CLASIFICACIÓN BINARIA DE 2 ESPECIES DE SEMILLAS DE CALABAZA

PRESENTA:

ING. ALEJANDRO NOEL HERNÁNDEZ GUTIÉRREZ
ING. SOFIA VILCHIS SANCHEZ
LIC. ALAN OMAR TOPETE SALAZAR

ITESO

Instituto Tecnológico de Estudios Superiores de Occidente

MAESTRÍA EN CIENCIA DE DATOS

INVESTIGACIÓN DESARROLLO E INNOVACIÓN II

IMPARTE: DR. FERNANDO IGNACIO BECERRA LÓPEZ

NOVIEMBRE 2022

Introducción

El objetivo del presente proyecto es estudiar el comportamiento de un conjunto de datos cuya concepción es clasificar semillas de calabaza de las especies Çerçevelik y Ürgüp Sivrisi con base en sus características morfológicas.

Dentro del **Marco Teórico**, se presenta el conjunto de datos y se explica brevemente el procesamiento de estos, en donde fueron eliminadas las columnas con alta correlación y se procesó la información para obtener un conjunto de datos normalizado para los modelos con una distribución normal estándar.

En el mismo capítulo se ahondan de forma breve y conceptual los algoritmos inteligentes de clasificación de datos, que en el caso de este proyecto se utiliza principalmente el perceptrón multicapa y de forma complementaria, con el fin de obtener un punto de comparación, se entrena una regresión logística y una máquina de soporte vectorial, que también son algoritmos de clasificación.

En el capítulo **Implementación** se utilizan los 3 algoritmos descritos respondiendo principalmente, pero no limitando, a las siguientes preguntas: ¿Qué abona al objetivo del proyecto este algoritmo? ¿Qué valor agrega en comparativa con los demás algoritmos/modelos utilizados? ¿Qué se busca demostrar con este modelo?

Además, se muestran los resultados obtenidos al correr el algoritmo en un formato secuencial.

En el capítulo **Conclusiones** se demuestra que el objetivo del proyecto se cumple y se realiza un resumen comparativo entre los modelos concluyendo cuál resulta ser mejor para este conjunto de datos, qué características afectaron más la efectividad del modelo y por qué.

Marco Teórico

Descripción del conjunto de datos

En esta sección se presenta la explicación del conjunto de datos disponible en:

https://github.com/AlexNoelHdz/Pumpkin_analysis

En color morado y azul, se observan las variables que tienen correlación alta. En verde las elegidas para correr los modelos del presente documento. Este tema se trata a profundidad en la sección **Pre - procesamiento de datos**.

Tabla 1. Descripción del conjunto de datos.

No.	Cor.	Cor.	Nombre	Explicación
0		-	Area (A)	Número de pixeles dentro de los bordes de una semilla de calabaza.
1			Perimeter(p)	La circunferencia en pixeles de una semilla de calabaza.
2			Major Axis Length (Maj.AL)	Distancia del eje mayor dentro del área de la semilla.
3			Minor Axis Length (Min.AL)	Distancia del eje menor dentro del área de la semilla.
4		-	Convex Area (CA)	Conjunto de pares ordenados de pixeles que forman una región convexa dentro de la semilla.
5		-	Equiv Diameter (ED)	Multiplicación del área de la semilla de calabaza por cuatro, dividido por pi, a todo esto, se le saca raíz cuadrada.
6	-		Eccentricity (e)	Parámetro que determina el grado de desviación de una sección cónica con respecto a la circunferencia.
7			Solidity (s)	Convexidad y condición convexa de las semillas.
8			Extent (E)	Proporción (ratio) del área delimitada.
9	-		Roundness	Redondez de las semillas sin considerar la distorsión del borde.
10	-		Aspect Radio	Relación de aspecto de las semillas.
11	-		Compactness (C)	Área de la semilla relativa al área del círculo con la misma circunferencia.

A continuación, se presenta un par de imágenes que ilustran algunas de las variables ya vistas:

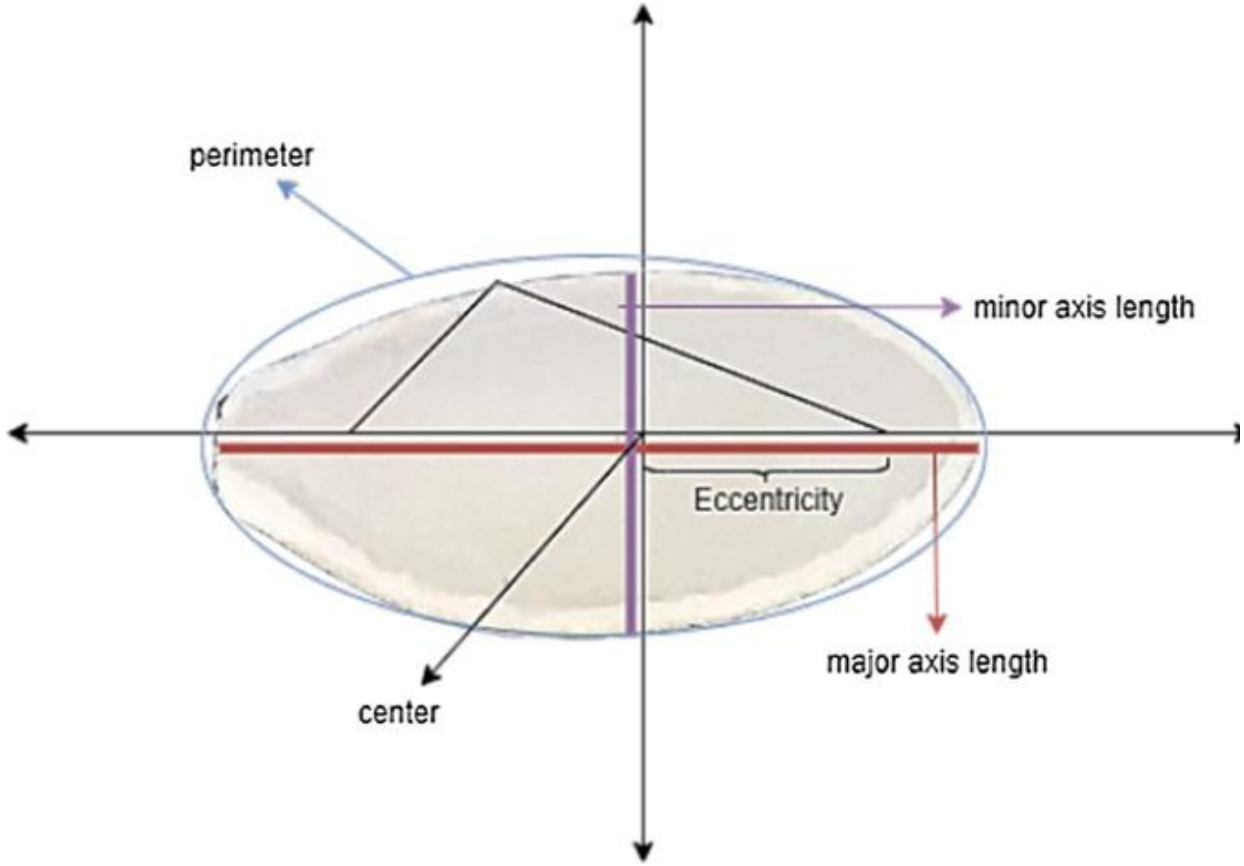
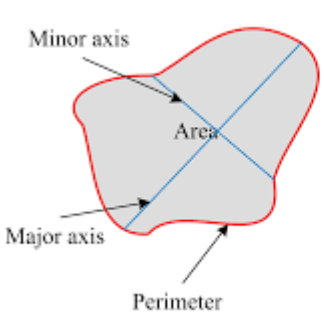


Ilustración 1 Pumpkin Data Descrription 1



1) Area (A)
2) Equivalent diameter (Ed)
3) Major axis length (LM)
4) Minor axis length (Lm)
5) Perimeter (Pr)
6) Roundness (Rd) = $\frac{4.\pi.A}{(Pr)^2}$
7) Elongation (El) = $\frac{LM}{Lm}$
8) Compactness(Cp) = $\sqrt{\frac{4.A}{\pi}}$

Ilustración 2 Pumkin Data Description 2

Una de las variables consideradas es el Área convexa. Para entenderla más a detalle el lector puede revisar la siguiente imagen.

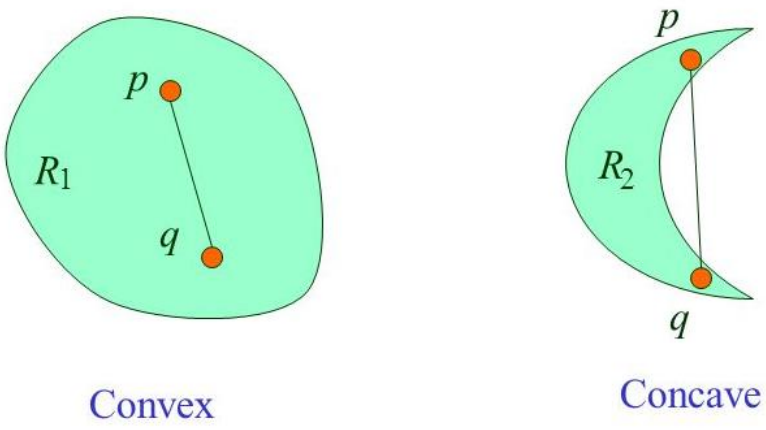


Ilustración 3 Explicación conjunto convexo.

A una región plana **R** se le llama región convexa si y solo si para cualquier par ordenado de pixeles (**p, q)** (en este caso puntos), la línea del segmento pq se encuentran completamente en **R.**

Accuracy Score

El *accuracy score* es una evaluación métrica que mide el número de predicciones correctas hechas por el modelo con relación al número total de predicciones.

Perceptrón multicapa

Las RNA de tipo Perceptrón Multicapa (PM) se encuentran entre las arquitecturas de red más poderosas y populares. Están formadas por una capa de entrada, un número arbitrario de capas ocultas, y una capa de salida. Cada una de las neuronas ocultas o de salida recibe una entrada de las neuronas de la capa previa (conexiones hacia atrás), pero no existen conexiones laterales entre las neuronas dentro de cada capa. La capa de entrada contiene tantas neuronas como categorías correspondan a las variables independientes que se desean representar. La capa de salida corresponde a la variable respuesta, que en este caso es una variable categórica (Longoni, 2010).

Según el artículo *Definición de una red neuronal para clasificación por medio de un programa evolutivo* de la revista mexicana de ingeniería biomédica, “Se ha demostrado que los PM con una capa oculta en su arquitectura pueden separar satisfactoriamente las clases involucradas en un problema dado; sin embargo el número de nodos ocultos es desconocido ya que no existe un método para definirlos y depende mucho del problema a resolver”, por lo que en el presente trabajo de investigación para el algoritmo de perceptrón

multicapa la métrica de éxito es tener el mejor accuracy con la menor cantidad de neuronas probado de manera empírica.

Máquina de soporte vectorial

Las Máquinas de vectores de soporte (SVM), introducidas por Vapnik et. al. en la década de 1990, son una familia de algoritmos para aprender funciones discriminantes de dos clases a partir de un conjunto de ejemplos de entrenamiento (Mammone, 2009).

Regresión logística

En cuanto a la regresión logística, el objetivo primordial que resuelve esta técnica es el de cuantificar cómo influye en la probabilidad de aparición de un suceso, habitualmente dicotómico, la presencia o no de diversos factores y el valor o nivel de estos. También puede ser usada para estimar la probabilidad de aparición de cada una de las posibilidades de un suceso con más de dos categorías (Peláez, 2016).

Pre - procesamiento de datos

Datos originales:

Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	Eccentricity	Solidity	Extent	Roundnes	Aspect_Ration	Compactness	Class
56276	888.242	3.261.485	2.202.388	56831	2.676.805	0.7376	0.9902	0.7453	0.8963	14.809	0.8207	Çerçevelik
76631	1.068.146	4.171.932	2.342.289	77280	3.123.614	0.8275	0.9916	0.7151	0.844	17.811	0.7487	Çerçevelik
71623	1.082.987	4.358.328	2.110.457	72663	3.019.822	0.8749	0.9857	0.74	0.7674	20.651	0.6929	Çerçevelik
66458	992.051	3.815.638	2.225.322	67118	2.908.899	0.8123	0.9902	0.7396	0.8486	17.146	0.7624	Çerçevelik
66107	998.146	3.838.883	2.204.545	67117	2.901.207	0.8187	0.985	0.6752	0.8338	17.413	0.7557	Çerçevelik

Tabla 2. Datos Originales.

Datos disponibles en https://github.com/AlexNoelHdz/Pumpkin_analysis

El primer paso en el procesamiento de datos es entender qué representa cada variable y eliminar las observaciones que contengan datos nulos.

Después de la limpieza de datos, se obtiene un mapa de calor (presentado a continuación) que indica la correlación entre variables.

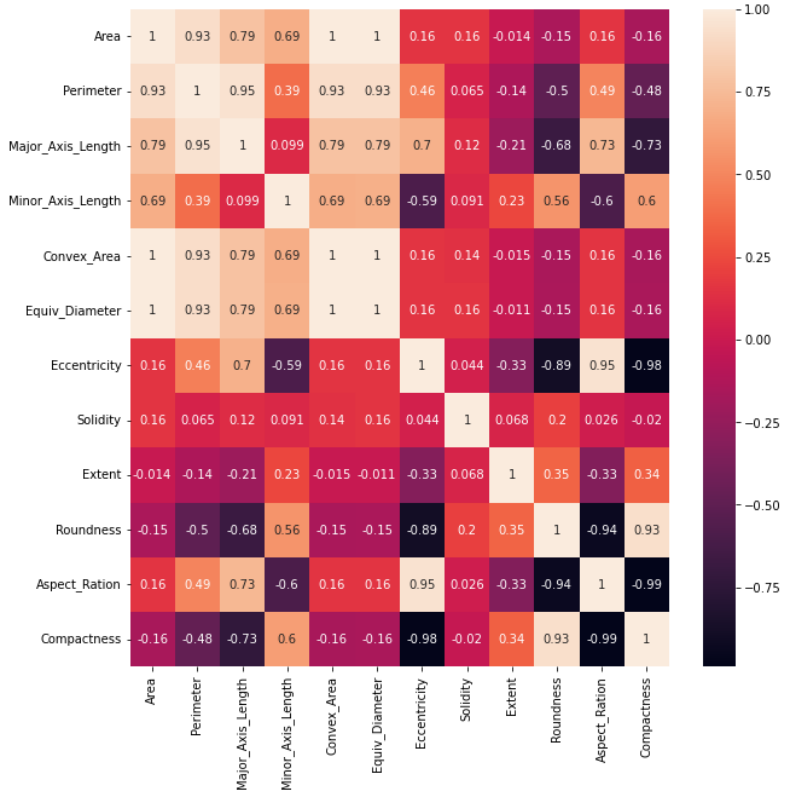


Ilustración 4 Mapa de correlación

Las clases *compactness*, *aspect ratio*, *eccentricity* y *roundness* presentan correlaciones de 0.99, 0.98, 0.94 entre ellas, por lo que se elige conservar solo aspect ratio, que representa correctamente la característica estudiada.

Las variables de *area*, *convex area* y *equiv diameter* también presentan correlación alta entre ellas con valores que rondan el 0.99, por lo que se elige conservar solo área, que representa correctamente la característica estudiada (la base de datos sin las columnas correlacionadas recibe en nombre de sin_correlacion).

Posteriormente se normalizaron los datos usando distribución normal estándar (entre –3,3).

Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Solidity	Extent	Aspect_Ration
-1.7847	-2.21575	-2.32022	-0.238536	0.202812	0.855406	-1.77506
-0.29478	-0.568804	-0.700916	0.362089	0.603626	0.359523	-0.824864
-0.66135	-0.43294	-0.369395	-0.633215	-1.08552	0.76838	0.0740588
-1.03941	-1.26542	-1.33462	-0.140076	0.202812	0.761812	-1.03535
-1.0651	-1.20963	-1.29327	-0.229276	-1.28593	-0.295634	-0.95084
-0.546578	-0.813104	-0.903319	0.241758	0.0024048	0.382511	-0.912224
-0.535818	-1.00906	-1.14452	0.547577	0.97581	0.418635	-1.25787
-0.802694	-0.743089	-0.624539	-0.60209	-0.026224	-0.321906	-0.1627
1.10299	0.927637	0.560576	1.09536	-0.255261	-1.22172	-0.31463

Tabla 3. Datos Normalizados.

Datos disponibles en https://github.com/AlexNoelHdz/Pumpkin_analysis

En este punto, es conveniente utilizar un análisis de componentes principales (PCA) para el caso de estudio, dado que los modelos elegidos en este caso de estudio (SVM y Red neuronal) pueden implicar un costo computacional considerable conforme aumente el número de observaciones.

Se realizo el PCA con el conjunto de datos sin las variables correlacionadas utilizando únicamente las 7 más relevantes (sin_correlacion).

Posterior al PCA se escogen solo ciertos componentes para optimizar, revisando la varianza acumulada, implementando dos opciones:

a) Solo los primero 4 componentes (pca_sincorrelacion_n4), ya que con ellos se llega a una varianza acumulada de 99.8%.

b) Con 5 componentes (pca_sincorrelacion_n5), ya que se obtiene 99.9% de varianza acumulada.

	0
0	46
1	74.3
2	88.5
3	99.8
4	99.9
5	100
6	100

Tabla 4. Porcentaje de varianza acumulada del PCA.

Teniendo 4 configuraciones de la base de datos:

1. La original (base_original)
2. Sin las columnas correlacionadas (sin_correlacion)
3. PCA con 4 componentes (pca_sincorrelacion_n4)
4. PCA con 5 componentes (pca_sincorrelacion_n5)

Todas se usarán para comparar qué configuración de datos es la que mejor ajusta los modelos.

Implementación

Perceptrón Multicapa

Los modelos de perceptrón simple son sencillos de entender, pero cuentan con la limitación de únicamente ser capaces de separar linealmente lo cual es poco útil para espacios de alta dimensionalidad como el conjunto de datos del problema actual. Alternativamente, existen los modelos de perceptrón multicapa que permiten separaciones no lineales a un conjunto de datos.

Para el problema actual, inicialmente se propone un modelo de perceptrón multicapa, PMC, este modelo fue programado sin usar las soluciones listas que ofrecen librerías como SciKitLearn. Con este modelo se utilizan las 4 configuraciones de datasets: aquella que contiene las 12 características, la que contiene únicamente las 7 características más relevantes y los dos modelos de PCA propuestos, pca_sincorrelacion_n4 y pca_sincorrelacion_n5.

La elección del número de neuronas en la capa oculta es especialmente relevante durante la implementación de este tipo de modelos. Para seleccionar el número adecuado de neuronas a utilizar para cada configuración del conjunto de datos, se opta por realizar 100 simulaciones para cada caso y estimar el *accuracy* promedio para cada combinatoria.

1. Después de evaluar modelos incrementando el número de neuronas en la capa oculta, se llegaron a las siguientes conclusiones principales:
2. Para un número de neuronas menor o igual a dos, el tiempo de convergencia es significativamente mayor que modelos con 3 o más neuronas.

El incremento marginal en el *accuracy score* por agregar neuronas es insignificante a partir de la cuarta neurona.

A continuación, se muestran los resultados obtenidos:

Accuracy promedio, 100 simulaciones por caso

Neuronas	sin PCA	PCA sin corr, 4	PCA sin corr, 5	sin PCA, 7 cols
3	0.8548	0.8251	0.8422	0.8624
4	0.8549	0.8268	0.8415	0.8612
5	0.8549	0.8316	0.8431	0.8592
6	0.8550	0.8342	0.8445	0.8579
7	0.8551	0.8351	0.8448	0.8577
8	0.8550	0.8380	0.8447	0.8572
9	0.8548	0.8399	0.8442	0.8560
10	0.8550	0.8402	0.8452	0.8539
11	0.8551	0.8410	0.8439	0.8533
12	0.8549	0.8428	0.8434	0.8525
13	0.8552	0.8431	0.8426	0.8520
14	0.8553	0.8442	0.8420	0.8524

70% del dataset usado como train, separado en cada simulación

Tabla 5. Comparativa de accuracy PNC.

Los modelos de PMC que usan todas las características y las 7 principales, presentan un desempeño ligeramente mejor que aquellos con componentes principales.

Debido a que el incremento de neuronas no presenta un incremento en *accuracy*, la elección del número de neuronas a utilizar en el PMC es tres. La única excepción se presenta en el modelo de PCA *n_4*, para el cual el incremento de neuronas sí está asociado al incremento de *accuracy*; aunque esta configuración de datos tiene un desempeño inferior respecto al PCA *n_5* y el modelo que utiliza las doce características.

Adicionalmente al modelo de PMC anteriormente explicado, que fue desarrollado paso por paso, se utiliza el modelo de perceptrón incluido en la librería *SciKitLearn* de *Python*. Este modelo ha sido desarrollado anteriormente y provee una solución sencilla de utilizar para problemas de este tipo.

Se observa que el uso de SciKit Learn resulta en un incremento de alrededor de 1 punto porcentual en el accuracy para este problema en cada conjunto de datos.

Una de las principales ventajas ofrecidas por los modelos de SciKitLearn es que tienen una gran cantidad de métodos ya implementados que resultan útiles y relativamente fáciles de utilizar. Uno de ellos es *permutation_importance* que permite estimar el impacto de las características en un modelo de red neuronal.

Los modelos de perceptrón multicapa tienen una naturaleza de caja negra, que dificulta la interpretación del modelo. Esta es una diferencia respecto a otros modelos de clasificación como la regresión logística, que puede resultar relevante dependiendo del contexto en el que se implementen estos modelos.

Sin embargo, es posible estimar el impacto de cada característica en los modelos de redes neuronales mediante procedimientos como el que implementa el método permutation_importance, que hace estimaciones mediante la medición de cuánto decrece una métrica de interés, en este caso accuracy, cuando cada característica no está disponible.

Los resultados obtenidos para cada configuración de datos son los siguientes:

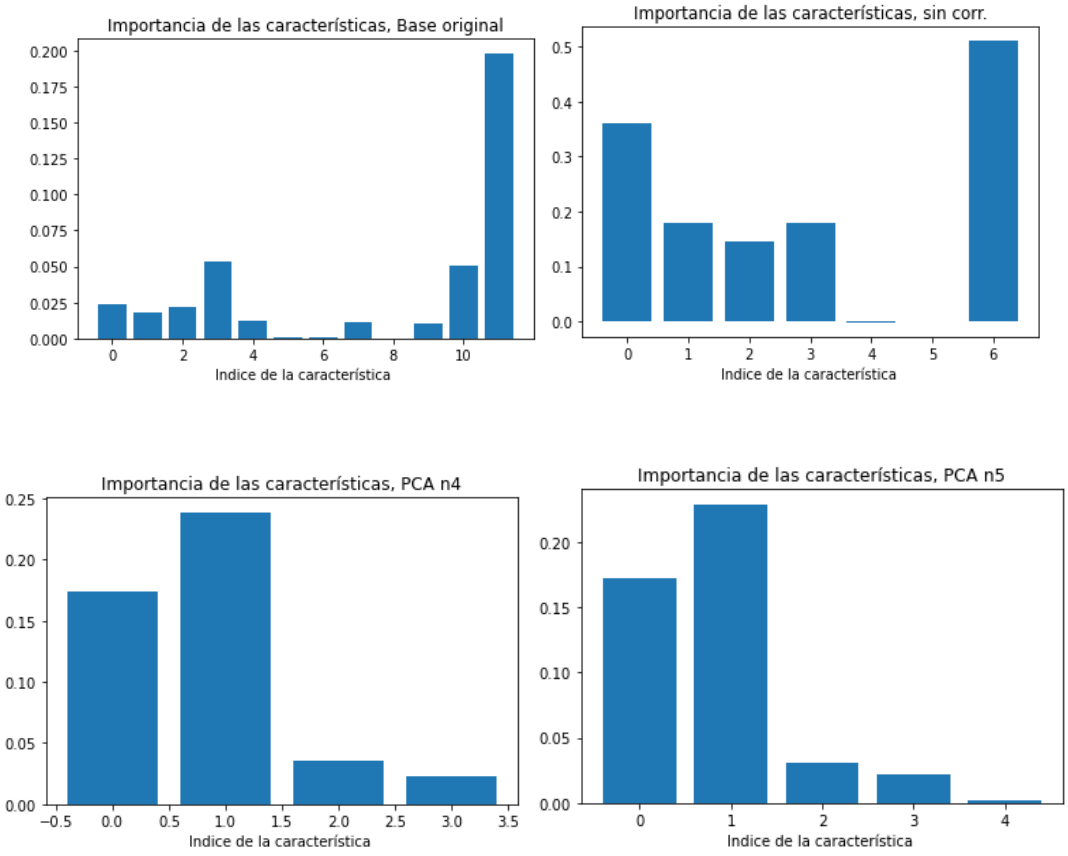


Ilustración 5. Importancia de las características para las diferentes configuraciones de datos.

Índices disponibles en Tabla 1. Descripción del conjunto de datos.

Para el modelo con el conjunto de datos original, *minor axis length*, *aspect ratio* y *compactness* son las tres características que tienen un impacto mayor en la estimación. Para el modelo de siete características, Aspect Ratio es la característica con mayor impacto seguido de *area* y *aspect ratio*. Destaca el bajo aporte que

tienen las características *solidity* y *extent*. Los modelos de PCA presentan un impacto mayor para los primeros dos componentes comparados con el resto, que era lo esperado del PCA.

Regresión Logística

Tener de referencia una regresión logística para comparar los resultados de modelos con mayor costo computacional es muy recomendable ya que no siempre el modelo más complicado es el mejor. En varias ocasiones una regresión logística es suficiente y tiene menos costo computacional.

En esta ocasión el *accuracy* de la regresión logística está en el rango de 84%-85% con las diferentes configuraciones de la base de datos (Tabla 6. Resultados de los Modelos.)

Máquina de soporte vectorial

Las máquinas de soporte vectorial, SVM, representan otro modelo clásico para la clasificación de datos, en particular los modelos de máquinas de soporte vectorial son especialmente efectivos en espacios de alta dimensionalidad.

Debido a ello, se prueba un modelo de SVM para cada configuración de datos, resultando en un *accuracy* alrededor de 86% a 87% (Vid Infra Tabla 6. Resultados de los Modelos.)

Modelo	Configuraciones	*Accuracy*
Logistic Regression	base_original	85.60
Logistic Regression	sin_correlacion	85.20
Logistic Regression	PCA con 4 componentes	84.20
Logistic Regression	PCA con 5 componentes	84.20
SVM	base_original	86.80
SVM	sin_correlacion	87.00
SVM	PCA con 4 componentes	87.00
SVM	PCA con 5 componentes	87.00
Perceptrón multicapa (3 neuronas)	base_original	85.48
Perceptrón multicapa (3 neuronas)	sin_correlacion	86.24
Perceptrón multicapa (14 neuronas)	PCA con 4 componentes	84.42
Perceptrón multicapa (3 neuronas)	PCA con 5 componentes	84.22
Perceptrón multicapa, SK Learn	base_original	86.13
Perceptrón multicapa SK Learn	sin_correlacion	86.80
Perceptrón multicapa SK Learn	PCA con 4 componentes	86.00
Perceptrón multicapa SK Learn	PCA con 5 componentes	85.73

Tabla 6. Resultados de los Modelos.

Conclusiones

Después de evaluar los diferentes modelos de PCM hemos comprobado que los modelos que tienen una o dos neuronas en la capa oculta presentan tiempos de cómputo significativamente altos (mayor a 15 min.) sin asegurar convergencia, mientras que el tiempo decrece significativamente a partir de 3 neuronas utilizadas. A partir de ese punto, las nuevas neuronas no significan un incremento ni siquiera de .01 en el *accuracy* promedio, por lo cual es fútil agregar 4 o más neuronas.

Además, en el modelo de PMC, utilizar componentes principales en lugar de la base original o el modelo de siete características no representó una reducción en el tiempo de cómputo ni un incremento en el *accuracy* promedio, tampoco permitió que modelos con 1 o 2 neuronas convergieran más rápidamente. El modelo con las siete características presenta un *accuracy* ligeramente mayor al que usa todas las columnas, por lo cual el mejor modelo de perceptrón para este problema es el que utiliza las 7 características con tres neuronas en la capa oculta. Siendo *aspect ratio, area* y *mayor axis* las variables de entrada que más peso tuvieron con la salida.

En la regresión logística la base original dio mejores resultados que las tratadas, pero no dio mejor resultado que el PMC o el SVM.

En el SVM los datos sin correlación o el PCA dieron el mismo resultado, dando un *accuracy* de 87% siendo el más alto de los modelos. Se quiso probar la diferencia entre 5 y 4 componentes del PCA, en este caso 4 componentes era suficiente y con 5 componentes no hubo diferencia significativa en ninguno de los modelos.

Referencias bibliográficas

Brownlee, J. (2020, August 20). *How to calculate feature importance with python*. MachineLearningMastery.com. Recuperado en noviembre 28, 2022 de https://machinelearningmastery.com/calculate-feature-importance-with-python/

Fraj, M. B. (2018, enero 5). *In depth: Parameter tuning for SVC*. Medium. Recuperado en noviembre 28, 2022, de https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769

Galarnyk, M. (2022, abril 27). *Logistic regression using python (scikit-learn)*. Medium. Recuperado en noviembre 28, 2022 de https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a

Géron, A. (2022). *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools, and techniques to build Intelligent Systems*. O'Reilly.

Koklu, M. (2022, marzo 28). *Pumpkin seeds dataset*. Kaggle. Recuperado en noviembre 28, 2022 de https://www.kaggle.com/datasets/muratkokludataset/pumpkin-seeds-dataset

Koklu, M., Sarigil, S., & Ozbek, O. (2021, June 25). *The use of machine learning methods in classification of pumpkin seeds (Cucurbita pepo L.) - genetic resources and crop evolution*. SpringerLink. Recuperado en noviembre 28, 2022 de https://link.springer.com/article/10.1007/s10722-021-01226-0/tables/1

Longoni, M. G., Porcel, E., López, M. V., & Dapozo, G. N. (2010). Modelos de Redes Neuronales Perceptrón Multicapa y de Base Radial para la predicción del rendimiento académico de alumnos universitarios. In XVI Congreso Argentino de Ciencias de la Computación. Recuperado en noviembre 28, 2022 de http://sedici.unlp.edu.ar/bitstream/handle/10915/19333/Documento_completo.pdf?sequence=1&isAllowed=y

Mammone, A., Turchi, M., & Cristianini, N. (2009). Support Vector Machines. WIREs Computational Statistics, 1(3), 283–289. https://doi.org/10.1002/wics.49

Pelaez, I. M. (2016). Modelos de Regresión lineal y logística. *Métodos Bioestadísticos*, 207–246. https://doi.org/10.2307/j.ctvvngkg.10

Pramoditha, R. (2022, junio 13). *Using PCA to reduce number of parameters in a neural network by 30X Times*. Medium. Recuperado en noviembre 28, 2022 de https://towardsdatascience.com/using-pca-to-reduce-number-of-parameters-in-a-neural-network-by-30x-times-fcc737159282

Wirth, M. A. (2004). *Shape Analysis & Measurement - Purdue University*. Purdue University. Recuperado en noviembre 28, 2022 de http://www.cyto.purdue.edu/cdroms/micro2/content/education/wirth10.pdf

Anexo 1. Código preprocesamiento, regresión logística y máquina de soporte vectorial.

Código y datos disponible en https://github.com/AlexNoelHdz/Pumpkin_analysis

```
import sympy as sp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sympy import Symbol, solveset, S, erf, log, sqrt, diff,Float
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn import svm
from sklearn.metrics import accuracy_score
# linear regression feature importance
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
from functools import wraps
from sklearn.linear_model import LogisticRegression

datos=pd.read_excel('Data/Pumpkin_Seeds_Dataset.xlsx')

#correlacion=datos.corr()
#checando nulos
datos.isnull().sum()
#checando cuantos hay de cada tipo
datos['Class'].value_counts()
datos['Class'] = [ 1 if each == 'Çerçevelik' else 0 for each in datos['Class']]
#splitting data

x = datos.drop('Class', axis = 1)
y = datos.Class.values


#%% correlation
plt.figure(figsize  = (10,10))
data = x.corr()
sns.heatmap(data, annot = True)
```

```
#quitamos columnas correlacionadas (compactness,aspect_ratio,eccentricity,
roundness)
#(area,convex area,equis_diameter)

X = x.drop(['Compactness','Roundness','Eccentricity'
,'Convex_Area','Equiv_Diameter'],axis=1)


#%% #Normales y sin outliers
scaler=StandardScaler()
scaler.fit(X)
scaled_data=scaler.transform(X)



scalerori=StandardScaler()
scalerori.fit(x)
scaled_dataori=scalerori.transform(x)



#%% Algoritmo pca
pca=PCA()
#pca=PCA(n_components=10) # Indicar el número de componentes principales
pca.fit(scaled_data)

# Ponderación de los componentes principales (vectores propios)
pca_score=pd.DataFrame(data    = pca.components_, columns = X.columns,)


# Pesos
loading_scores = pd.DataFrame(pca.components_[1])
#Nombre de las columnas

loading_scores.index=X.columns
# Ordena de mayor a menor los pesos
sorted_loading_scores = loading_scores[0].abs().sort_values(ascending=False)

per_var=np.round(pca.explained_variance_ratio_*100, decimals=1)
# Porcentaje de varianza acumulado de los componentes
porcent_acum = np.cumsum(per_var)

#con 4 compenetes es decente
pca=PCA(n_components=4) # Indicar el número de componentes principales
```

```
df=pd.DataFrame( pca.fit_transform(scaled_data))
pca_sincorrelacion_n4=df.copy()

#a partir del quinto componente ya no cambias
pca5=PCA(n_components=5) # Indicar el número de componentes principales
df5=pd.DataFrame( pca5.fit_transform(scaled_data))
pca_sincorrelacion_n5=df5.copy()



df.to_excel('pca_sincorrelacion_n4.xlsx')
#%% SVC con PCA n=4
#empezando modelo dividir train test

x_train_pca, x_test_pca, y_train_pca, y_test_pca = train_test_split(df, y,
test_size = 0.2, random_state = 42)

##modelo final
svc = svm.SVC().fit(x_train_pca, y_train_pca)
y_pred_pca = svc.predict(x_test_pca)
print("Accuracy = ", accuracy_score(y_test_pca, y_pred_pca)*100)
#Accuracy =  87.0


#%% SVC con PCA n=5
#empezando modelo dividir train test

x_train_pca, x_test_pca, y_train_pca, y_test_pca =
train_test_split(pca_sincorrelacion_n5, y, test_size = 0.2, random_state = 42)

##modelo final
svc = svm.SVC().fit(x_train_pca, y_train_pca)
y_pred_pca = svc.predict(x_test_pca)
print("Accuracy = ", accuracy_score(y_test_pca, y_pred_pca)*100)
#Accuracy =  87.0




#%% SVC sin PCA sin correlacion

x_train, x_test, y_train, y_test = train_test_split(scaled_data, y, test_size =
0.2, random_state = 42)
```

```
##modelo final
svc2 = svm.SVC().fit(x_train, y_train)
y_pred = svc2.predict(x_test)
print("Accuracy = ", accuracy_score(y_test, y_pred)*100)
#Accuracy =  87.0


#%% SVC con datos originales

x_train, x_test, y_train, y_test = train_test_split(scaled_dataori, y,
test_size = 0.2, random_state = 42)


##modelo final
svc2 = svm.SVC().fit(x_train, y_train)
y_pred = svc2.predict(x_test)
print("Accuracy = ", accuracy_score(y_test, y_pred)*100)
#Accuracy =  86.8


#%% LR con PCA n=4
x_train_pca, x_test_pca, y_train_pca, y_test_pca = train_test_split(df, y,
test_size = 0.2, random_state = 42)
# define the model
logisticRegr = LogisticRegression()
logisticRegr.fit(x_train_pca, y_train_pca)
y_pred_lr = logisticRegr.predict(x_test_pca)
print("Accuracy = ", accuracy_score(y_test_pca, y_pred_lr)*100)
#Accuracy =  84.2


#%% LR con PCA n=5
x_train_pca, x_test_pca, y_train_pca, y_test_pca =
train_test_split(pca_sincorrelacion_n5, y, test_size = 0.2, random_state = 42)
# define the model
logisticRegr = LogisticRegression()
logisticRegr.fit(x_train_pca, y_train_pca)
y_pred_lr = logisticRegr.predict(x_test_pca)
print("Accuracy = ", accuracy_score(y_test_pca, y_pred_lr)*100)
#Accuracy =  84.2



#%% LR sin PCA
# define the model
```

```
x_train, x_test, y_train, y_test = train_test_split(scaled_data, y, test_size =
0.2, random_state = 42)
logisticRegr = LogisticRegression()
logisticRegr.fit(x_train, y_train)
y_pred_lr2 = logisticRegr.predict(x_test)
print("Accuracy = ", accuracy_score(y_test, y_pred_lr2)*100)
#Accuracy =  85.2

#%% LR datos original
# define the model

x_train, x_test, y_train, y_test = train_test_split(scaled_dataori, y,
test_size = 0.2, random_state = 42)

logisticRegr = LogisticRegression()
logisticRegr.fit(x_train, y_train)
y_pred_lr2 = logisticRegr.predict(x_test)
print("Accuracy = ", accuracy_score(y_test, y_pred_lr2)*100)
#Accuracy =  85.6
```

Anexo 2. Perceptrón multicapa

Código y datos disponible en https://github.com/AlexNoelHdz/Pumpkin_analysis

```
import pandas as pd
import numpy as np
import time
from functools import wraps

def print_duration(func):
    """Decorador de Python para imprimir la duracion de un metodo
    """
    @wraps(func)
    def timeit_wrapper(*args, **kwargs):
        start_time = time.perf_counter()
        result = func(*args, **kwargs)
        end_time = time.perf_counter()
        total_time = end_time - start_time
        print(f'Función toma {total_time:.10f} seconds')
        return result
    return timeit_wrapper

def forward(x, w_h, w_o):
    a = 1
    net_h = w_h@x.T
    yh = 1/(1 + np.e**(-a*net_h)) #sigmoid function
    net_o = w_o @ yh
    y = 1/(1 + np.e**(-a*net_o)) #sigmoid function
    return yh, y

def backward(x, w_h, w_o, d, alpha):
    yh, y = forward(x, w_h, w_o)
    delta_o = (d.T-y) * y * (1-y)
    delta_h = yh * (1-yh) * (w_o.T@delta_o)
    dwo = alpha*delta_o @ yh.T
    dwh = alpha*delta_h @ x
    return dwo, dwh, np.linalg.norm(delta_o)

@print_duration
def training(N, M, x, d):
    alpha = 0.8
    #print(L)
    w_h = np.random.uniform(-1, 1, (L,N))
    w_o = np.random.uniform(-1, 1, (M,L))
    while (True) :
```

```
        for j in range(len(x)) :
            dwo, dwh, norm = backward(x[j].reshape(1,N), w_h, w_o,
d[j].reshape(1,M), alpha)
            w_o = w_o + dwo #adjusting output weights
            w_h = w_h + dwh #adjusting hidden weights
        if norm < 10 **-1 :
            #if condition to stop adjusting is sattisfied
            break
    return w_h, w_o

def read_data(features_used, response_index):
    N = len(features_used)
    #data = np.array(pd.read_excel('Data/pca_sincorrelacion_n4.xlsx', usecols =
features_used + response_index ).replace({'Class' : { 'Çerçevelik' : 0, 'Ürgüp
Sivrisi' : 1}}))
    #data = np.array(pd.read_excel('Data/pca_sincorrelacion_n5.xlsx', usecols =
features_used + response_index ).replace({'Class' : { 'Çerçevelik' : 0, 'Ürgüp
Sivrisi' : 1}}))

    #data = np.array(pd.read_excel('Data/Pumpkin_Seeds_Dataset.xlsx', usecols =
features_used + response_index  ).replace({'Class' : { 'Çerçevelik' : 0, 'Ürgüp
Sivrisi' : 1}}))
    data = np.array(pd.read_excel('Data/Pumpkin_Seeds_Dataset_sin_c.xlsx',
usecols = features_used + response_index  ).replace({'Class' : { 'Çerçevelik' :
0, 'Ürgüp Sivrisi' : 1}}))

    for i in range(N):
        data[:,i] =  (data[:,i] - data[:,i].mean()) / np.std(data[:,i])
#scaling 0-1

    np.random.shuffle(data)
    return data

def estimate(test, wh, wo):
    #Last forward, with estimated weights
    est = np.zeros(test[:,N:].shape)
    for j in range(len(test[:,:N])):
        _, y = forward(test[:,:N][j].reshape(1,N), wh, wo)
        est[j] = y.reshape(len(y),)
    return np.round(est) #replacing in test array, rounds [0-1] decimal values
to 0 or 1
```

```
features_used = list(range(7)) #indices de las características a incluir. En
este caso todas
response_index = [7]


N = len(features_used) #M number of inputs
M = 1 #M number of outputs
perc_train = 0.7 #porcentaje a usar como train

data = read_data(features_used, response_index)
nrows, ncols = data.shape

simulaciones = []  #lista que contendra las 100 simulaciones (para guardar y
promediar)
promedios_simulaciones = []  # Lista que contendra el promedio para cada n
diferente usada (3 neuronas, 4, 5, etc)

for n in range ( 3, 16 ): #numero de neuronas usadas
    L = n   #L number  of neurons
    for i in range( 100 ) : #ajusta para n neuronas
        train, test = data[:int(nrows*perc_train),:],
data[int(nrows*perc_train):,:] #separacion data en test/train
        x_train, d_train = train[:,:N], train[:,N:]   #separa la columna de
respuesta de las features
        wh, wo = training(N, M, x_train, d_train) #estima los pesos de la red
        estimations = estimate(test, wh, wo)   # predice
        evaluation = np.reshape(test[:,N] , [int(nrows*(1-perc_train)),1] ) ==
estimations # compara (estimo correcto o no?)
        simulaciones.append( np.sum(evaluation) / int(nrows*(1-perc_train))
)  #calcula accuracy
    promedios_simulaciones.append( np.mean(simulaciones) )  #  promedios de
accuracy de las 100 sims para cada numero de neuronas se guarda en arreglo
    simulaciones = [] # reinicio simulaciones para que el promedio de la
proxima neurona sea independiente

print(promedios_simulaciones)
```

Anexo 3. Perceptrón Sklearn.

Código y datos disponible en https://github.com/AlexNoelHdz/Pumpkin_analysis

```
#Modelo Perceptron Multicapa con SKLearn

import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.inspection import permutation_importance
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

N = 7 # num caracteristicas
features_used = list(range(N)) #indices de las características a incluir. En
este caso todas
response_index = [len(features_used)]  # indice de la variable respuesta

#data = np.array(pd.read_excel('Data/Pumpkin_Seeds_Dataset.xlsx', usecols =
features_used + response_index  ).replace({'Class' : { 'Çerçevelik' : 0, 'Ürgüp
Sivrisi' : 1}}))
#data = np.array(pd.read_excel('Data/pca_sincorrelacion_n4.xlsx', usecols =
features_used + response_index ).replace({'Class' : { 'Çerçevelik' : 0, 'Ürgüp
Sivrisi' : 1}}))
#data = np.array(pd.read_excel('Data/pca_sincorrelacion_n5.xlsx', usecols =
features_used + response_index ).replace({'Class' : { 'Çerçevelik' : 0, 'Ürgüp
Sivrisi' : 1}}))
data = np.array(pd.read_excel('Data/Pumpkin_Seeds_Dataset_sin_c.xlsx', usecols
= features_used + response_index  ).replace({'Class' : { 'Çerçevelik' : 0,
'Ürgüp Sivrisi' : 1}}))




for i in range(N):
     data[:,i] =  (data[:,i] - data[:,i].min()) / (data[:,i].max() -
data[:,i].min()) #scaling 0-1

X = data[:, :N]
Y = data[:, N:]

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,
random_state = 42)
```

```
# modelo final
clf = MLPClassifier(solver='lbfgs', alpha = 10**-5, hidden_layer_sizes=(2, 2),
random_state=1)
clf = clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
print("Accuracy = ", accuracy_score(y_test, y_pred)*100)


result = permutation_importance(clf,X,Y,scoring='accuracy')
importance = result.importances_mean

for i,v in enumerate(importance):
     print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.title("Importancia de las características, sin corr.")
pyplot.xlabel("Indice de la característica " )
pyplot.show()

MLPClassifier(alpha=1e-05, hidden_layer_sizes=(2, 2), random_state=1,
solver='lbfgs')
MLPClassifier
```