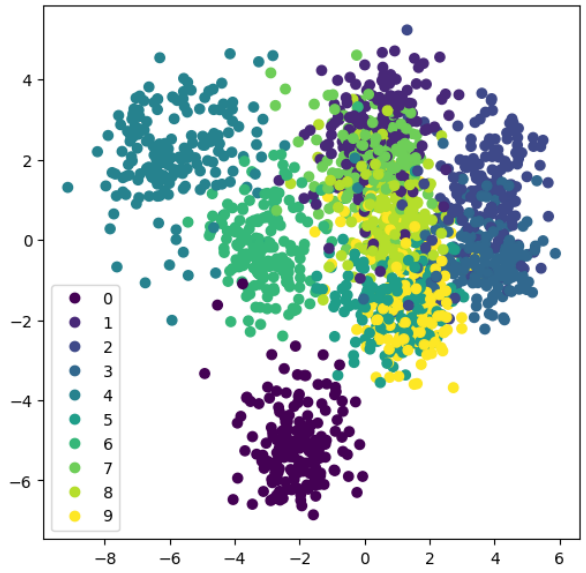


Fig. 4: Redução 2D com LDA dos dados Digits. Fonte: Acervo Pessoal

Nela podemos ver que a separação dos dados não ocorreu de forma muito satisfatória, pois houve pelo menos 3 classes sobrepostas, tornando difícil uma separação.

Na Fig. 5 está uma redução 2D com a utilização do método ISOMAP.

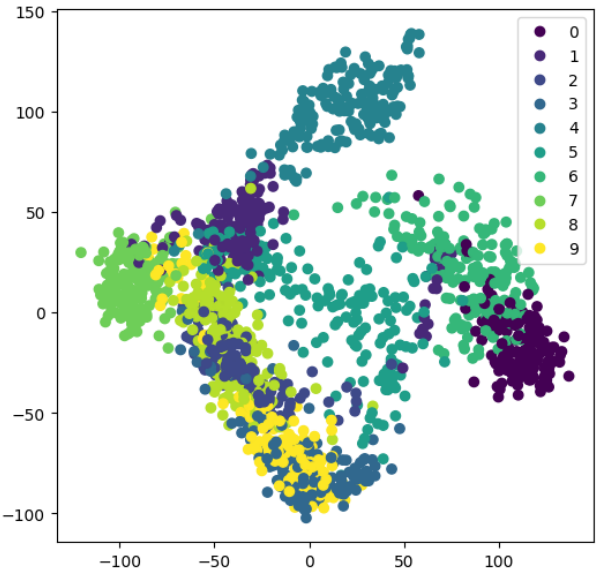


Fig. 5: Redução 2D com ISOMAP dos dados Digits. Fonte: Acervo Pessoal

Neste caso, é possível ver um sobreposição de dados, menor que com a utilização do LDA, entretanto um espalhamento maior dos dados de algumas das classes.

E para concluir, a Fig. 6 mostra uma redução 2D com a utilização do método de t-SNE.

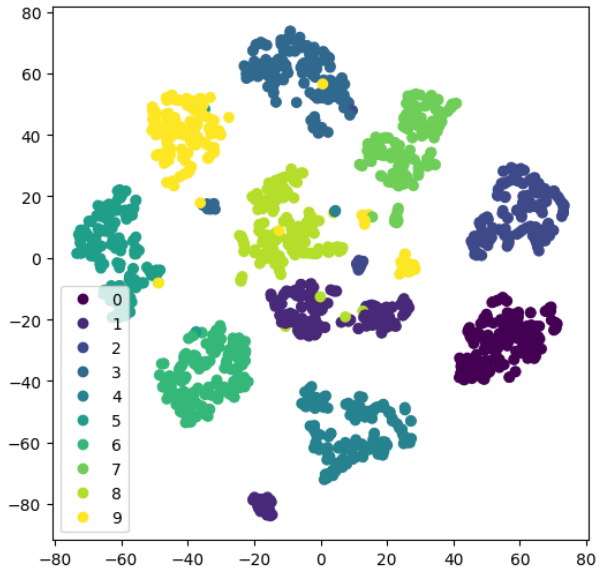


Fig. 6: Redução 2D com ISOMAP dos dados Digits. Fonte: Acervo Pessoal

Na Fig. 6, a separação das classes é mais clara e definida além da sobreposição dos dados ser mínima.

O valores das acurácias podem vir a ter uma pequena variação por conta das randomizações dos algoritmos. Devido a isso, foi escolhido os três melhores métodos de redução, para a aplicação dos métodos de classificação com redução em 2D. Estes três métodos foram escolhidos por estarem mostrando sempre os melhores valores em diversas execuções do programa, mesmo com variações. Ao aplicar os métodos de classificação nos 3 conjuntos de dados 2D, chegamos aos seguintes valores:

TABELA III:
Tabela de acurácia 2D para os Dados da base "Digits"

	LDA	ISOMAP	t-SNE
Perceptron	0.952	0.814	0.657
Reg. Logística	0.953	0.964	0.939
MLP	0.964	0.976	0.982
KNN	0.969	0.983	0.987
Nearest Mean	0.963	0.954	0.964
Bayesiano	0.960	0.957	0.973
SVM (linear)	0.959	0.980	0.970
SVM (rbf)	0.966	0.987	0.989
K-médias	0.396	0.337	0.481
GMM	0.390	0.335	0.481
Média	0.770	0.753	0.766

Na tabela III, a melhor acurácia média está com o método de redução LDA, no valor de 0,770, seguida do t-SNE, com 0,766, e a terceira acurácia mais alta é ISOMAP com 0,753.

De acordo com os dados da Tabela III, o melhor método de classificação foi o SVM com kernel rbf, no valor de 0,989, utilizando da redução de dados do t-SNE, pois apresenta a acurácia mais alta. Seguido dos métodos de classificação KNN, no valor de 0,987, utilizando a redução de ISOMAP e SVM com kernel rbf, com valor de 0,987, utilizando o método de redução t-SNE.

O código em Python utilizado para o banco de dados *Digits* consta no apêndice 01.

2) Base de Dados mfeath-pixel: A principal diferença neste banco de dados foi a necessidade de um pré-processamento dos dados, para em seguida ser possível aplicar os métodos de redução e de classificação.

Para este pré-processamento foi necessário utilizar 2 submódulos disponíveis no scikit-learn. O primeiro tem como objetivo converter rótulos em números inteiros. Está aplicação foi feita da seguinte maneira:

```
1 from sklearn.preprocessing import
      LabelEncoder
2 Y = LabelEncoder().fit_transform(Y)
```

A necessidade deste pré-processamento foi observada ao tentar fazer fazer a plotagem dos dados em um espaço 2D com uso do "matplotlib.pyplot". No qual o compilador informava um erro referente as categorias.

O outro pré-processamento foi devido a falta de padronização ou normalização nos dados. Foi optado por padroniza-los da seguinte forma:

```
1 from sklearn.preprocessing import
      StandardScaler
2 X = StandardScaler().fit_transform(X)
```

Para chegar a está conclusão foi devido a diversos erros nos métodos de redução e classificação. Primeiramente foi observado um problema de convergir os dados quando aplicado redução 20D utilizando ISOMAP com método de classificação de regressão logística. Uma das tentativas para resolver essa questão foi aumentar o numero máximo de interação, que foi aumentada em intervalos de 5000 até o total de 500000. Como não resolveu o problema, foi então necessário a aplicação do pré-processamento com o objetivo de ajustar os parâmetros dos *scaler* para uma padronização.

Houveram outros problemas que essa padronização resolveu como no método de redução LDA, que apresentou problemas ao identificar o tipo dos dados. Já nos dados brutos, resolveu a questão de não ser possível multiplicar sequências não inteiras, no Percetron, na regressão logística e no MLP. Ainda falando dos dados brutos, ao utilizar o método de classificação *Nearest Centroid* ele informou não suportar os tipos dos operandos dessa base de dados.

As limitações ditas anteriormente nos resultados da base de dados *Digits* referente aos métodos de redução LDA e t-SNE, permanecem da mesma forma para esse banco de dados.

As acurácias dos dados obtidos com reduções de 20D com todos os métodos de classificação estão nas tabelas IV e V:

Na tabela IV, é possível observar que novamente o método de redução LDA mostra a maior acurácia média com 0,795, seguidos de t-SNE com valor de 0,765 da acurácia média como segundo maior valor, na tabela V e ISOMAP com 0,737 na tabela IV.

Devido a esses três métodos terem mostrado novamente as maiores acurácias, neles foram aplicados a redução em 2D e utilizado os 10 métodos de classificação, além de feito inicialmente a plotagem dos dados no espaço 2D, como pode ser observado nas Fig. 7, 8 e 9.

TABELA IV:
Tabela de acurácia para os Dados da base ”Mfeat-Pixel” (Parte 1)

	PCA	LDA	KPCA	ISOMAP	LLE
Perceptron	0.901	0.968	0.923	0.893	0.844
Reg. Logística	0.950	0.969	0.950	0.954	0.931
MLP	0.957	0.978	0.957	0.941	0.954
KNN	0.960	0.976	0.960	0.960	0.960
Nearest Mean	0.915	0.979	0.915	0.935	0.955
Bayesiano	0.936	0.976	0.937	0.942	0.936
SVM (linear)	0.962	0.974	0.962	0.959	0.495
SVM (rbf)	0.976	0.975	0.976	0.965	0.955
K-médias	0.190	0.475	0.190	0.278	0.366
GMM	0.182	0.472	0.182	0.277	0.365
Média	0.721	0.795	0.723	0.737	0.706

TABELA V:
Tabela de acurácia para os Dados da base ”Mfeat-Pixel” (Parte 2)

	Lap. Eig.	t-SNE	Raw data
Perceptron	0.586	0.774	0.909
Reg. Logística	0.087	0.969	0.961
MLP	0.087	0.950	0.962
KNN	0.933	0.969	0.960
Nearest Mean	0.924	0.965	0.919
Bayesiano	0.893	0.964	0.894
SVM (linear)	0.087	0.973	0.966
SVM (rbf)	0.946	0.970	0.974
K-médias	0.212	0.443	0.119
GMM	0.210	0.442	0.119
Média	0.451	0.765	0.707

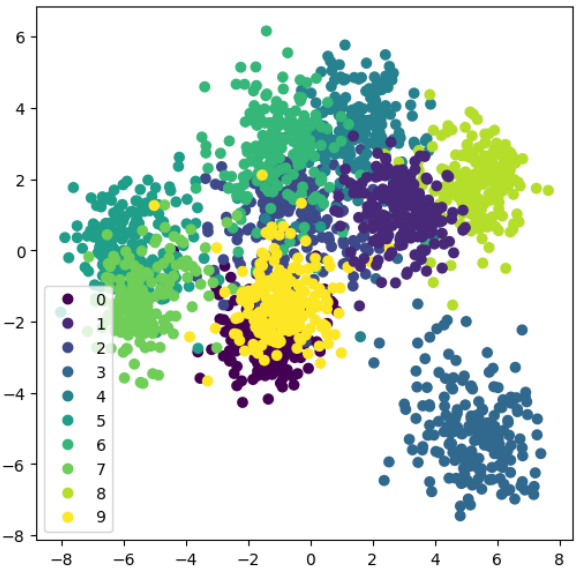


Fig. 7: Redução 2D com LDA dos dados Mfeat-Pixel. Fonte: Acervo Pessoal

Neste caso, é possível observar o agrupamento dos dados para cada classe, um espalhamento de dados intraclasse que acaba mostrando sobreposição de diversas classes.

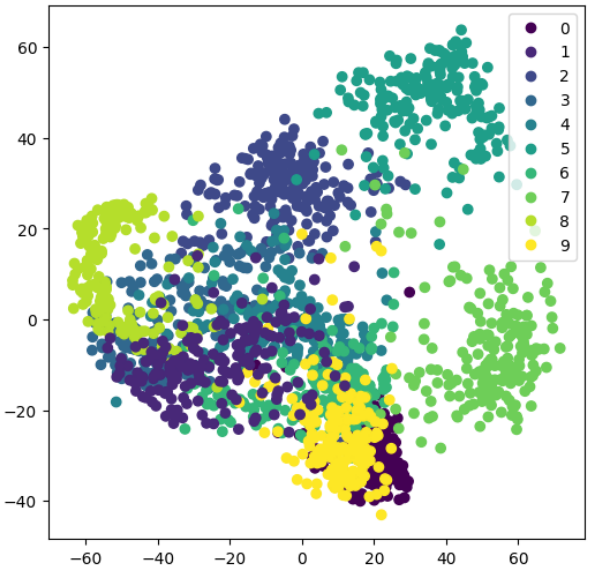


Fig. 8: Redução 2D com ISOMAP dos dados Mfeat-Pixel. Fonte: Acervo Pessoal

No ISOMAP houve um espalhamento muito grande de dados intraclasse, e uma difícil observação de todas as classes devido a alta sobreposição.

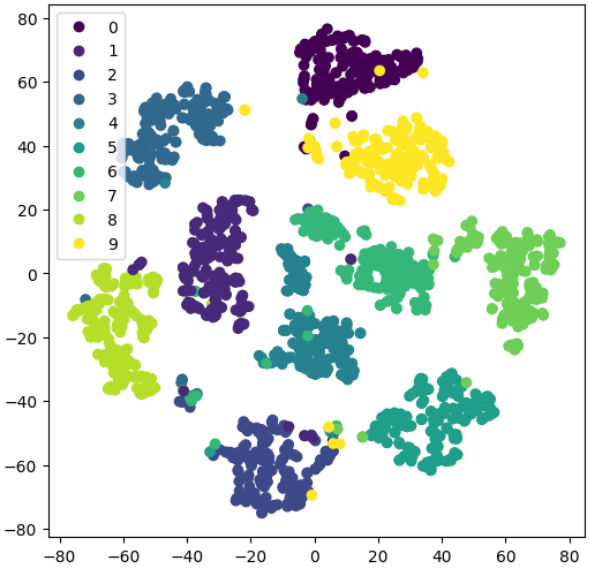


Fig. 9: Redução 2D com t-SNE dos dados Mfeat-Pixel. Fonte: Acervo Pessoal

E novamente mostrando maior eficacia em separar as classes, e agrupar os dados intraclasse com baixa sobreposição pode ser observada no método t-SNE.

Agora pode ser observado a tabela de acurácia para os dados 2D e os métodos de classificação aplicados:

Pode ser observado na tabela VI, que a acurácia média utilizando o método de redução t-SNE apresentou o maior valor, 0,724, seguido dos métodos LDA e ISOMAP com 0,634 e 0,529, respectivamente.

TABELA VI:
Tabela de acurácia 2D para os Dados da base "Mfeat-Pixel"

	LDA	ISOMAP	t-SNE
Perceptron	0.514	0.241	0.169
Reg. Logística	0.797	0.689	0.960
MLP	0.799	0.683	0.963
KNN	0.790	0.690	0.965
Nearest Mean	0.797	0.665	0.956
Bayesiano	0.786	0.668	0.962
SVM (linear)	0.798	0.702	0.961
SVM (rbf)	0.797	0.702	0.958
K-médias	0.450	0.404	0.536
GMM	0.443	0.377	0.535
Média	0.634	0.529	0.724

O método de classificação que mostrou maior desempenho foi o KNN com redução de t-SNE com valores médio de acurácia 0,965. Desta vez, todo os métodos supervisionados quando aplicados na redução de t-SNE mostraram valores acima de 0,95 com exceção do Perceptron, apresentando valor de 0,169. Mesmo assim, os métodos não supervisionados com redução t-SNE mostram os maiores valores se comparados com outros métodos de redução, sendo eles de 0,536 para K-médias e 0,535 para GMM.

Portanto, o método que mostrou melhores resultados para o banco de dados *Mfeat-Pixel* é o método de redução t-SNE com classificador KNN.

O código em Python utilizado para o banco de dados *Mfeat-Pixel* consta no apêndice 01.

VI. Conclusão

Portanto, é essencial entender como os dados estão dispostos no espaço e em qual formato, para então poder escolher com maior assertividade o método de redução mais apropriado para o caso em questão, entendendo que todos métodos tem suas limitações e são mais apropriados para determinadas situações. Como foi observado no presente artigo, os 2 bancos de dados precisavam de uma redução de dimensionalidade para reduzir o tempo de processamento no método de classificação.

Os métodos de classificação supervisionados mostraram que majoritariamente se adéquam melhor aos dados, mostrando valores acima de 0,8, com certas exceções quando utilizado métodos de redução não apropriados. Desde modo pode-se observar o método de redução t-SNE, que apresentou visualmente um adequado espaçamento das classes, e todos os métodos de classificação supervisionados apresentou valores acima de 0,9, com exceção do uso do Perceptron, com valor abaixo de 0,7, onde aparentou dificuldades para se adequar aos dados, se comparado com os outros métodos supervisionados. Todavia, ele ainda mostrou valores superiores aos métodos de classificação não supervisionados, que apresentou valores abaixo de 0,5. O interessante deste método de redução, é a possibilidade de poder alterar em sua configuração para que ele possa executar com outras dimensões, entretanto isso aumenta o custo computacional significativamente.

Dito isso, a questão das dificuldades apresentadas neste artigo, sendo ela principalmente a adequação dos métodos de redução para executar em 20D, onde não foi possível com uso do LDA e foi possível com t-SNE alterando a configuração de execução, mas elevando o tempo de processamento. Outro ponto é a limitação de alguns métodos de classificação, onde foi necessário alterar o submódulo do Bayesiano sob hipótese Gaussiana para poder executar dados que apresentavam valores colineares, passou-se a utilizando Naive Bayes no lugar de *Quadratic Discriminant Analysis*.

Concluindo, o melhor método de redução apresentado neste artigo foi o t-SNE, para ambas base de dados, pois nela houve uma acurácia das mais altas dentre as colocadas em teste, apresentou um adequado espaçamento das classes e dos agrupamento dos dados dessas classes. Um tempo de processamento que não foi superior a 20 segundos, quando executado a redução e todas as classificações, desde que, utilizado o *method = 'barnes_hut'* e obrigatoriamente *n_components*= valores menores do que 4.

APÊNDICE 1

```
1
2  from sklearn import datasets
3
4  from sklearn.decomposition import PCA
5  from sklearn.discriminant_analysis import
       LinearDiscriminantAnalysis
6  from sklearn.decomposition import KernelPCA
7  from sklearn.manifold import Isomap
8  from sklearn.manifold import
       LocallyLinearEmbedding
9  from sklearn.manifold import
       SpectralEmbedding
10 from sklearn.manifold import TSNE
11
12 from sklearn.linear_model import Perceptron
13 from sklearn.linear_model import
       LogisticRegression
14 from sklearn.neural_network import
       MLPClassifier
15 from sklearn.neighbors import
       KNeighborsClassifier
16 from sklearn.neighbors import NearestCentroid
17 from sklearn.naive_bayes import GaussianNB
18 from sklearn.svm import SVC
19 from sklearn.cluster import KMeans
20 from sklearn.mixture import GaussianMixture
21
22 from sklearn.metrics import silhouette_score
23 from sklearn.model_selection import
       train_test_split
24 from sklearn.metrics import
       pairwise_distances
25
26 import pandas as pd
27 import numpy as np
28 import matplotlib.pyplot as plt
29
30 digits = datasets.load_digits()
31
32 X = digits.data
33 Y = digits.target
34
35 matriz_acc = [[0] * 8 for _ in range(11)]
36
37 print("Features:", digits.data.shape)
38
39 plt.figure(figsize=(20,4))
40 for index, (image, label) in enumerate(zip(X
       [0:10], Y[0:10])):
41     plt.subplot(1, 10, index +1)
42     plt.imshow(np.reshape(image, (8, 8)),
           cmap=plt.cm.grey)
43     plt.title('Target:■{}\n'.format(label,
           fontsize = 20))
44
45 def funcao_perceptron(X_reduzido, Y):
46     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
47
48     clf = Perceptron(random_state=1000, tol=1
           e-3)
49     clf.fit(X_treino, Y_treino)
50
51     return clf.score(X_teste, Y_teste)
52
53 def funcao_reg_log(X_reduzido, Y):
54     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
55
56     clf = LogisticRegression(max_iter=5000,
           random_state=1000, tol=1e-3)
57     clf.fit(X_treino, Y_treino)
58
59     return clf.score(X_teste, Y_teste)
60
61 def funcao_MLP(X_reduzido, Y):
62     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
63
64     clf = MLPClassifier(max_iter=1000,
           random_state=1000, tol=1e-3)
65     clf.fit(X_treino, Y_treino)
66
67     return clf.score(X_teste, Y_teste)
68
69 def funcao_KNN(X_reduzido, Y):
70     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
71
72     clf = KNeighborsClassifier(n_neighbors=7)
73     clf.fit(X_treino, Y_treino)
74
75     return clf.score(X_teste, Y_teste)
76
77 def funcao_NC(X_reduzido, Y):
78     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
79
80     clf = NearestCentroid()
81     clf.fit(X_treino, Y_treino)
82
83     return clf.score(X_teste, Y_teste)
84
85 def funcao_BSHG(X_reduzido, Y):
86     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
87
88     clf = GaussianNB()
89     clf.fit(X_treino, Y_treino)
90
91     return clf.score(X_teste, Y_teste)
92
93 def funcao_SVM_linear(X_reduzido, Y):
94     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
95
96     clf = SVC(kernel='linear')
97     clf.fit(X_treino, Y_treino)
98
99     return clf.score(X_teste, Y_teste)
100
101 def funcao_SVM_RBF(X_reduzido, Y):
102     X_treino, X_teste, Y_treino, Y_teste =
           train_test_split(X_reduzido, Y,
           test_size=0.5, random_state=5)
103
104     clf = SVC(kernel='rbf')
105     clf.fit(X_treino, Y_treino)
106
107     return clf.score(X_teste, Y_teste)
108
109 def funcao_KMeans(X_reduzido):
110
111     clf = KMeans(n_clusters=10, random_state
           =1000, n_init='auto', init='k-means++
           ')
112     clf.fit(X_reduzido)
113
```

```
114         return silhouette_score(X_reduzido, clf.
                labels_)
115
116     def funcao_GMM(X_reduzido):
117
118         clf = GaussianMixture(n_components=10,
                random_state=1000, max_iter=1000,
                init_params='kmeans')
119         clf.fit(X_reduzido)
120
121         previsoes = clf.predict(X_reduzido)
122         return silhouette_score(X_reduzido,
                previsoes)
123
124     funcoes = {
125         0: funcao_perceptron,
126         1: funcao_reg_log,
127         2: funcao_MLP,
128         3: funcao_KNN,
129         4: funcao_NC,
130         5: funcao_BSHG,
131         6: funcao_SVM_linear,
132         7: funcao_SVM_RBF,
133         8: funcao_KMeans,
134         9: funcao_GMM,
135     }
136
137     dados_PCA = PCA(n_components=20, random_state
            =1000)
138     X_reduzido = dados_PCA.fit_transform(X)
139
140     print(dados_PCA.explained_variance_ratio_)
141
142     for i in range (0, 10):
143         if i < 8:
144             resultados = [funcoes[i](X_reduzido,
                    Y) for _ in range(20)]
145             matriz_acc[i][0] = np.mean(resultados
                    )
146         else:
147             resultados = [funcoes[i](X_reduzido)
                    for _ in range(20)]
148             matriz_acc[i][0] = np.mean(resultados
                    )
149
150     dados_LDA = LinearDiscriminantAnalysis(
            n_components=9)
151     X_reduzido = dados_LDA.fit_transform(X, Y)
152
153     print(dados_LDA.explained_variance_ratio_)
154
155     for i in range (0, 10):
156         if i < 8:
157             resultados = [funcoes[i](X_reduzido,
                    Y) for _ in range(20)]
158             matriz_acc[i][1] = np.mean(resultados
                    )
159         else:
160             resultados = [funcoes[i](X_reduzido)
                    for _ in range(20)]
161             matriz_acc[i][1] = np.mean(resultados
                    )
162
163     dados_KPCA = KernelPCA(n_components=20,
            kernel='linear')
164     X_reduzido = dados_KPCA.fit_transform(X)
165     explained_variance_ratio = np.var(X_reduzido,
             axis=0) / np.var(X, axis=0).sum()
166
167     print(explained_variance_ratio)
168
169     for i in range (0, 10):
170         if i < 8:
171             resultados = [funcoes[i](X_reduzido,
                    Y) for _ in range(20)]
172             matriz_acc[i][2] = np.mean(resultados
                    )
173         else:
174             resultados = [funcoes[i](X_reduzido)
                    for _ in range(20)]
175             matriz_acc[i][2] = np.mean(resultados
                    )
176
177
178     dados_ISOMAP = Isomap(n_components=20,
            n_neighbors=9, eigen_solver='auto')
179     X_reduzido = dados_ISOMAP.fit_transform(X)
180
181     dist_original = pairwise_distances(X, metric=
            'euclidean')
182     dist_reduzida = pairwise_distances(X_reduzido
            , metric='euclidean')
183     erro_relativo = np.abs(dist_original -
            dist_reduzida).sum() / dist_original.sum
            ()
184
185     print("Erro Relativo de Preserva  o de
            Dist ncias:", erro_relativo)
186
187     for i in range (0, 10):
188         if i < 8:
189             resultados = [funcoes[i](X_reduzido,
                    Y) for _ in range(20)]
190             matriz_acc[i][3] = np.mean(resultados
                    )
191         else:
192             resultados = [funcoes[i](X_reduzido)
                    for _ in range(20)]
193             matriz_acc[i][3] = np.mean(resultados
                    )
194
195     dados_LLE = LocallyLinearEmbedding(
            n_components=20, n_neighbors=9, max_iter
            =1000)
196     X_reduzido = dados_LLE.fit_transform(X)
197
198
199
200     for i in range (0, 10):
201         if i < 8:
202             resultados = [funcoes[i](X_reduzido,
                    Y) for _ in range(20)]
203             matriz_acc[i][4] = np.mean(resultados
                    )
204         else:
205             resultados = [funcoes[i](X_reduzido)
                    for _ in range(20)]
206             matriz_acc[i][4] = np.mean(resultados
                    )
207
208     dados_lap_eig = SpectralEmbedding(
            n_components=20, random_state=1000)
209     X_reduzido = dados_lap_eig.fit_transform(X)
210     explained_variance_ratio = np.var(X_reduzido,
             axis=0) / np.var(X, axis=0).sum()
211
212     print(X_reduzido.shape)
213     print(explained_variance_ratio)
214
215     for i in range (0, 10):
216         if i < 8:
217             resultados = [funcoes[i](X_reduzido,
                    Y) for _ in range(20)]
218             matriz_acc[i][5] = np.mean(resultados
                    )
219         else:
220             resultados = [funcoes[i](X_reduzido)
                    for _ in range(20)]
```

```
            matriz_acc[i][5] = np.mean(resultados
                )

dados_tsne = TSNE(n_components=3,
    learning_rate='auto', init='random',
    perplexity=10, method = 'barnes_hut')
X_reduzido = dados_tsne.fit_transform(X)

print("Shape do dataset reduzido:",
    X_reduzido.shape)
print(dados_PCA.explained_variance_ratio_)

for i in range (0, 10):
    if i < 8:
        resultados = [funcoes[i](X_reduzido,
            Y) for _ in range(20)]
        matriz_acc[i][6] = np.mean(resultados
            )
    else:
        resultados = [funcoes[i](X_reduzido)
            for _ in range(20)]
        matriz_acc[i][6] = np.mean(resultados
            )

for i in range (0, 10):
    if i < 8:
        resultados = [funcoes[i](X, Y) for _
            in range(20)]
        matriz_acc[i][7] = np.mean(resultados
            )
    else:
        resultados = [funcoes[i](X) for _ in
            range(20)]
        matriz_acc[i][7] = np.mean(resultados
            )

nomes_colunas = ["PCA", "LDA", "KPCA", "
    ISOMAP", "LLE", "Lap. Eig.", "t-SNE", "
    Raw data"]
nomes_linhas = ["Perceptron", "Reg.
    Log stica", "MLP", "KNN", "Nearest Mean"
    , "Bayesiano", "SVM (linear)", "SVM (rbf)
    ", "K-m dias", "GMM", "M dia"]

df = pd.DataFrame(matriz_acc[:11], columns=
    nomes_colunas, index=nomes_linhas)

df.loc["M dia"] = df.mean()

pd.set_option('display.precision', 3)
print(df)


x_lda = LinearDiscriminantAnalysis(
    n_components=2).fit_transform(X,Y)

plt.figure(figsize=(6,6))
sc = plt.scatter(x_lda [:, 0], x_lda [:, 1],
    c=Y)
plt.legend(handles = sc.legend_elements()[0],
     labels=list(range(10)))
plt.show()

x_isomap = Isomap(n_components=2, n_neighbors
    =9, eigen_solver='auto').fit_transform(X)

plt.figure(figsize=(6,6))
sc = plt.scatter(x_isomap [:, 0], x_isomap
    [:, 1], c=Y)
plt.legend(handles = sc.legend_elements()[0],
     labels=list(range(10)))
plt.show()

x_tsne = TSNE(n_components=2, learning_rate='
    auto', init='random', perplexity=10).
    fit_transform(X)

plt.figure(figsize=(6,6))
sc = plt.scatter(x_tsne [:, 0], x_tsne [:,
    1], c=Y)
plt.legend(handles = sc.legend_elements()[0],
     labels=list(range(10)))
plt.show()
```

APÊNDICE 2

```

from sklearn.datasets import fetch_openml

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis
from sklearn.decomposition import KernelPCA
from sklearn.manifold import Isomap
from sklearn.manifold import
    LocallyLinearEmbedding
from sklearn.manifold import
    SpectralEmbedding
from sklearn.manifold import TSNE

from sklearn.linear_model import Perceptron
from sklearn.linear_model import
    LogisticRegression
from sklearn.neural_network import
    MLPClassifier
from sklearn.neighbors import
    KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette_score
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import
    pairwise_distances
from sklearn.preprocessing import
    StandardScaler
from sklearn.preprocessing import
    LabelEncoder

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

mfeat = fetch_openml(name='mfeat-pixel',
    version=1, parser='liac-arff')

X = mfeat.data
Y = mfeat.target

Y = LabelEncoder().fit_transform(Y)

X = StandardScaler().fit_transform(X)

matriz_acc = [[0] * 8 for _ in range(11)]

print("Shape dos dados:", X.shape)
print("N mero de classes:", len(set(Y)))

def funcao_perceptron(X_reduzido, Y):
    X_treino, X_teste, Y_treino, Y_teste =
        train_test_split(X_reduzido, Y,
        test_size=0.5, random_state=5)

```

```
49      clf = Perceptron(random_state=1000, tol=1
            e-3)
50      clf.fit(X_treino, Y_treino)
51
52      return clf.score(X_teste, Y_teste)
53
54  def funcao_reg_log(X_reduzido, Y):
55      X_treino, X_teste, Y_treino, Y_teste =
            train_test_split(X_reduzido, Y,
            test_size=0.5, random_state=5)
56
57      clf = LogisticRegression(max_iter=5000,
            random_state=1000, tol=1e-3)
58      clf.fit(X_treino, Y_treino)
59
60      return clf.score(X_teste, Y_teste)
61
62  def funcao_MLP(X_reduzido, Y):
63      X_treino, X_teste, Y_treino, Y_teste =
            train_test_split(X_reduzido, Y,
            test_size=0.5, random_state=5)
64
65      clf = MLPClassifier(max_iter=1000,
            random_state=1000, tol=1e-3)
66      clf.fit(X_treino, Y_treino)
67
68      return clf.score(X_teste, Y_teste)
69
70  def funcao_KNN(X_reduzido, Y):
71      X_treino, X_teste, Y_treino, Y_teste =
            train_test_split(X_reduzido, Y,
            test_size=0.5, random_state=5)
72
73      clf = KNeighborsClassifier(n_neighbors=7)
74      clf.fit(X_treino, Y_treino)
75
76      return clf.score(X_teste, Y_teste)
77
78  def funcao_NC(X_reduzido, Y):
79      X_treino, X_teste, Y_treino, Y_teste =
            train_test_split(X_reduzido, Y,
            test_size=0.5, random_state=5)
80
81      clf = NearestCentroid()
82      clf.fit(X_treino, Y_treino)
83
84      return clf.score(X_teste, Y_teste)
85
86  def funcao_BSHG(X_reduzido, Y):
87      X_treino, X_teste, Y_treino, Y_teste =
            train_test_split(X_reduzido, Y,
            test_size=0.5, random_state=5)
88
89      clf = GaussianNB()
90      clf.fit(X_treino, Y_treino)
91
92      return clf.score(X_teste, Y_teste)
93
94  def funcao_SVM_linear(X_reduzido, Y):
95      X_treino, X_teste, Y_treino, Y_teste =
            train_test_split(X_reduzido, Y,
            test_size=0.5, random_state=5)
96
97      clf = SVC(kernel='linear')
98      clf.fit(X_treino, Y_treino)
99
100     return clf.score(X_teste, Y_teste)
101
102 def funcao_SVM_RBF(X_reduzido, Y):
103     X_treino, X_teste, Y_treino, Y_teste =
            train_test_split(X_reduzido, Y,
            test_size=0.5, random_state=5)
104
105     clf = SVC(kernel='rbf')
106     clf.fit(X_treino, Y_treino)
107
108     return clf.score(X_teste, Y_teste)
109
110 def funcao_KMeans(X_reduzido):
111
112     clf = KMeans(n_clusters=10, random_state
            =1000, n_init='auto', init='k-means++
            ')
113     clf.fit(X_reduzido)
114
115     return silhouette_score(X_reduzido, clf.
            labels_)
116
117 def funcao_GMM(X_reduzido):
118
119     clf = GaussianMixture(n_components=10,
            random_state=1000, max_iter=1000,
            init_params='kmeans')
120     clf.fit(X_reduzido)
121
122     previsoes = clf.predict(X_reduzido)
123     return silhouette_score(X_reduzido,
            previsoes)
124
125 funcoes = {
126     0: funcao_perceptron,
127     1: funcao_reg_log,
128     2: funcao_MLP,
129     3: funcao_KNN,
130     4: funcao_NC,
131     5: funcao_BSHG,
132     6: funcao_SVM_linear,
133     7: funcao_SVM_RBF,
134     8: funcao_KMeans,
135     9: funcao_GMM,
136 }
137
138 dados_PCA = PCA(n_components=20, random_state
        =1000)
139 X_reduzido = dados_PCA.fit_transform(X)
140
141 print(dados_PCA.explained_variance_ratio_)
142
143 for i in range (0, 10):
144     if i < 8:
145         resultados = [funcoes[i](X_reduzido,
                Y) for _ in range(20)]
146         matriz_acc[i][0] = np.mean(resultados
                )
147     else:
148         resultados = [funcoes[i](X_reduzido)
                for _ in range(20)]
149         matriz_acc[i][0] = np.mean(resultados
                )
150
151 dados_LDA = LinearDiscriminantAnalysis(
        n_components=9)#9 ,2
152 X_reduzido = dados_LDA.fit_transform(X, Y)
153
154 print(dados_LDA.explained_variance_ratio_)
155
156 for i in range (0, 10):
157     if i < 8:
158         resultados = [funcoes[i](X_reduzido,
                Y) for _ in range(20)]
159         matriz_acc[i][1] = np.mean(resultados
                )
160     else:
161         resultados = [funcoes[i](X_reduzido)
                for _ in range(20)]
162         matriz_acc[i][1] = np.mean(resultados
                )
163
```

```
164 dados_KPCA = KernelPCA(n_components=20,
        kernel='linear')
165 X_reduzido = dados_KPCA.fit_transform(X)
166 explained_variance_ratio = np.var(X_reduzido,
        axis=0) / np.var(X, axis=0).sum()
167
168 print(explained_variance_ratio)
169
170 for i in range (0, 10):
171     if i < 8:
172         resultados = [funcoes[i](X_reduzido,
                Y) for _ in range(20)]
173         matriz_acc[i][2] = np.mean(resultados
                )
174     else:
175         resultados = [funcoes[i](X_reduzido)
                for _ in range(20)]
176         matriz_acc[i][2] = np.mean(resultados
                )
177
178
179 dados_ISOMAP = Isomap(n_components=20,
        n_neighbors=9, eigen_solver='auto')#20, 2
180 X_reduzido = dados_ISOMAP.fit_transform(X)
181
182 dist_original = pairwise_distances(X, metric=
        'euclidean')
183 dist_reduzida = pairwise_distances(X_reduzido
        , metric='euclidean')
184 erro_relativo = np.abs(dist_original -
        dist_reduzida).sum() / dist_original.sum
        ()
185
186 print("Erro Relativo de Preserva  o de
        Dist ncias:", erro_relativo)
187
188 for i in range (0, 10):
189     if i < 8:
190         resultados = [funcoes[i](X_reduzido,
                Y) for _ in range(20)]
191         matriz_acc[i][3] = np.mean(resultados
                )
192     else:
193         resultados = [funcoes[i](X_reduzido)
                for _ in range(20)]
194         matriz_acc[i][3] = np.mean(resultados
                )
195
196 dados_LLE = LocallyLinearEmbedding(
        n_components=20, n_neighbors=9, max_iter
        =1000)
197 X_reduzido = dados_LLE.fit_transform(X)
198
199
200
201 for i in range (0, 10):
202     if i < 8:
203         resultados = [funcoes[i](X_reduzido,
                Y) for _ in range(20)]
204         matriz_acc[i][4] = np.mean(resultados
                )
205     else:
206         resultados = [funcoes[i](X_reduzido)
                for _ in range(20)]
207         matriz_acc[i][4] = np.mean(resultados
                )
208
209 dados_lap_eig = SpectralEmbedding(
        n_components=20, random_state=1000)
210 X_reduzido = dados_lap_eig.fit_transform(X)
211 explained_variance_ratio = np.var(X_reduzido,
        axis=0) / np.var(X, axis=0).sum()
212
213 print(X_reduzido.shape)
214 print(explained_variance_ratio)
215
216 for i in range (0, 10):
217     if i < 8:
218         resultados = [funcoes[i](X_reduzido,
                Y) for _ in range(20)]
219         matriz_acc[i][5] = np.mean(resultados
                )
220     else:
221         resultados = [funcoes[i](X_reduzido)
                for _ in range(20)]
222         matriz_acc[i][5] = np.mean(resultados
                )
223
224 dados_tsne = TSNE(n_components=3,
        learning_rate='auto', init='random',
        perplexity=10, method = 'barnes_hut')#3,
        2
225 X_reduzido = dados_tsne.fit_transform(X)
226
227 print("Shape do dataset reduzido:",
        X_reduzido.shape)
228 print(dados_PCA.explained_variance_ratio_)
229
230 for i in range (0, 10):
231     if i < 8:
232         resultados = [funcoes[i](X_reduzido,
                Y) for _ in range(20)]
233         matriz_acc[i][6] = np.mean(resultados
                )
234     else:
235         resultados = [funcoes[i](X_reduzido)
                for _ in range(20)]
236         matriz_acc[i][6] = np.mean(resultados
                )
237
238 for i in range (0, 10):
239     if i < 8:
240         resultados = [funcoes[i](X, Y) for _
                in range(20)]
241         matriz_acc[i][7] = np.mean(resultados
                )
242     else:
243         resultados = [funcoes[i](X) for _ in
                range(20)]
244         matriz_acc[i][7] = np.mean(resultados
                )
245
246 nomes_colunas = ["PCA", "LDA", "KPCA", "
        ISOMAP", "LLE", "Lap. Eig.", "t-SNE", "
        Raw data"]
247 nomes_linhas = ["Perceptron", "Reg.
        Log stica", "MLP", "KNN", "Nearest Mean"
        , "Bayesiano", "SVM (linear)", "SVM (rbf)
        ", "K-m dias", "GMM", "M dia"]
248
249 df = pd.DataFrame(matriz_acc[:11], columns=
        nomes_colunas, index=nomes_linhas)
250
251 df.loc["M dia"] = df.mean()
252
253 pd.set_option('display.precision', 3)
254 print(df)
255
256 x_lda = LinearDiscriminantAnalysis(
        n_components=2).fit_transform(X,Y)
257
258 plt.figure(figsize=(6,6))
259 sc = plt.scatter(x_lda [:, 0], x_lda [:, 1],
        c=Y)
260 plt.legend(handles = sc.legend_elements()[0],
         labels=list(range(10)))
261 plt.show()
262
```

```
263 x_isomap = Isomap(n_components=2, n_neighbors
        =9, eigen_solver='auto').fit_transform(X)
264
265 plt.figure(figsize=(6,6))
266 sc = plt.scatter(x_isomap [:, 0], x_isomap
        [:, 1], c=Y)
267 plt.legend(handles = sc.legend_elements()[0],
         labels=list(range(10)))
268 plt.show()
269
270 x_tsne = TSNE(n_components=2, learning_rate='
        auto', init='random', perplexity=10).
        fit_transform(X)
271
272 plt.figure(figsize=(6,6))
273 sc = plt.scatter(x_tsne [:, 0], x_tsne [:,
        1], c=Y)
274 plt.legend(handles = sc.legend_elements()[0],
         labels=list(range(10)))
275 plt.show()
```