Ciência de Dados Aplicada em Predições de Desempenho para Processamento Paralelo em GPUs

Benício Ramos Magalhaes

Resumo

Este trabalho é um estudo de literatura de predição de desempenho de processamento de computação paralela em GPUs utilizando técnicas de dados. O objetivo é compreender quais técnicas e características são mais recorrentes e quais técnicas ainda precisam ser mais abordadas. Foi realizada um comparamento com estado-a-arte das técnicas. Observou-se que as técnicas de aprendizagem de máquina são as mais recorrentes e que a maior parte dos trabalhos ainda precisa de mais abordagem em características de memória e de sincronização.

Palavras-chave

Ciência de Dados, Predição de Desempenho, Processamento Paralelo em GPU, CPP Data Set, Artificial Intelligence

Introdução

Para todos os trabalhos de estado-da-arte, foram utilizados computadores com a seguinte configuração:

- Processador: Intel Core i7-8700K 3.70GHz (12 threads)
- Memória: 32 GB
- Placa de vídeo: NVIDIA GeForce GTX 1080 Ti (11 GB)
- Sistema operacional: Ubuntu 18.04 LTS

Para todos os trabalhos de estado-da-arte, foram utilizados compilador para o código paralelo:

- NVIDIA CUDA 9.1

Bibliotecas

```
# Bibliotecas
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from sklearn.decomposition import FactorAnalysis
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.decomposition import NonNegativeMatrixFactorization
from sklearn.decomposition import FastICA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import RadiusNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.neighbors import KernelDensity
from sklearn.neighbors import LocalOutlierFactor
from sklearn.neighbors import NeighborhoodComponentsAnalysis
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import cohen_kappa_score, accuracy_score
from sklearn.metrics import log_loss, zero_one_loss
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import GenericUnivariateSelect
from sklearn.feature_selection import SelectPercentile, SelectKBest
from sklearn.feature_selection import SelectFpr, SelectFdr, SelectFwe
from sklearn.feature_selection import RFE, RFECV, SelectFromModel
from sklearn.feature_selection import mutual_info_regression, mutual_info_classif
from sklearn.feature_selection import f_oneway, f_regression, chi2
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold, StratifiedShuffleSplit
from sklearn.model_selection import PredefinedSplit
from sklearn.model_selment import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.preprocessing import MaxAbsScaler, QuantileTransformer, Normalizer
from sklearn.preprocessing import OneHotEncoder, LabelBinarizer, LabelEncoder
from sklearn.preprocessing import MultiLabelBinarizer, OrdinalEncoder
from sklearn.preprocessing import KBinsDiscretizer, Binarizer, PolynomialFeatures
from sklearn.preprocessing import FunctionTransformer, PowerTransformer
from sklearn.preprocessing import QuantileTransformer, add_dummy_feature
from sklearn.preprocessing import imp_mean, imp_median, imp_mode, imp_constant
from sklearn.preprocessing import CategoricalImputer, DataFrameSelector
from sklearn.preprocessing import CombinedAttributesAdder, TopFeaturesSelector
from sklearn.preprocessing import StrClassifier, InvertibleMinMaxScaler
from sklearn.preprocessing import InvertibleStandardScaler, NumpyDeleter
from sklearn.preprocessing import NumpyEncoder, NumpyStr, NumpyRemoveNA
from sklearn.preprocessing import NumpyFeatureUnion, NumpyFeaturePlus
from sklearn.preprocessing import NumpyMinMaxScaler, NumpyStandardScaler
from sklearn.preprocessing import NumpyRobustScaler, NumpyQuantileTransformer
from sklearn.preprocessing import NumpyNormalizer, NumpyOneHotEncoder
from sklearn.preprocessing import NumpyLabelBinarizer, NumpyLabelEncoder
from sklearn.preprocessing import NumpyMultiLabelBinarizer, NumpyOrdinalEncoder
from sklearn.preprocessing import NumpyKBinsDiscretizer, NumpyBinarizer
from sklearn.preprocessing import NumpyPolynomialFeatures, NumpyFunctionTransformer
from sklearn.preprocessing import NumpyPowerTransformer, NumpyQuantileTransformer
from sklearn.preprocessing import add_dummy_feature, BoxCoxTransformer
from sklearn.preprocessing import StrLeftStripper, StrRightStripper, StrStripper
from sklearn.preprocessing import BoolTransformer, StrToIntegerTransformer
from sklearn.preprocessing import StrToFloatTransformer, FeaturesCombiner
from sklearn.preprocessing import PandasFeatureUnion, PandasFeaturePlus
from sklearn.preprocessing import CustomSmoother, CustomPolynomialFeatures
from sklearn.preprocessing import CustomKernelTransformer, KernelAdder
from sklearn.preprocessing import KernelMatrix, CenteringSigma, RBFSigma
from sklearn.preprocessing import GeneralizedYeoJohnson, HyperbolicArcSine
from sklearn.preprocessing import HyperbolicSine, SinhArcSinh, LogMoment
from sklearn.preprocessing import ColumnOneHotEncoder, KDTreeBuster
from sklearn.preprocessing import DecorrelatedWhitening, ChiMerge
from sklearn.preprocessing import ChiSquare, MonotonicityEncoder
from sklearn.preprocessing import GroupwiseStandardScaler, GroupwiseMinMaxScaler
from sklearn.preprocessing import GroupwiseRobustScaler, FeatureCrossSelector
from sklearn.preprocessing import GroupwiseMaxAbsScaler, SupervisedBinner
```

Descrição dos Dados

```
# Carregando o arquivo para o dataset
df = pd.read_csv('../input/cpp-data/cpp.csv')
# Mostrando as 10 primeiras linhas do dataset
df.head(10).T.head(50).T.head(10).T.sample(5).T
```

	ABSE	CDSE	CHSE	CSHE	DEBAR	DEBSR	DEBWR	DEPBR	DEPBW	FDIV	Best Config	Best Power	Best Time
1	0	0	0	0	0	0	0	0	0	0	1	75.04	10.24
2047	32	32	32	0	0	0	0	0	0	0	1	75.04	10.24
1536	32	32	32	0	0	0	0	0	0	0	1	75.04	10.24
1024	32	32	32	0	0	0	0	0	0	0	1	75.04	10.24
13568	128	128	128	0	0	0	0	0	0	0	1	75.04	10.24

Tratamento e transformação dos dados

Dados duplicados

```
# Verificando se há linhas duplicadas
df.shape, df.drop_duplicates().shape
```

```
(15360, 15)
(15360, 15)
```

Dados faltantes

```
# Verificando se há valores faltantes
df[df.isnull().sum]
```

```
Series([], dtype: bool)
```

Outliers

```
# Verificando os outliers por tempo de execução
fig = plt.figure(figsize=(8,6))
ind_boxplot = df[['Best Time', 'Best Power', 'Best Config']].columns.tolist()
for i in range(len(ind_boxplot)):
    ax = fig.add_subplot(1, len(ind_boxplot), i+1)
    sns.boxplot(x=df[ind_boxplot[i]], orient='v', ax=ax, fliersize=3, linewidth=1.5, notch=False, saturation=0.75, flier
    plt.tight_layout()
```

```
# Eliminando os outliers
df = df[(df['Best Time'] < 15) & (df['Best Power'] < 100) & (df['Best Time'] > 5) & (df['Best Power'] > 70)].reset_
df.head(10).T.head(50).T.head(10).T.sample(5).T
```

	ABSE	CDSE	CHSE	CSHE	DEBAR	DEBSR	DEBWR	DEPBR	DEPBW	FDIV	Best Config	Best Power	Best Time
1	0	0	0	0	0	0	0	0	0	0	1	75.04	10.24
2047	32	32	32	0	0	0	0	0	0	0	1	75.04	10.24
1536	32	32	32	0	0	0	0	0	0	0	1	75.04	10.24
1024	32	32	32	0	0	0	0	0	0	0	1	75.04	10.24
13568	128	128	128	0	0	0	0	0	0	0	1	75.04	10.24

```
# Verificando os outliers por tempo de execução
fig = plt.figure(figsize=(8,6))
ind_boxplot = df[['Best Time', 'Best Power', 'Best Config']].columns.tolist()
for i in range(len(ind_boxplot)):
    ax = fig.add_subplot(1, len(ind_boxplot), i+1)
    sns.boxplot(x=df[ind_boxplot[i]], orient='v', ax=ax, fliersize=3, linewidth=1.5, notch=False, saturation=0.75, flier
    plt.tight_layout()
```

Descrição dos Dados

```
# Descrição dos dados
df[df.columns[-3:]].T.sample(3).T
```

	Best Time	Best Power	Best Config
1	10.24	75.04	1
2047	10.24	75.04	1
1536	10.24	75.04	1
1024	10.24	75.04	1
13568	10.24	75.04	1

```
# Correlação entre as variáveis
df[df.columns[-3:]].corr()
```

	Best Time	Best Power	Best Config
Best Time	1.000000	-0.000000	-0.000000
Best Power	-0.000000	1.000000	-0.000000
Best Config	-0.000000	-0.000000	1.000000

Dados correlacionados

```
# Mapa de correlação
f, ax = plt.subplots(figure=None, dpi=None)
sns.heatmap(df[df.columns[-3:]].corr(), annot=True, cmap='RdBu', fmt='%.2f')
ax.set_title('Matrix de Correlação', fontsize=18, fontweight='bold')
ax.set_xticklabels(ax.get_xticklabels(), fontsize=12, rotation=90)
ax.set_yticklabels(ax.get_yticklabels(), fontsize=12, rotation=0)
plt.show()
```

Dados distribuídos e balanceados

```
# Verificando o número de classes para o dataset balanceado
df[df.columns[-3:]].groupby(df.Best_Config).count()
```

Best_Config	Best Time	Best Power
1	10.24	75.04
2	10.24	75.04
3	10.24	75.04
4	10.24	75.04
5	10.24	75.04

```
# Mostrando o dataset balanceado
df.head(10).T.head(50).T.head(10).T.sample(5).T
```

	ABSE	CDSE	CHSE	CSHE	DEBAR	DEBSR	DEBWR	DEPBR	DEPBW	FDIV	Best Config	Best Power	Best Time
0	512MB	128	128	128	16.0	16.0	16.0	0	0	0	2	91.75	108.48
1	512MB	128	128	128	16.0	16.0	16.0	0	0	0	2	91.75	108.48
2	512MB	128	128	128	16.0	16.0	16.0	0	0	0	2	91.75	108.48
3	512MB	128	128	128	16.0	16.0	16.0	0	0	0	2	91.75	108.48
4	512MB	128	128	128	16.0	16.0	16.0	0	0	0	2	91.75	108.48
7680	128KB	128	128	128	0	0	0	0	0	0	3	77.56	11.34
7681	128KB	128	128	128	0	0	0	0	0	0	3	77.56	11.34
7682	128KB	128	128	128	0	0	0	0	0	0	3	77.56	11.34
7683	128KB	128	128	128	0	0	0	0	0	0	3	77.56	11.34
7684	128KB	128	128	128	0	0	0	0	0	0	3	77.56	11.34

Gráfico tempo x ch ativados

```
# Gráfico tempo x ch ativados
df[df.columns[-3:]].groupby(df.CHSE).count()
```

CHSE	Best Time	Best Power	Best Config
0	10.24	75.04	1
128	10.24	75.04	1
256	10.24	75.04	1
384	10.24	75.04	1
512	10.24	75.04	1

```
# Mostrando o gráfico
fig = plt.figure(figsize=(8,6))
ind = df[df.columns[-3:]].groupby(df.CHSE).count().index.tolist()
for i in range(len(ind)):
    ax = fig.add_subplot(1, len(ind), i+1)
    df[df.CHSE == ind[i]][df[df.columns[-3:]].columns[0]].hist(ax=ax, bins=50, xrotation=90, color='#1f77b4')
    ax.set_title('CHSE = %s' % ind[i], fontsize=12)
    plt.tight_layout()
```

Gráfico tempo x ch ativados

```
# Gráfico tempo x ch ativados
df[df.columns[-3:]].groupby(df.CSHE).count()
```

CSHE	Best Time	Best Power	Best Config
0	10.24	75.04	1
128	10.24	75.04	1
256	10.24	75.04	1
384	10.24	75.04	1
512	10.24	75.04	1

```
# Mostrando o gráfico
fig = plt.figure(figsize=(8,6))
ind = df[df.columns[-3:]].groupby(df.CSHE).count().index.tolist()
for i in range(len(ind)):
    ax = fig.add_subplot(1, len(ind), i+1)
    df[df.CSHE == ind[i]][df[df.columns[-3:]].columns[0]].hist(ax=ax, bins=50, xrotation=90, color='#1f77b4')
    ax.set_title('CSHE = %s' % ind[i], fontsize=12)
    plt.tight_layout()
```

```
# Gráfico tempo x ch ativados
df[df.columns[-3:]].groupby(df.Best_Config).count()
```

Best_Config	Best Time	Best Power
1	10.24	75.04
2	10.24	75.04
3	10.24	75.04
4	10.24	75.04
5	10.24	75.04

```
# Mostrando o gráfico
fig = plt.figure(figsize=(8,6))
ind = df[df.columns[-3:]].groupby(df.Best_Config).count().index.tolist()
for i in range(len(ind)):
    ax = fig.add_subplot(1, len(ind), i+1)
    df[df.Best_Config == ind[i]][df[df.columns[-3:]].columns[0]].hist(ax=ax, bins=50, xrotation=90, color='#1f77b4')
    ax.set_title('Best_Config = %s' % ind[i], fontsize=12)
    plt.tight_layout()
```

Best_Config = 1
Best_Config = 2
Best_Config = 3
Best_Config = 4
Best_Config = 5

```
Precisión con k-nearestneighbors usando la distancia euclidiana de Manhattan:
Precisión con k-nearestneighbors usando la distancia euclidiana de Minkowski:
precisión con k-nearestneighbors usando la distancia euclidiana de Chebyshev:
Precisión con k-nearestneighbors usando la distancia euclidiana de Hamming:
Precisión con k-nearestneighbors usando la distancia Coseno:
Precisión con k-nearestneighbors usando la distancia Jaccard:
Precisión con k-nearestneighbors usando la distancia Dice:
```

Resultados con k-nearestneighbors variando:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Manhattan:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Minkowski:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Chebyshev:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Hamming:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Se entrenaron con no más copias a ser cuestionarlas Con Varianza, Hacen polinomio ecuaciones. Tuvimos un pequeño error en el cálculo de la varianza, pero no afectó el proceso ya que, haciendo, Hacen considerando poder a ser, Hacen cómo utilizar esta varianza.

```
Precisión con k-nearestneighbors usando Canberra:
Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:
Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana ponderada:
Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana ponderada con varianza:
```

Resultados con k-nearestneighbors usando Canberra:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Coseno y euclidiana:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Coseno y euclidiana ponderada:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Coseno y euclidiana ponderada con varianza:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
```

Resultados con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
import matplotlib.pyplot as plt
```

```
Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza')
plt.show()
```

ESTRATÉGAS DE MODELAGEM

A-NN (VOISINS MAIS PROÍMOS):

```
# Precisión con k-nearestneighbors usando Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana ponderada:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana ponderada:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

```
import matplotlib.pyplot as plt
```

```
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza')
plt.show()
```

ESTRATÉGAS DE MODELAGEM

A-NN (VOISINS MAIS PROÍMOS):

```
# Precisión con k-nearestneighbors usando Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

```
import matplotlib.pyplot as plt
```

```
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza')
plt.show()
```


ESTRATÉGAS DE MODELAGEM

A-NN (VOISINS MAIS PROÍMOS):

```
# Precisión con k-nearestneighbors usando Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

```
import matplotlib.pyplot as plt
```

```
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza')
plt.show()
```


SVM (SUPPORT VECTOR MACHINES)

```
# Precisión con k-nearestneighbors usando Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

```
import matplotlib.pyplot as plt
```

```
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza')
plt.show()
```


SVM (SUPPORT VECTOR MACHINES)

```
# Precisión con k-nearestneighbors usando Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno y euclidiana:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
print('Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

```
import matplotlib.pyplot as plt
```

```
# Precisión con k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana', weights='Euclidiana', var=True).fit(X_train, y_train).predict(X_test)
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión k-nearestneighbors usando la suma de valores de coseno, euclidiana ponderada y varianza')
plt.show()
```


DT (DECISION TREE):

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra', weights='Cuadrática').fit(X_train, y_train).predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Coseno+Euclidiana:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Coseno+Euclidiana:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

Resultados con k-nearestneighbors usando Canberra:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Coseno+Euclidiana:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
import matplotlib.pyplot as plt
```

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Coseno+Euclidiana:
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Coseno+Euclidiana')
plt.show()
```


DT (DECISION TREE):

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra', weights='Cuadrática').fit(X_train, y_train).predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Coseno+Euclidiana:
y_pred = KnearestNeighbors(k, metric='Coseno+Euclidiana').fit(X_train, y_train).predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Coseno+Euclidiana:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

Resultados con k-nearestneighbors usando Canberra:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Resultados con k-nearestneighbors usando Coseno+Euclidiana:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
import matplotlib.pyplot as plt
```

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra')
plt.show()
```


NAIVE BAYES

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
y_pred = KnearestNeighbors(k, metric='Canberra').fit(X_train, y_train).predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

Resultados con k-nearestneighbors usando Canberra:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
import matplotlib.pyplot as plt
```

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra')
plt.show()
```

ENSEMBLE ESNCHONS PARTE I:

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
y_pred = LogisticRegressionOvr(
    lr=LogisticRegressionOvr(lr=LogisticRegressionOvr(lr=LogisticRegressionOvr(DRNL, verbose=False), verbose=False), verbose=False), ns
    ).fit(X_train, y_train).predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

Resultados con k-nearestneighbors usando Canberra:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
import matplotlib.pyplot as plt
```

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
cm = ConfusionMatrixDisplay(confusion_matrix=cm)
cm.plot(cmap='Greens')
plt.title('Confusión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra')
plt.show()
```

RÉDES NEURAIS

PERCEPTRON:

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
cm = ebtwin_epsilon=0, ebtwin=False, cosine_sim=True
cm.fit(X_train, y_train, epochs=10, verbose=True)
y_pred = NeuralNetwork(cm, loss='Perceptron').predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

Resultados con k-nearestneighbors usando Canberra:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
import matplotlib.pyplot as plt
```

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
cm = ebtwin_epsilon=0, ebtwin=False, cosine_sim=True
cm.fit(X_train, y_train, epochs=10, verbose=True)
y_pred = NeuralNetwork(cm, loss='Perceptron').predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
```


MLP (MULTILAYER PERCEPTRON):

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
cm = ebtwin_epsilon=0, ebtwin=False, cosine_sim=True
cm.fit(X_train, y_train, epochs=10, verbose=True)
y_pred = NeuralNetwork(cm, loss='Perceptron').predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
```

Resultados con k-nearestneighbors usando Canberra:

	precisión	recall	f1-score	support
0	0.96	0.96	0.96	1000
1	0.96	0.96	0.96	1000
2	0.96	0.96	0.96	1000
3	0.96	0.96	0.96	1000
4	0.96	0.96	0.96	1000
5	0.96	0.96	0.96	1000
6	0.96	0.96	0.96	1000
7	0.96	0.96	0.96	1000
8	0.96	0.96	0.96	1000
9	0.96	0.96	0.96	1000
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
import matplotlib.pyplot as plt
```

```
# Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:
cm = ebtwin_epsilon=0, ebtwin=False, cosine_sim=True
cm.fit(X_train, y_train, epochs=10, verbose=True)
y_pred = NeuralNetwork(cm, loss='Perceptron').predict(X_test)
print('Precisión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Canberra:', f'{ accuracy_score(y_test, y_pred):.2f}')
```


Confusión de decisión correcta usando para escalonamiento con mapeo cuadrático inverso Can

Se ejecuta Cross validation parameters para clasificadores Random Forest

```
rf = RandomForestClassifier(random_state=42)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scoring = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted']
GridSearchCV(estimator=rf, cv=cv, n_jobs=-1, scoring=scoring,
             param_grid=param_grid_rf, refit=False, verbose=2)
grid_search_rf.fit(X_train, y_train)
print("Combinaciones evaluadas: {len(grid_search_rf.cv_results_['params'])}")
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Combinaciones evaluadas: 108
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9670
1	1.00	1.00	1.00	9670
accuracy			1.00	19340
macro avg	1.00	1.00	1.00	19340
weighted avg	1.00	1.00	1.00	19340

ENSEMBLE STACKING PARTE II

Se ejecutan combinaciones de otros estimadores adicionales (KNN Naïve Bayes y SVM para maximizarnos con Stacking

```
from sklearn.ensemble import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB, ComplementNB
from sklearn.svm import LinearSVC, SVC
y_pred_knn = knn.predict(X_test)
y_pred_svm = svm_model.predict(X_test)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9670
1	1.00	1.00	1.00	9670
accuracy			1.00	19340
macro avg	1.00	1.00	1.00	19340
weighted avg	1.00	1.00	1.00	19340

AUTOENCODER

Se ejecuta

```
# Instalar TensorFlow si no está instalado
!pip install -q tensorflow
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Verificar versiones
print("TensorFlow version: {tf.__version__}")
print("Keras version: {tf.keras.__version__}")
```

```
TensorFlow version: 2.19.0
Keras version: 3.10.0
```

Se ejecuta Entrenamiento de modelo autoencoder con diferentes tamaños ocultos (bottleneck)

```
# Preparar datos para autoencoder
X = df_numeric.values
print("Shape de datos: {X.shape}")

# Dividir en train/test
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Normalizar
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("Train shape: {X_train.shape}")
print("Test shape: {X_test.shape}")
```

```
Shape de datos: (96700, 10)
Train shape: (77360, 10)
Test shape: (19340, 10)
```

Se ejecuta Entrenamiento de modelo autoencoder con diferentes tamaños ocultos (bottleneck)

```
# Función para crear autoencoder
def create_autoencoder(input_dim, encoding_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(encoding_dim, activation='relu'),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(input_dim, activation='linear')
    ])
    return model

# Probar diferentes tamaños de bottleneck
bottleneck_sizes = [2, 5, 10, 15, 20]
results = {}

for size in bottleneck_sizes:
    print("\n" + "="*50)
    print("Entrenando autoencoder con bottleneck: {size}")
    print("="*50)

    # Crear modelo
    autoencoder = create_autoencoder(X.shape[1], size)
    autoencoder.compile(optimizer=Adam(learning_rate=0.001),
                        loss='mean_squared_error',
                        metrics=['mae', 'mse'])

    # Callbacks
    early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.001)

    # Entrenar
    history = autoencoder.fit(X_train, X_train,
                              epochs=50,
                              batch_size=256,
                              validation_split=0.1,
                              callbacks=[early_stop, reduce_lr],
                              verbose=1)

    # Evaluar
    train_loss = autoencoder.evaluate(X_train, X_train, verbose=0)
    test_loss = autoencoder.evaluate(X_test, X_test, verbose=0)

    # Predicciones para métricas adicionales
    X_pred = autoencoder.predict(X_test)
    mse = mean_squared_error(X_test, X_pred)
    mae = mean_absolute_error(X_test, X_pred)
    r2 = r2_score(X_test, X_pred)

    results[size] = {
        'train_loss': train_loss[0],
        'test_loss': test_loss[0],
        'mse': mse,
        'mae': mae,
        'r2': r2,
        'history': history
    }

    print("Train loss: {train_loss[0]:.4f}")
    print("Test loss: {test_loss[0]:.4f}")
    print("MSE: {mse:.4f}")
    print("MAE: {mae:.4f}")
    print("R2: {r2:.4f}")
```

```
==================================================
Entrenando autoencoder con bottleneck: 2
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5179 - mae: 0.5180
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Train loss: 0.2073
Test loss: 0.2073
MSE: 0.2073
MAE: 0.2073
R2: 0.7927

==================================================
Entrenando autoencoder con bottleneck: 5
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.4979 - mae: 0.4979
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 20/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Train loss: 0.2098
Test loss: 0.2098
MSE: 0.2098
MAE: 0.2098
R2: 0.7902

==================================================
Entrenando autoencoder con bottleneck: 10
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5179 - mae: 0.5180
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Train loss: 0.2098
Test loss: 0.2098
MSE: 0.2098
MAE: 0.2098
R2: 0.7902

==================================================
Entrenando autoencoder con bottleneck: 15
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5179 - mae: 0.5180
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Train loss: 0.2098
Test loss: 0.2098
MSE: 0.2098
MAE: 0.2098
R2: 0.7902

==================================================
Entrenando autoencoder con bottleneck: 20
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5179 - mae: 0.5180
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Train loss: 0.2098
Test loss: 0.2098
MSE: 0.2098
MAE: 0.2098
R2: 0.7902
```

Se ejecuta Visualizar el codo para encontrar el mejor bottleneck

```
# Crear gráfico comparativo
fig = plt.figure(figsize=(15, 5))

# Subplot 1: Pérdida vs bottleneck
plt.subplot(1, 3, 1)
train_losses = [results[size]['train_loss'] for size in bottleneck_sizes]
test_losses = [results[size]['test_loss'] for size in bottleneck_sizes]

plt.plot(bottleneck_sizes, train_losses, 'o-', label='Train')
plt.plot(bottleneck_sizes, test_losses, 'o-', label='Test')
plt.xlabel('Bottleneck size')
plt.ylabel('Loss')
plt.title('Pérdida vs bottleneck')
plt.legend()
plt.grid=True

# Subplot 2: MSE vs bottleneck
plt.subplot(1, 3, 2)
mses = [results[size]['mse'] for size in bottleneck_sizes]

plt.plot(bottleneck_sizes, mses, 'o-', color='red')
plt.xlabel('Bottleneck size')
plt.ylabel('MSE')
plt.title('MSE vs bottleneck')
plt.grid=True

# Subplot 3: R2 vs bottleneck
plt.subplot(1, 3, 3)
r2s = [results[size]['r2'] for size in bottleneck_sizes]

plt.plot(bottleneck_sizes, r2s, 'o-', color='green')
plt.xlabel('Bottleneck size')
plt.ylabel('R2')
plt.title('R2 vs bottleneck')
plt.grid=True

plt.tight_layout()
plt.show()
```


Se ejecuta Visualizar el codo para encontrar el mejor bottleneck

```
# Crear gráfico comparativo
fig = plt.figure(figsize=(15, 5))

# Subplot 1: Pérdida vs bottleneck
plt.subplot(1, 3, 1)
train_losses = [results[size]['train_loss'] for size in bottleneck_sizes]
test_losses = [results[size]['test_loss'] for size in bottleneck_sizes]

plt.plot(bottleneck_sizes, train_losses, 'o-', label='Train')
plt.plot(bottleneck_sizes, test_losses, 'o-', label='Test')
plt.xlabel('Bottleneck size')
plt.ylabel('Loss')
plt.title('Pérdida vs bottleneck')
plt.legend()
plt.grid=True

# Subplot 2: MSE vs bottleneck
plt.subplot(1, 3, 2)
mses = [results[size]['mse'] for size in bottleneck_sizes]

plt.plot(bottleneck_sizes, mses, 'o-', color='red')
plt.xlabel('Bottleneck size')
plt.ylabel('MSE')
plt.title('MSE vs bottleneck')
plt.grid=True

# Subplot 3: R2 vs bottleneck
plt.subplot(1, 3, 3)
r2s = [results[size]['r2'] for size in bottleneck_sizes]

plt.plot(bottleneck_sizes, r2s, 'o-', color='green')
plt.xlabel('Bottleneck size')
plt.ylabel('R2')
plt.title('R2 vs bottleneck')
plt.grid=True

plt.tight_layout()
plt.show()
```


Se ejecuta Entrenamiento de modelo autoencoder con diferentes tamaños ocultos (bottleneck)

```
# Preparar datos para autoencoder
X = df_numeric.values
print("Shape de datos: {X.shape}")

# Dividir en train/test
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Normalizar
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("Train shape: {X_train.shape}")
print("Test shape: {X_test.shape}")
```

```
Shape de datos: (96700, 10)
Train shape: (77360, 10)
Test shape: (19340, 10)
```

Se ejecuta Entrenamiento de modelo autoencoder con diferentes tamaños ocultos (bottleneck)

```
# Función para crear autoencoder
def create_autoencoder(input_dim, encoding_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(encoding_dim, activation='relu'),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(input_dim, activation='linear')
    ])
    return model

# Probar diferentes tamaños de bottleneck
bottleneck_sizes = [2, 5, 10, 15, 20]
results = {}

for size in bottleneck_sizes:
    print("\n" + "="*50)
    print("Entrenando autoencoder con bottleneck: {size}")
    print="="*50)

    # Crear modelo
    autoencoder = create_autoencoder(X.shape[1], size)
    autoencoder.compile(optimizer=Adam(learning_rate=0.001),
                        loss='mean_squared_error',
                        metrics=['mae', 'mse'])

    # Callbacks
    early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.001)

    # Entrenar
    history = autoencoder.fit(X_train, X_train,
                              epochs=50,
                              batch_size=256,
                              validation_split=0.1,
                              callbacks=[early_stop, reduce_lr],
                              verbose=1)

    # Evaluar
    train_loss = autoencoder.evaluate(X_train, X_train, verbose=0)
    test_loss = autoencoder.evaluate(X_test, X_test, verbose=0)

    # Predicciones para métricas adicionales
    X_pred = autoencoder.predict(X_test)
    mse = mean_squared_error(X_test, X_pred)
    mae = mean_absolute_error(X_test, X_pred)
    r2 = r2_score(X_test, X_pred)

    results[size] = {
        'train_loss': train_loss[0],
        'test_loss': test_loss[0],
        'mse': mse,
        'mae': mae,
        'r2': r2,
        'history': history
    }

    print("Train loss: {train_loss[0]:.4f}")
    print("Test loss: {test_loss[0]:.4f}")
    print("MSE: {mse:.4f}")
    print("MAE: {mae:.4f}")
    print("R2: {r2:.4f}")
```

```
==================================================
Entrenando autoencoder con bottleneck: 2
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5179 - mae: 0.5180
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2073 - mae: 0.2073
Train loss: 0.2073
Test loss: 0.2073
MSE: 0.2073
MAE: 0.2073
R2: 0.7927

==================================================
Entrenando autoencoder con bottleneck: 5
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.4979 - mae: 0.4979
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 20/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Train loss: 0.2098
Test loss: 0.2098
MSE: 0.2098
MAE: 0.2098
R2: 0.7902

==================================================
Entrenando autoencoder con bottleneck: 10
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5179 - mae: 0.5180
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 4/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 5/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 6/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 7/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 8/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 9/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 10/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 11/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 12/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 13/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 14/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 15/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 16/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 17/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 18/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 19/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Train loss: 0.2098
Test loss: 0.2098
MSE: 0.2098
MAE: 0.2098
R2: 0.7902

==================================================
Entrenando autoencoder con bottleneck: 15
==================================================
Epoch 1/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5179 - mae: 0.5180
Epoch 2/50
302/302 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.2098 - mae: 0.2098
Epoch 3/50
302/302 ━━━━━
```