

# PEC 4: Predicción de subtipos de cáncer de piel

Raúl Pérez Prats

28/06/2024

## Tabla de Contenidos

### 1. Exploración de los datos

- a) Visualización del dataset
- b) Distribución de las clases
- c) Correlaciones entre las variables
- d) Distribución de las variables
- e) PCA
- f) Partición de los datos

### 2. Entrenamiento Algoritmos

- a) Definimos funciones para entrenamiento y evaluación con CrossValidation y Matriz de Confusión
- b) k-Nearest Neighbour
- c) Naive Bayes
- d) Artificial Neural Network
- e) Support Vector Machine (SVM)
- f) Árbol de Clasificación
- g) Random Forest

### 3. Discusión y conclusión sobre el rendimiento de los algoritmos

# 1. Exploración de los datos

## a) Visualización del dataset

In [209...]

```
import pandas as pd
import numpy as np
```

In [210...]

```
# Cargamos dataset y lo printamos
df = pd.read_csv('rna1.csv')
df
```

Out[210...]

	Class	RTN2	NDRG2	CCDC113	FAM63A	ACADS	GMDS	HLA.H	SEMA4A
<b>0</b>	tipus1	4.362183	7.533461	3.956124	4.457170	2.256817	6.017940	5.006907	3.217812
<b>1</b>	tipus1	1.984492	7.455194	5.427623	5.440957	4.028813	4.341692	6.178668	2.864659
<b>2</b>	tipus1	1.727323	8.079968	2.227300	5.543480	2.629855	6.363030	6.039563	5.946028
<b>3</b>	tipus1	4.363996	5.793750	3.544866	4.737114	4.269101	4.001104	7.087633	5.007565
<b>4</b>	tipus1	2.447562	7.158993	4.691256	4.808728	2.442135	7.029723	5.936138	5.901459
...	...	...	...	...	...	...	...	...	...
<b>145</b>	tipus3	3.139900	6.222359	4.306467	6.912236	5.317992	4.686750	5.687450	5.601147
<b>146</b>	tipus3	4.346977	5.708690	3.995944	6.189790	4.488860	3.186237	4.942399	4.299292
<b>147</b>	tipus3	3.287485	5.043387	3.665758	5.943128	4.823124	5.077987	7.738051	1.269034
<b>148</b>	tipus3	4.684112	6.274357	4.599318	5.335620	4.247261	3.627571	4.795321	5.166827
<b>149</b>	tipus3	4.294419	5.236494	4.310629	5.955053	4.631673	4.648448	5.596224	5.023054

150 rows × 201 columns

In [211...]

```
df.shape
```

Out[211...]

```
(150, 201)
```

Tenemos 150 muestras con 201 variables por muestra

In [238...]

```
df.isnull().sum()
```

```

Class      0
RTN2       0
NDRG2      0
CCDC113    0
FAM63A     0
..
KRT8       0
TP53INP2   0
JAM3       0
ZNF680     0
PBX1       0
Length: 201, dtype: int64

```

No hay valores nulos

	<b>RTN2</b>	<b>NDRG2</b>	<b>CCDC113</b>	<b>FAM63A</b>	<b>ACADS</b>	<b>GMDS</b>	<b>HLA.H</b>
<b>count</b>	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
<b>mean</b>	3.549331	6.010226	3.765484	5.325782	3.586201	4.554646	6.287433
<b>std</b>	1.166884	1.387006	1.029216	1.071280	1.026817	0.992257	1.126872
<b>min</b>	0.332753	3.310945	0.284974	1.840127	0.871557	1.983430	2.957603
<b>25%</b>	2.811015	4.997007	3.269642	4.561730	3.006913	3.950741	5.574561
<b>50%</b>	3.649901	5.888765	3.878613	5.437620	3.495100	4.540508	6.185800
<b>75%</b>	4.393586	6.821142	4.308948	6.066787	4.227888	5.166531	7.009373
<b>max</b>	6.142850	10.260165	7.685072	7.680580	6.577862	7.105627	9.130370

8 rows × 200 columns

Observamos un resumen del dataset que incluye la media, la desviación estándar, los valores máximos y mínimos y los percentiles para cada variable

## b) Distribución de las clases

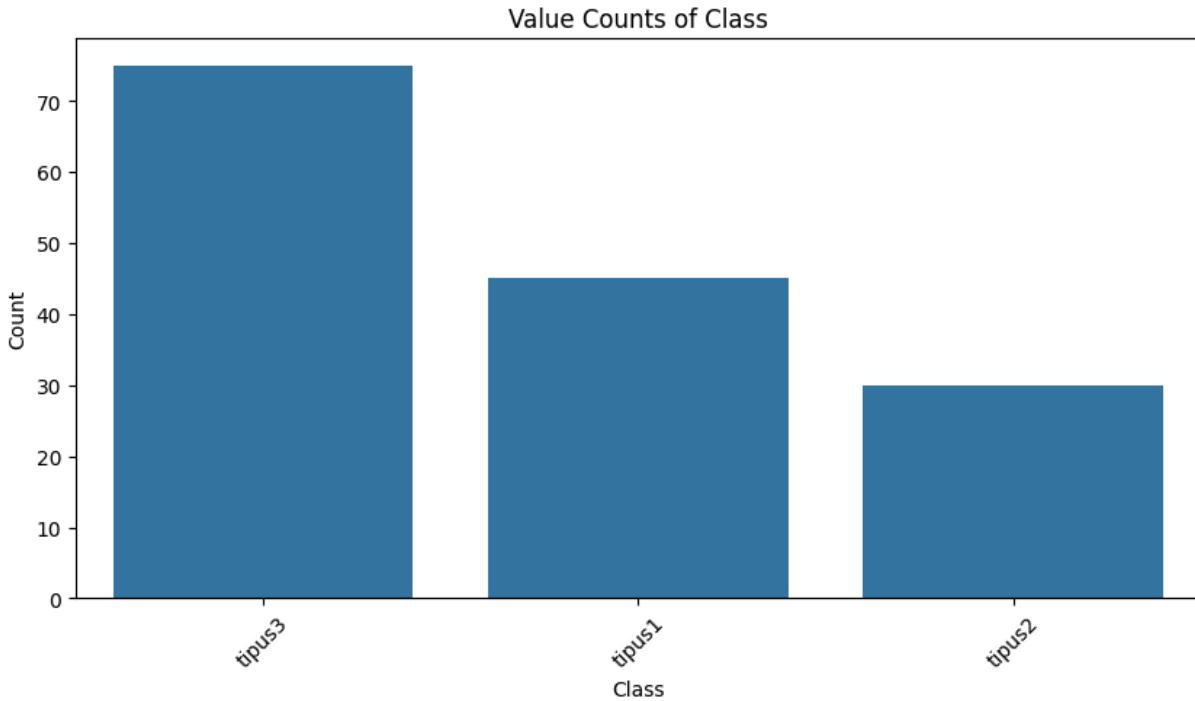
```

In [214...]: import pandas as pd
           import matplotlib.pyplot as plt
           import seaborn as sns

# Iteramos sobre las variables categoricas para crear barplot
for column in df.select_dtypes(include=['object']).columns:
    plt.figure(figsize=(10, 5))
    sns.countplot(x=df[column], order=df[column].value_counts().index)
    plt.title(f'Value Counts of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')

```

```
plt.xticks(rotation=45)
plt.show()
```

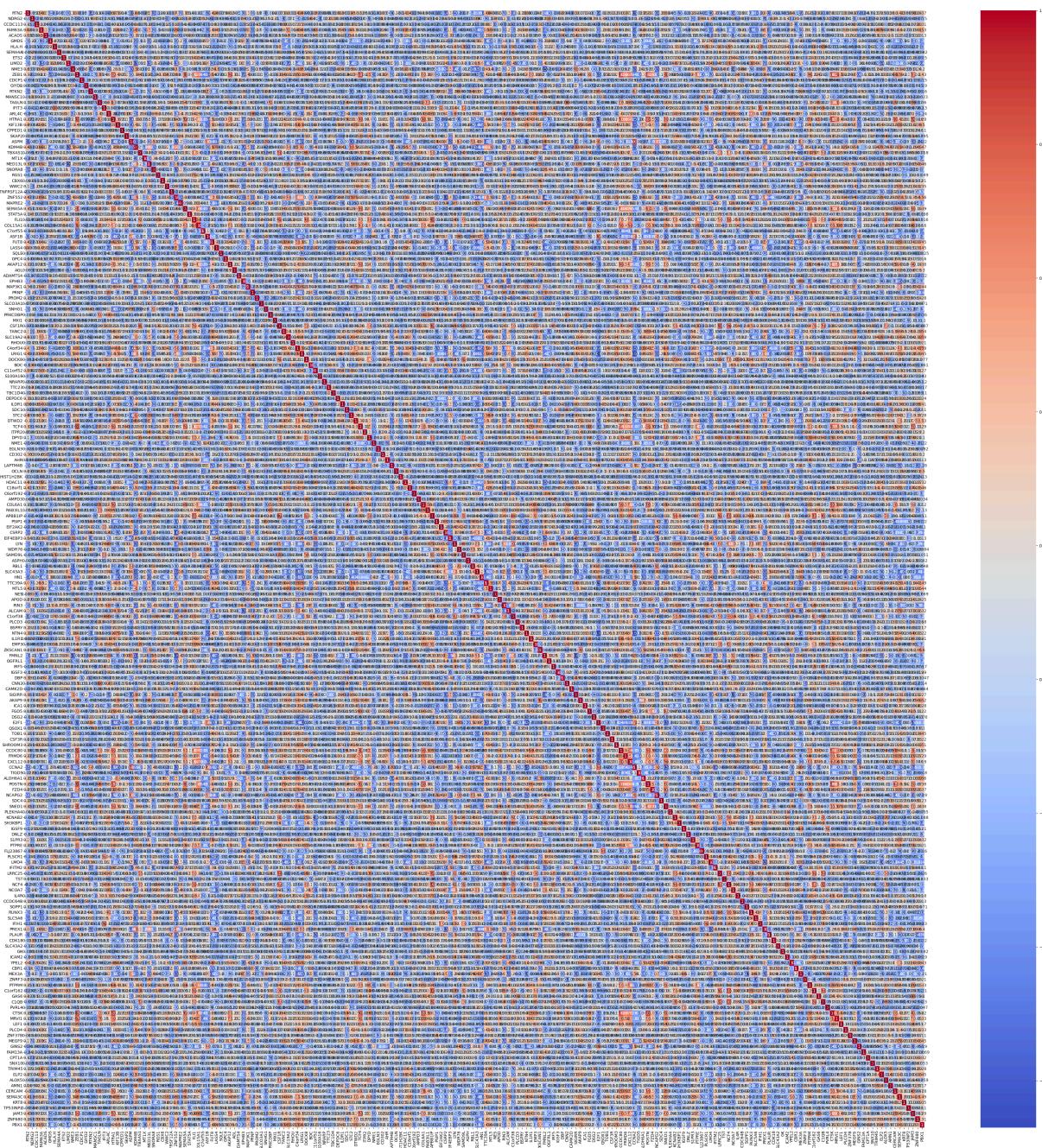


El gráfico de barras muestra el número de clases para cada tipo. Tenemos 3 clases no balanceadas, por lo tanto, se trata de un problema de 'multiclass classification' con datos no balanceados.

### c) Correlaciones entre las variables

```
In [215...]: import seaborn as sns
import matplotlib.pyplot as plt
# Añadimos a la X todas las variables predictoras
X = df.drop(['Class'], axis=1)
# Añadimos a la y la variable objetivo
y = df['Class']

plt.figure(figsize=(50, 50))
sns.heatmap(X.corr(), annot=True, cmap='coolwarm')
plt.show()
```



El heatmap muestra las correlaciones positivas y negativas entre las variables. Si hacemos zoom en la imagen podemos observar que existe una correlación positiva entre muchas variables, como podría ser JAM3 con ZEB1. Esto indica que cuando el RNA de una de estas variables se expresa, también aumenta la expresión de el otro.

## d) Distribución de las variables

```
In [237...]: import matplotlib.pyplot as plt
import seaborn as sns
```

```
# incluimos columnas numericas
num_cols = df.select_dtypes(include=['float64', 'int64']).columns
```

```

# Establecemos number of rows
plots_per_row = 3

# Calcula en numero de rows que necesita
num_rows = (len(num_cols) + plots_per_row - 1) // plots_per_row

# creamos subplots
fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(15, num_rows * 5))

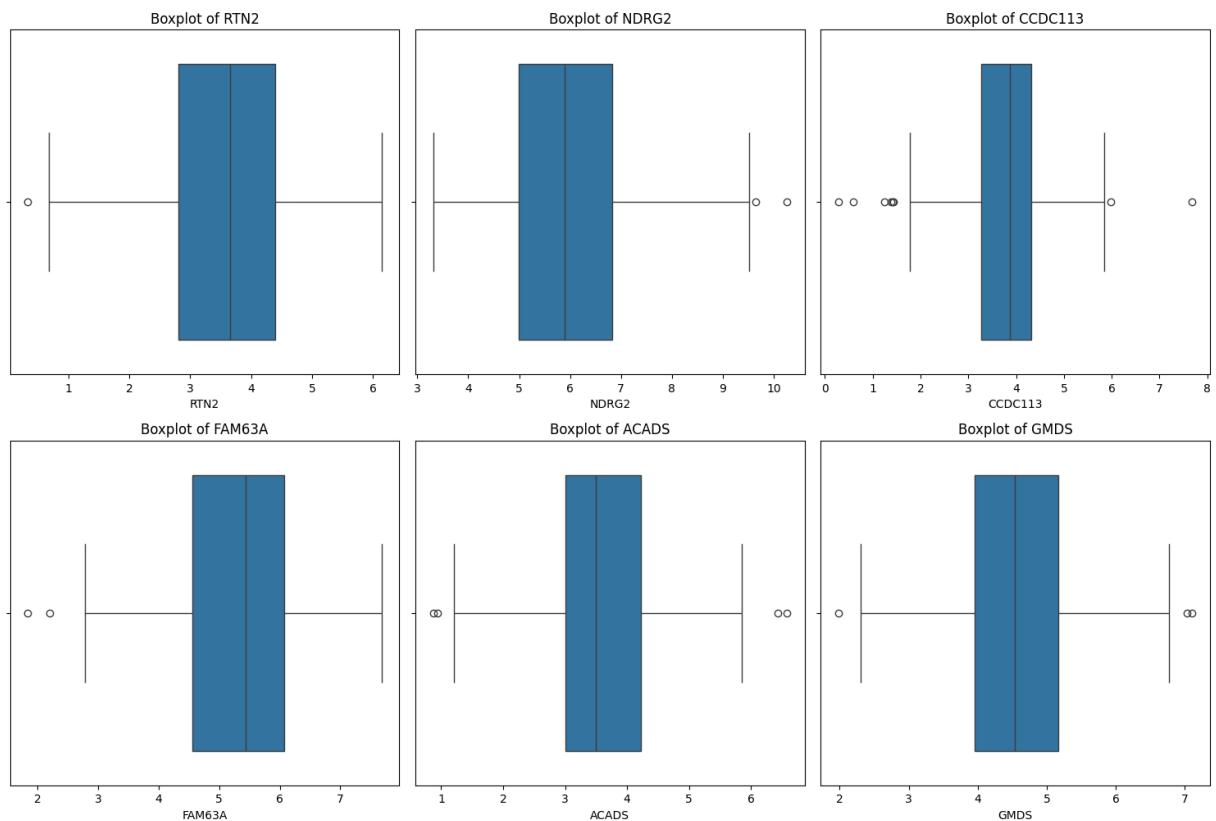
axes = axes.flatten()

# Iteramos sobre las columnas
for i, column in enumerate(num_cols[:6]): # solo seleccionamos 6 porque si no ocupa
    sns.boxplot(x=df[column], ax=axes[i])
    axes[i].set_title(f'Boxplot of {column}')

# Eliminamos subplots no usados
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()

```



Los gráficos de cajas y bigotes permiten observar la distribución de los valores de las muestras para cada variable predictora e identificar la presencia de posibles outliers, como es el caso de los gráficos NDRG2 y CCDC113

## e) PCA

In [217...]

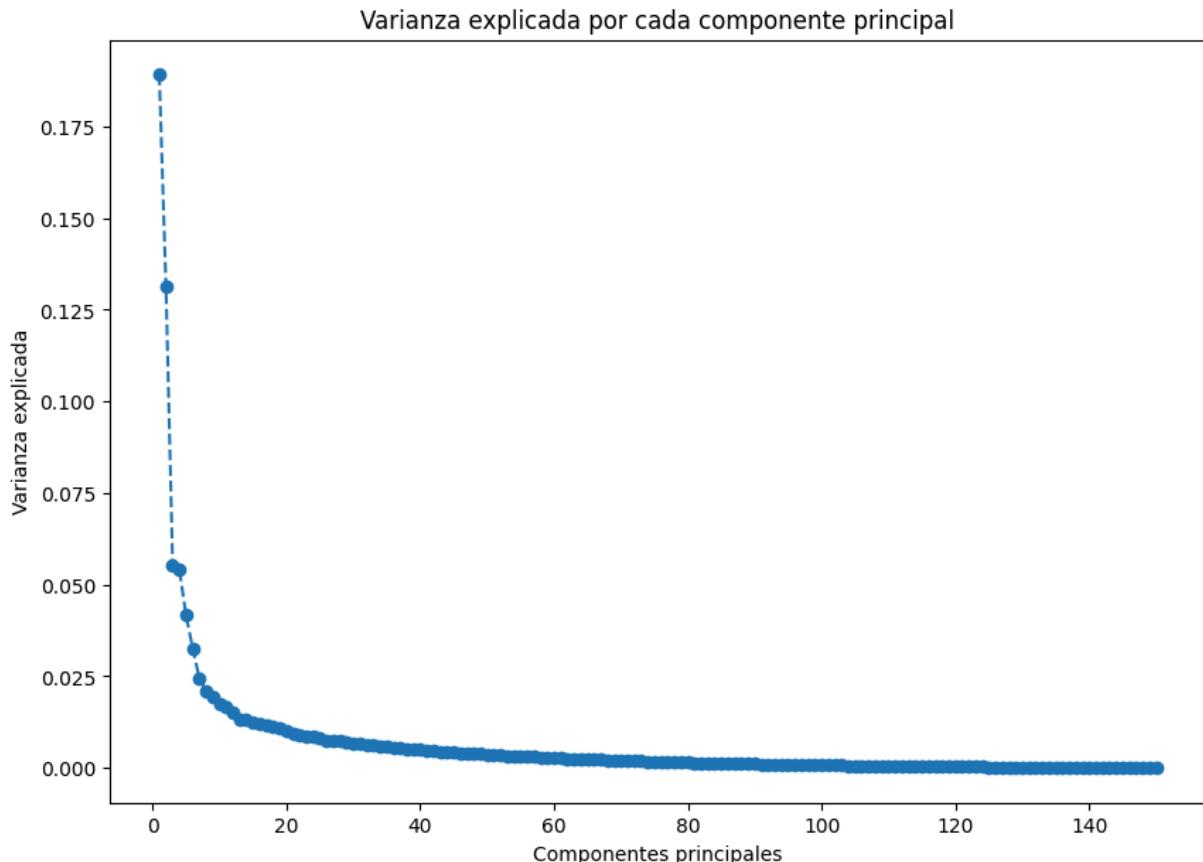
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Estandarizar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Crear un objeto PCA y ajustar los datos estandarizados
pca = PCA()
pca.fit(X_scaled)

# Transformar los datos
df_pca = pca.transform(X_scaled)

# Graficar la varianza explicada por cada componente principal
plt.figure(figsize=(10, 7))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_)
plt.title('Varianza explicada por cada componente principal')
plt.xlabel('Componentes principales')
plt.ylabel('Varianza explicada')
plt.show()
```

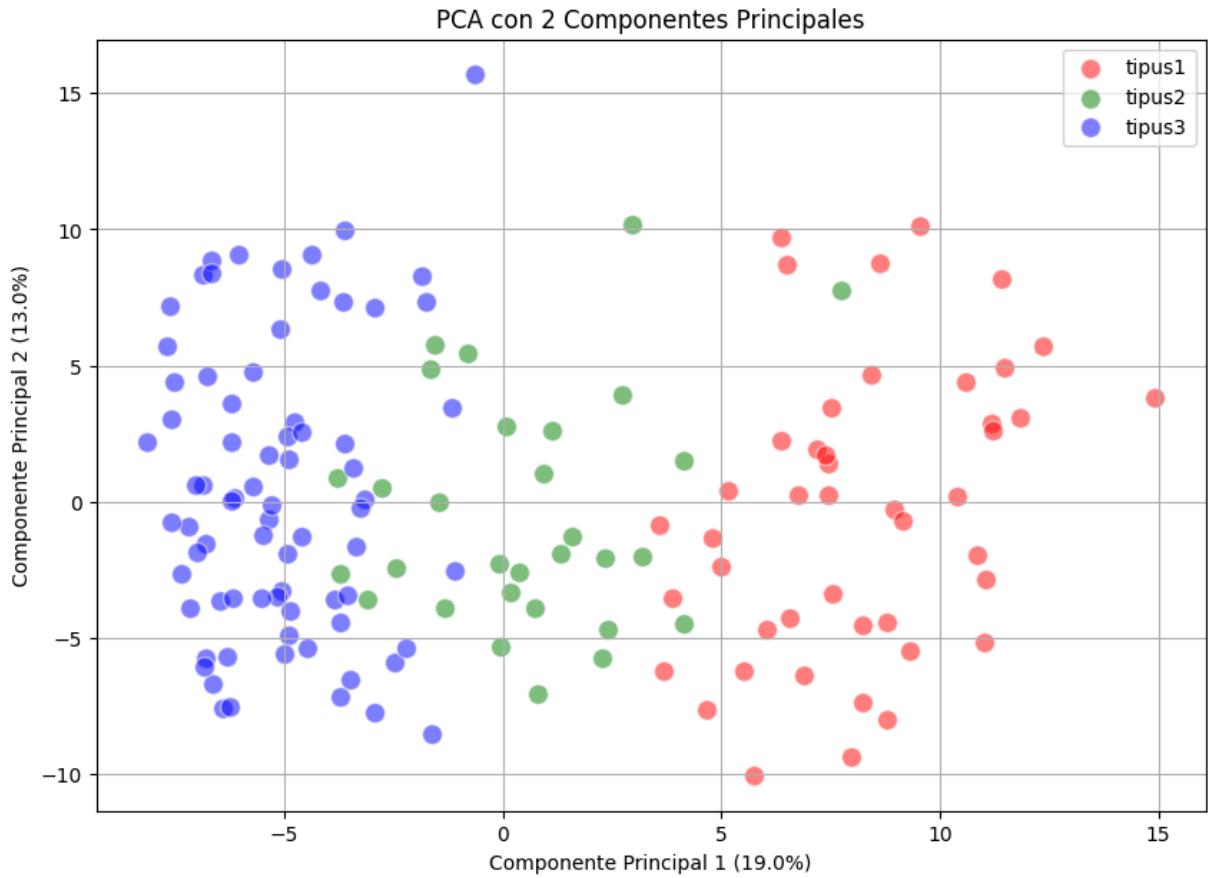


Se observa como el % de varianza explicada por componente va disminuyendo a medida que aumentamos el número de componentes. El óptimo de % por de varianza explicada por

componente se encuentra en aquellas componentes antes de la componente principal número 20

```
In [218...]  
# Varianza explicada por cada componente principal  
explained_variance = pca.explained_variance_ratio_  
  
# Cargas de cada variable en cada componente principal  
loadings = pca.components_  
  
# Crear un DataFrame para las cargas  
loadings_df = pd.DataFrame(loadings, columns=X.columns)
```

```
In [219...]  
from matplotlib.colors import ListedColormap  
  
# Crear un PCA que conserve 2 componentes principales  
pca_2 = PCA(n_components=2)  
df_pca_2 = pca_2.fit_transform(X_scaled)  
  
# Crear un DataFrame con los resultados del PCA  
df_pca_2 = pd.DataFrame(df_pca_2, columns=['PC1', 'PC2'])  
  
explained_variance_ratio_2 = pca_2.explained_variance_ratio_  
  
# Definir los colores para cada tipo  
colormap = ListedColormap(['red', 'green', 'blue'])  
  
df_pca_2['Tipo'] = y  
  
# Crear la gráfica  
plt.figure(figsize=(10, 7))  
  
# Graficar cada tipo con su respectivo color  
for i, tipo in enumerate(np.unique(y)):  
    indices = df_pca_2['Tipo'] == tipo  
    plt.scatter(df_pca_2.loc[indices, 'PC1'], df_pca_2.loc[indices, 'PC2'],  
                color=colormap(i), label=tipo, alpha=0.5, edgecolors='w', s=100)  
  
plt.xlabel(f'Componente Principal 1 ({round(explained_variance_ratio_2[0], 2)*100}%)')  
plt.ylabel(f'Componente Principal 2 ({round(explained_variance_ratio_2[1], 2)*100}%)')  
plt.title('PCA con 2 Componentes Principales')  
plt.legend()  
plt.grid(True)  
plt.show()
```



En el PCA se observa claramente una separación entre los componentes principales de los diferentes tipos de cáncer sobre la componente principal 1. Esto es una excelente noticia, ya que al graficar las primeras dos componentes principales, que explican solo el 32% de la varianza total, podemos distinguir visualmente los tipos de cáncer. Cuando entrenemos un algoritmo que considere más variables, también podrá realizar esta distinción. El hecho de que con solo el 19% de la varianza explicada (componente principal 1) podamos separar los tipos de cáncer en un gráfico 2D sugiere que hay variables menos relevantes que podemos excluir durante el entrenamiento, reduciendo así la dimensionalidad y el coste computacional sin perder precisión ('accuracy').

## f) Partición de los datos

In [220...]

```
from sklearn.model_selection import train_test_split

# División en conjunto de entrenamiento y prueba de forma estratificada para mitigar el sesgo
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify=y)
```

## 2. Entrenamiento Algoritmos

### a) Definimos funciones para entrenamiento y evaluación con CrossValidation y Matriz de Confusión

In [221...]

```
import warnings

# Suprimir todos los warnings para que el pdf report quede limpio
warnings.filterwarnings('ignore')
```

In [222...]

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import make_scorer, accuracy_score, recall_score, precision_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import confusion_matrix

def crossvalidation(modelo):

    # Define las métricas que quieres evaluar
    scoring = {
        'accuracy': make_scorer(accuracy_score),
        'recall': make_scorer(recall_score, average='weighted'),
        'precision': make_scorer(precision_score, average='weighted'),
        'f1': make_scorer(f1_score, average='weighted'),
    }

    # Realiza la validación cruzada de 3 folds
    cv_results = cross_validate(modelo, X, y, cv=3, scoring=scoring)

    # Calcula las medias de las métricas
    accuracy_mean = cv_results['test_accuracy'].mean()
    recall_mean = cv_results['test_recall'].mean()
    precision_mean = cv_results['test_precision'].mean()
    f1_mean = cv_results['test_f1'].mean()

    # Agrega los resultados a la lista
    results = {
        'parametros': k,
        'accuracy': accuracy_mean,
        'recall': recall_mean,
        'precision': precision_mean,
        'f1': f1_mean,
    }

    return results

def matriz_confusion(modelo):
    # Entrenamos
    modelo.fit(X_train, y_train)

    # Hacemos predicciones
    y_pred = modelo.predict(X_test)

    # Generamos la confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Define class labels (if applicable)
    class_names = np.unique(y)
```

```

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

```

## b) k-Nearest Neighbour

In [223...]

```

values_knn = [1, 3, 5, 7, 11]
results_knn = []
for k in values_knn:
    result = crossvalidation(modelo = KNeighborsClassifier(n_neighbors=k))
    results_knn.append(result)

print("Resultados de validación cruzada:")
df_knn = pd.DataFrame(results_knn)
df_knn

```

Resultados de validación cruzada:

Out[223...]

	parametros	accuracy	recall	precision	f1
<b>0</b>	1	0.900000	0.900000	0.902149	0.896391
<b>1</b>	3	0.913333	0.913333	0.920802	0.907258
<b>2</b>	5	0.913333	0.913333	0.916793	0.905750
<b>3</b>	7	0.900000	0.900000	0.907939	0.891610
<b>4</b>	11	0.906667	0.906667	0.910003	0.899598

El parámetro K = 3 es el óptimo, ya que muestra un rendimiento superior en todas las métricas, tan solo lo iguala k = 5 en accuracy y recall.

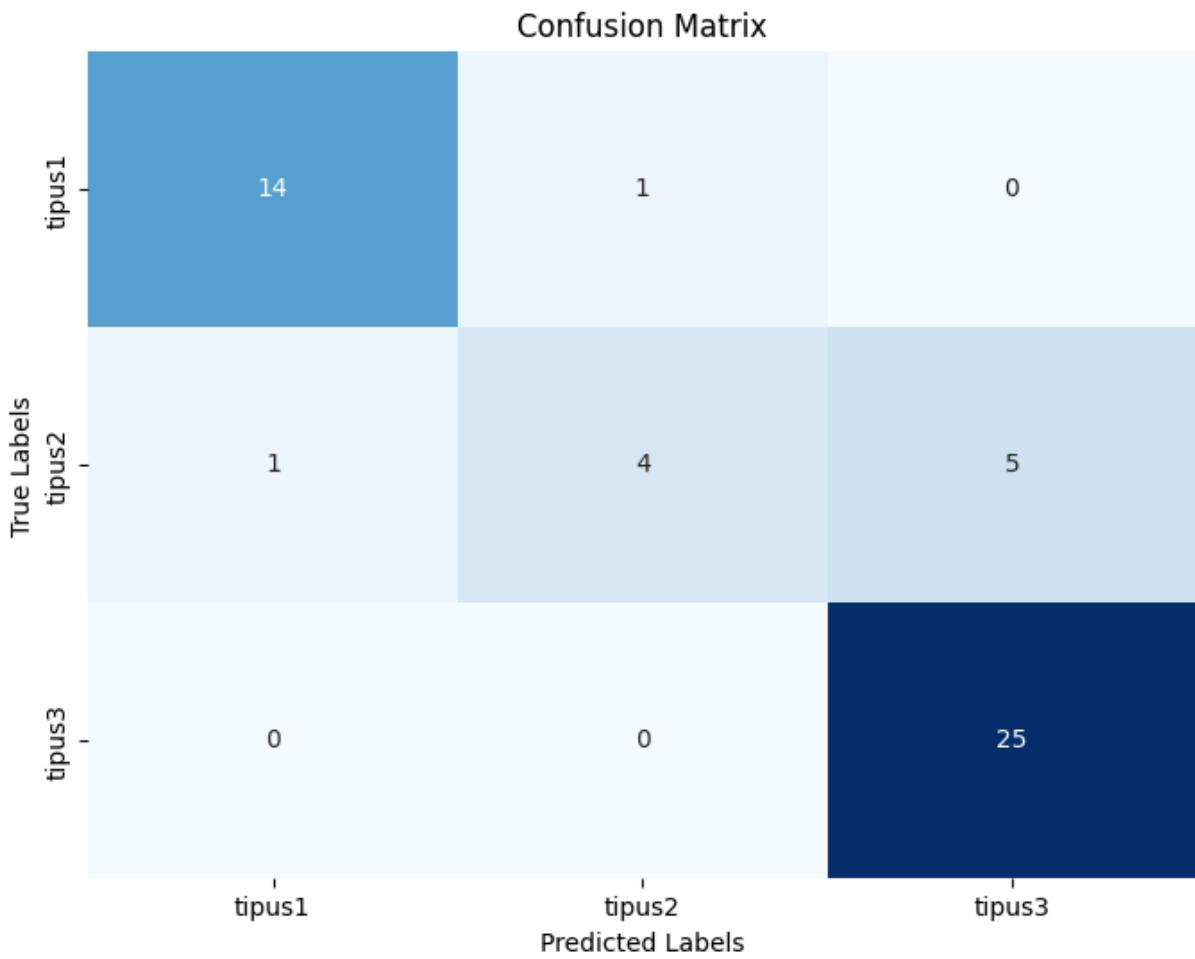
Hacemos una matriz de confusión para K = 3

In [224...]

```

cm_knn = matriz_confusion(modelo=KNeighborsClassifier(n_neighbors=3))
cm_knn

```



Vemos que clasifica correctamente todas las muestras de la clase tipus 1 y tipus 3. Tiene dificultades para clasificar correctamente 4 muestras de tipus 2 que las clasifica incorrectamente como tipus 3.

Si observamos el PCA anterior, se observa como algunos puntos pertenecientes a tipus 2 se mezclan con la nube de puntos tipus 3, de ahí que el KNN pueda haberse confundido en la clasificación, debido a que los centroides correspondientes al tipus 3 también actúa sobre estos puntos tipus 2 que se mezclan en los puntos tipus 3.

### c) Naive Bayes

En scikit-learn, el suavizado de Laplace se controla a través del parámetro alpha en los clasificadores Naive Bayes:

- alpha = 0.0 significa sin suavizado (equivalente a Laplace=False).
- alpha > 0.0 activa el suavizado de Laplace (equivalente a Laplace=True).

In [225...]

```
from sklearn.naive_bayes import MultinomialNB
values_nb = [0,1]
results_nb = []
for k in values_nb:
    result = crossvalidation(modelo = MultinomialNB(alpha=k))
```

```

    results_nb.append(result)

print("Resultados de validación cruzada:")
df_nb = pd.DataFrame(results_nb)
df_nb

```

Resultados de validación cruzada:

	parametros	accuracy	recall	precision	f1
<b>0</b>	0	0.94	0.94	0.945483	0.940969
<b>1</b>	1	0.94	0.94	0.945483	0.940969

No se observan diferencias entre las métricas de evaluación cuando entrenamos el algoritmo Naive Bayes con alpha = 0 o alpha = 1. Esto es debido a que alpha = 1 agrega un 1 a todas las frecuencias observadas de características en las distribuciones de probabilidad. Esto se hace para evitar problemas con probabilidades condicionales que podrían ser cero cuando una característica no está presente en una categoría específica de datos.

Comprobamos si hay valores = 0 que podrían afectar el entrenamiento:

```

In [226...]: zero_check = X.isin([0]).any()
zero_check

```

```

Out[226...]: RTN2      False
NDRG2      False
CCDC113    False
FAM63A     False
ACADS      False
...
KRT8       False
TP53INP2   False
JAM3       False
ZNF680     False
PBX1       False
Length: 200, dtype: bool

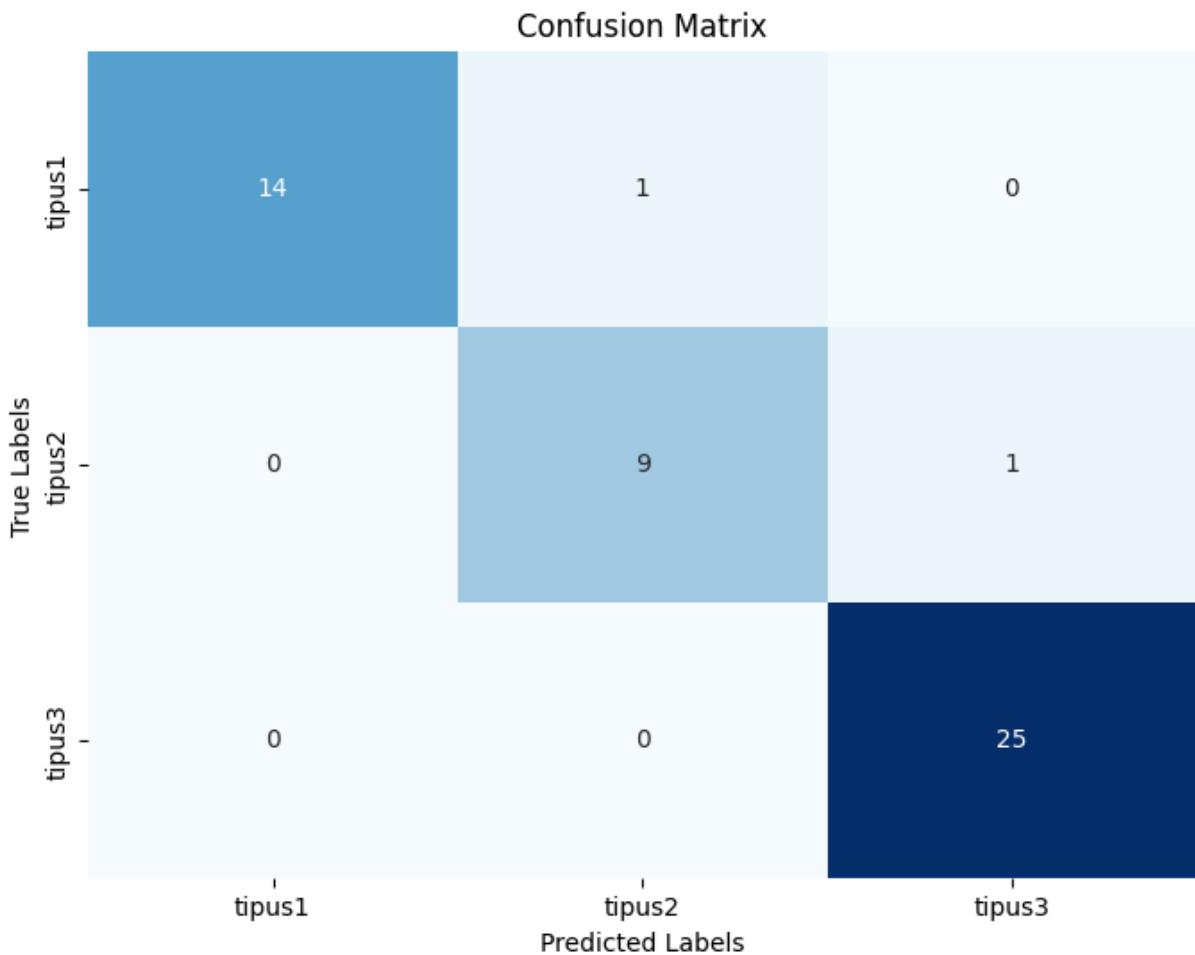
```

Como no hay valores = 0 en nuestro dataset, el valor de alpha (activar o no Laplace) no es relevante para nuestras predicciones.

```

In [227...]: cm_nb = matriz_confusion(modelo=MultinomialNB(alpha=1))
cm_nb

```



El algoritmo Naive Bayes hace una muy buena clasificación con un 94% de accuracy. Tan solo se equivoca en una muestra del TestSet que clasifica como tipus 3 cuando en realidad se trata de tipus 2

## d) Artificial Neural Network

Utilizaremos una arquitectura de multilayer perceptrón (MLP) como modelo de red neuronal artificial para la clasificación de subtipo de cáncer de piel

```
In [228...]: from sklearn.neural_network import MLPClassifier

capas_mlp = [(30,), (50, 10)]
results_mlp = []
for k in capas_mlp:
    result = crossvalidation(modelo = MLPClassifier(hidden_layer_sizes=k, activation='relu'), cv=5)
    results_mlp.append(result)

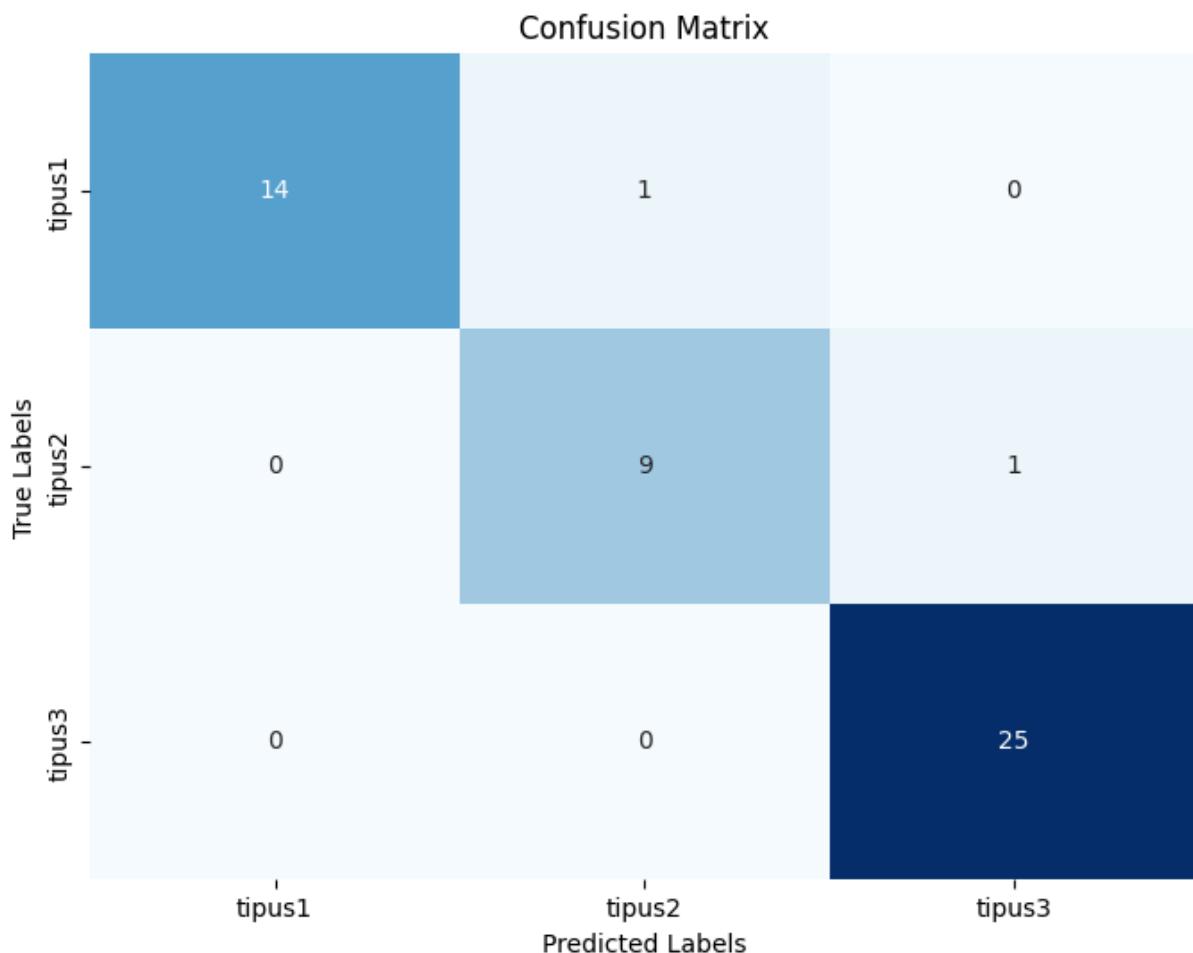
print("Resultados de validación cruzada:")
df_mlp = pd.DataFrame(results_mlp)
```

Resultados de validación cruzada:

	parametros	accuracy	recall	precision	f1
0	(30,)	0.940000	0.940000	0.942015	0.939584
1	(50, 10)	0.926667	0.926667	0.929964	0.926041

El parámetro óptimo es de 1 sola capa con 30 neuronas, con una accuracy, recall, precisión y F1-score del 94%

```
In [229...]: cm_mlp = matriz_confusion(modelo=MLPClassifier(hidden_layer_sizes=(30,), activation
```



El algoritmo MLP hace una clasificación casi perfecta, igual que el algoritmo Naive Bayes, tan solo se equivoca en una muestra del TestSet que clasifica como tipus 3 cuando en realidad se trata de tipus 2

## e) Support Vector Machine (SVM)

```
In [230...]: from sklearn.svm import SVC
kernel = ['linear', 'rbf']
results_svc = []
for k in kernel:
    result = crossvalidation(modelo = SVC(kernel=k))
    results_svc.append(result)
```

```

print("Resultados de validación cruzada:")
df_svc = pd.DataFrame(results_svc)
df_svc

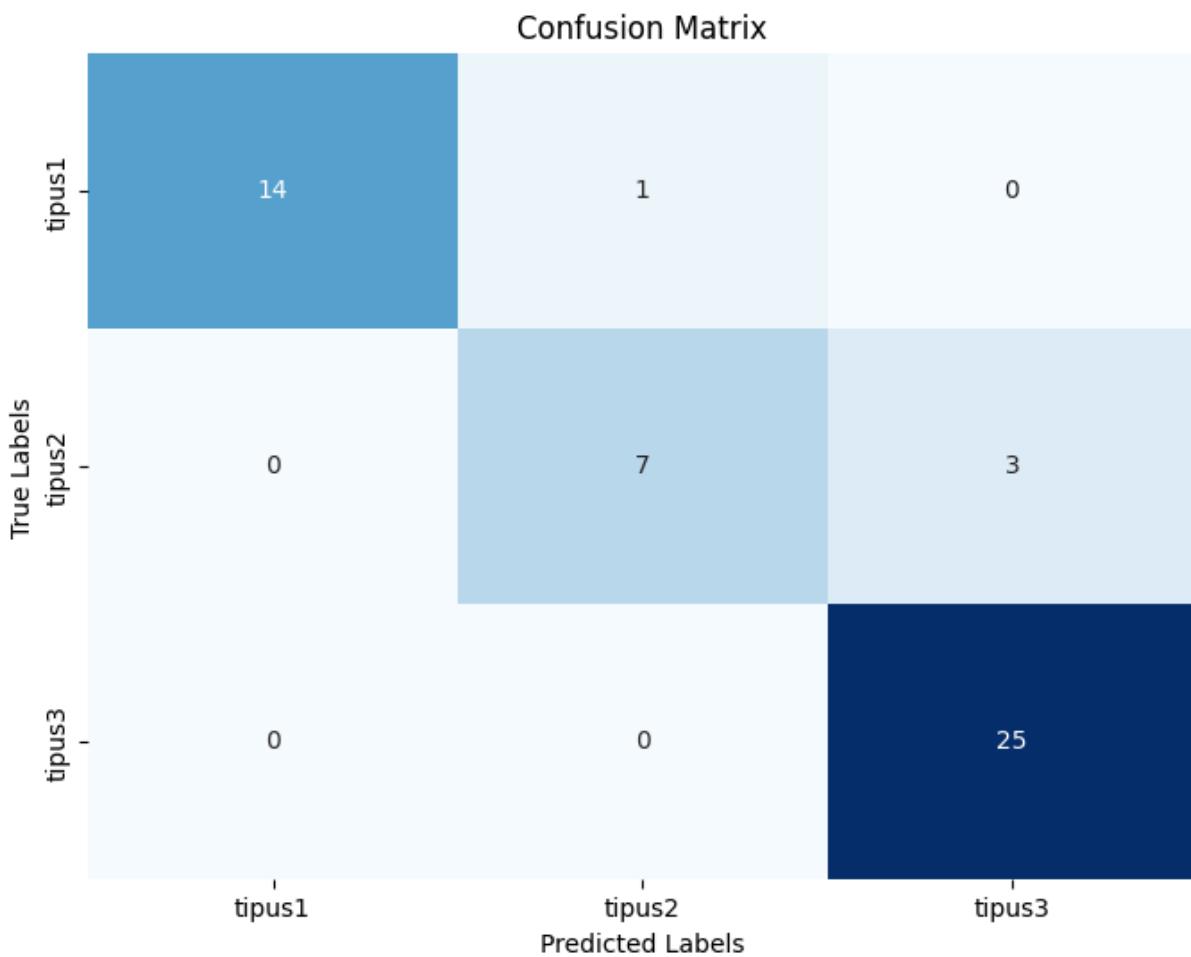
```

Resultados de validación cruzada:

	parametros	accuracy	recall	precision	f1
0	linear	0.933333	0.933333	0.938224	0.932688
1	rbf	0.946667	0.946667	0.948547	0.944175

El Kernel óptimo es 'rbf' debido a que obtiene métricas superiores al kernel 'linear' en todos los parámetros medidos

```
In [231...]: cm_svc = matriz_confusion(modelo = SVC(kernel='rbf'))
```



El SVC hace una buena clasificación, superando al KNN, aunque ligeramente inferior a los clasificadores MLP y Naive Bayes. Clasifica incorrectamente 3 muestras como tipus 3 cuando en realidad pertenecen a la clase tipus 2

## f) Árbol de Clasificación

El boosting aplicado a los árboles de clasificación (AdaBoost), es una técnica que combina múltiples clasificadores débiles (generalmente árboles de decisión simples) para formar un modelo robusto y preciso. En cada iteración, se ajustan los pesos de las instancias mal clasificadas para enfocarse en los casos más difíciles, mejorando gradualmente la precisión del modelo. Este enfoque es especialmente efectivo en conjuntos de datos desbalanceados, donde puede mejorar la detección de clases minoritarias al asignarles mayores pesos.

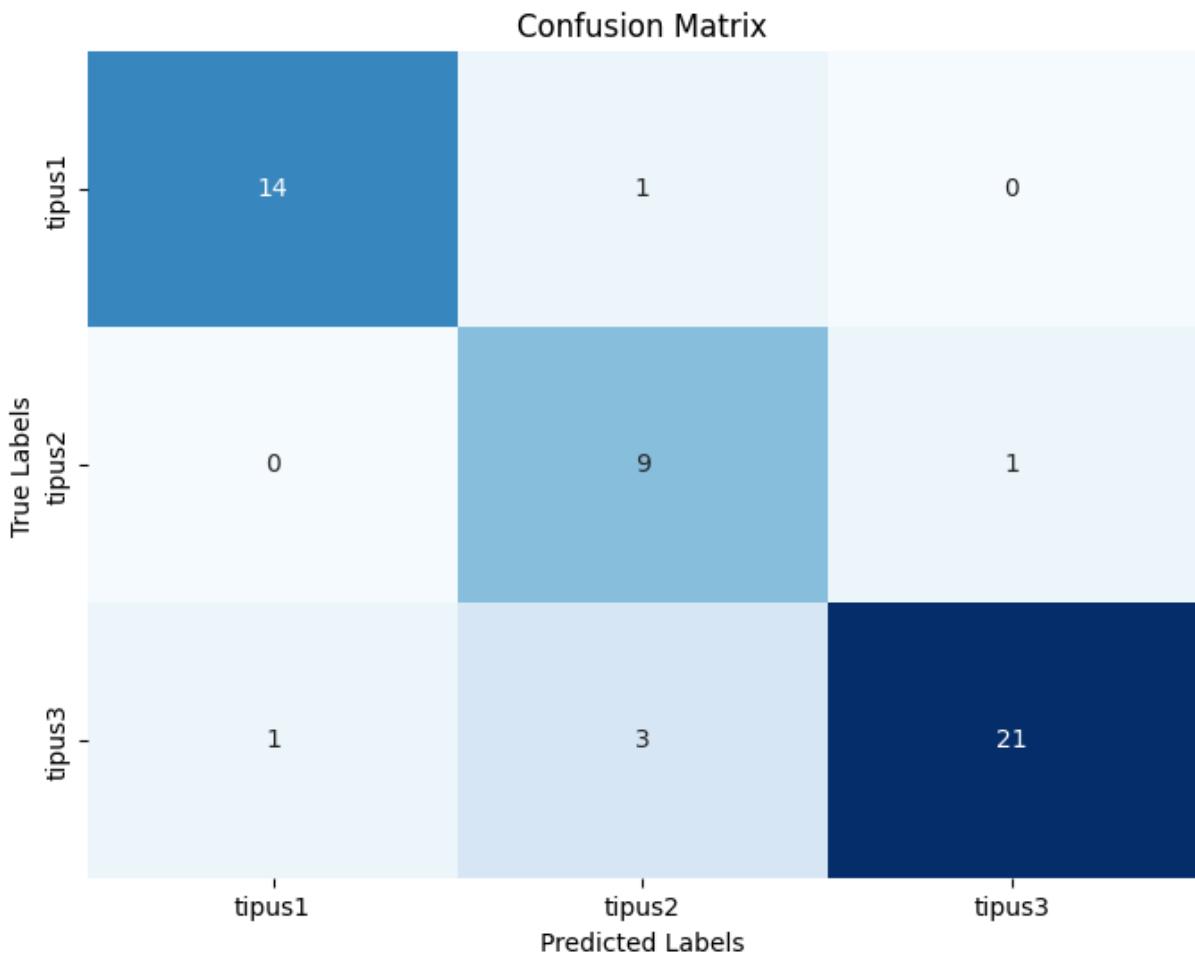
```
In [232...]  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.tree import DecisionTreeClassifier  
  
boosting = [1,10,20,30,50]  
results_dt = []  
  
for k in boosting:  
    result = crossvalidation(modelo = AdaBoostClassifier(estimator=DecisionTreeClassifi  
n_estimators=k, # Número de clasificadores  
algorithm='SAMME'))  
    results_dt.append(result)  
  
print("Resultados de validación cruzada:")  
df_dt = pd.DataFrame(results_dt)  
df_dt
```

Resultados de validación cruzada:

```
Out[232...]  
parametros accuracy recall precision f1  
0 1 0.686667 0.686667 0.554053 0.611639  
1 10 0.753333 0.753333 0.769733 0.752457  
2 20 0.826667 0.826667 0.836269 0.827343  
3 30 0.840000 0.840000 0.847674 0.838904  
4 50 0.860000 0.860000 0.867709 0.859641
```

El árbol de decisión funciona mejor con boosting y tiene una tendencia a mejorar su accuracy cuanto mayor es el número de estimadores (modelos débiles). De los valores probados, el número de estimadores que mejor resultados y rendimiento muestra es 50.

```
In [233...]  
cm_dt = matriz_confusion(modelo = AdaBoostClassifier(estimator=DecisionTreeClassifi  
n_estimators=50, # Número de clasificador  
algorithm='SAMME'))
```



Obtenemos una buena clasificación en la que el árbol de clasificación con boosting solo falla en 2 muestras. Una la clasifica como tipus 3 cuando pertenece al tipus 2 y la otra la clasifica como tipus 2 cuando realmente pertenece a tipus 3

## g) Random Forest

Random Forest es un método de aprendizaje supervisado que utiliza múltiples árboles de decisión entrenados con diferentes subconjuntos de datos y características aleatorias. Combina las predicciones de estos árboles para mejorar la precisión y reducir el sobreajuste, siendo eficaz para grandes conjuntos de datos y variables de entrada. Es robusto frente al sobreajuste gracias a la diversidad de árboles y permite medir la importancia de las características.

```
In [234...]: from sklearn.ensemble import RandomForestClassifier
estimators = [100,200]
results_rf = []

for k in estimators:
    result = crossvalidation(modelo = RandomForestClassifier(n_estimators=k, random_
    results_rf.append(result)

print("Resultados de validación cruzada:")
```

```
df_rf = pd.DataFrame(results_rf)
df_rf
```

Resultados de validación cruzada:

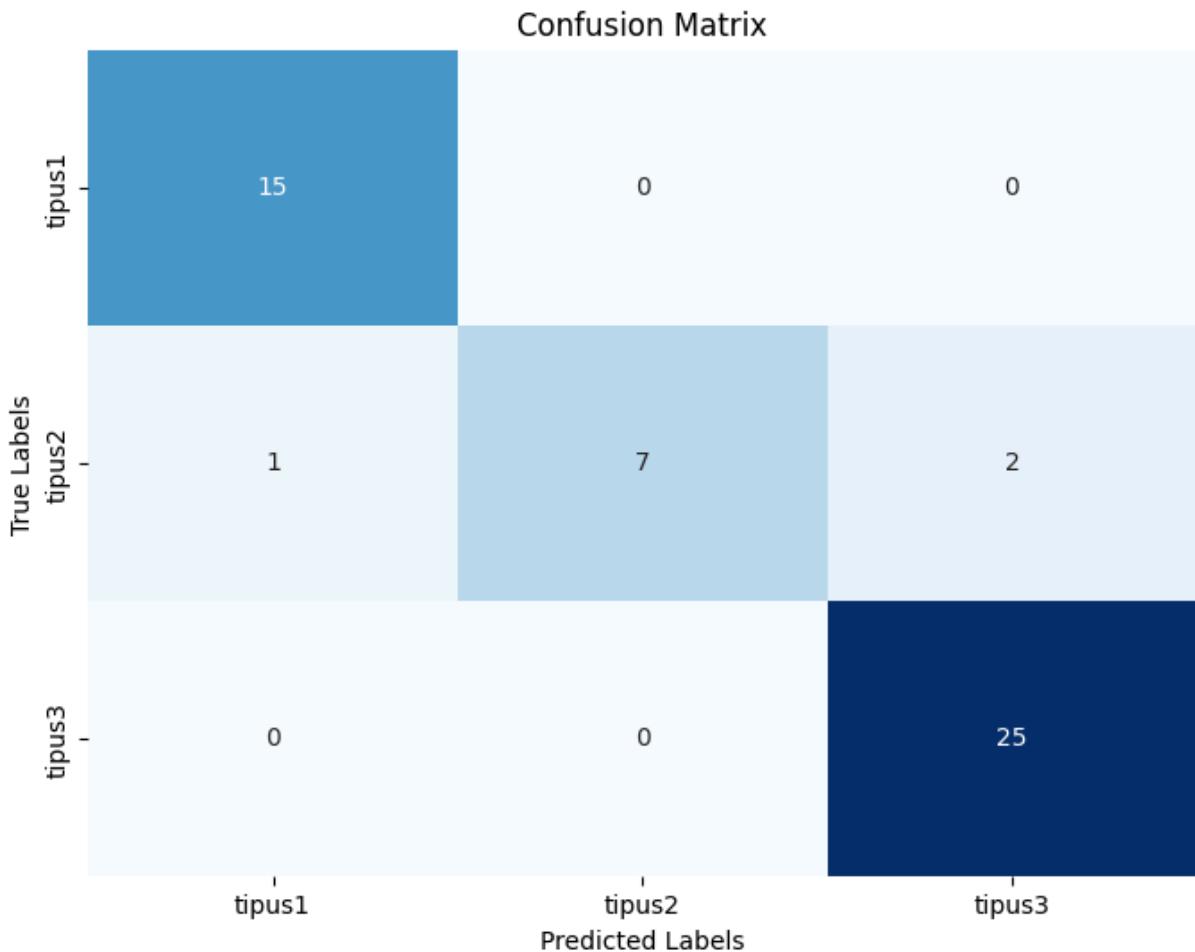
Out[234...]

	parametros	accuracy	recall	precision	f1
0	100	0.926667	0.926667	0.929791	0.923395
1	200	0.920000	0.920000	0.922428	0.915693

El random forest con un número de árboles = 100 tiene un rendimiento ligeramente superior al random forest con un número de árboles = 200

In [235...]

```
cm_rf = matriz_confusion(modelo = RandomForestClassifier(n_estimators=100, random_s
```



El random forest hace una buena clasificación pero clasifica 3 muestras de forma errónea en la clase tipus 3 cuando realmente pertenecen a la clase tipus 2

### 3. Discusión y conclusión sobre el rendimiento de los algoritmos

In [236...]

```
modelos = {
    'KNN': KNeighborsClassifier(n_neighbors=3),
```

```

'NB':MultinomialNB(alpha=1),
'MLP':MLPClassifier(hidden_layer_sizes=(30,), activation='relu', solver='lbfgs'),
'SVM':SVC(kernel='rbf'),
'DT':AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
                        n_estimators=50, # Número de
                        algorithm='SAMME'),
'RF':RandomForestClassifier(n_estimators=100, random_state=42),
}
results_total = []

for model_name, model in modelos.items():
    result = crossvalidation(modelo=model)
    result['Model'] = model_name
    results_total.append(result)

df_total = pd.DataFrame(results_total).drop(['parametros'], axis=1)

# Cambiamos orden columna Model
cols = list(df_total.columns)
cols = ['Model'] + [col for col in cols if col != 'Model']
df_total = df_total[cols]

df_sorted = df_total.sort_values(by='accuracy', ascending=False)
df_sorted

```

Out[236...]

	<b>Model</b>	<b>accuracy</b>	<b>recall</b>	<b>precision</b>	<b>f1</b>
<b>3</b>	SVM	0.946667	0.946667	0.948547	0.944175
<b>1</b>	NB	0.940000	0.940000	0.945483	0.940969
<b>2</b>	MLP	0.940000	0.940000	0.942015	0.939584
<b>5</b>	RF	0.926667	0.926667	0.929791	0.923395
<b>0</b>	KNN	0.913333	0.913333	0.920802	0.907258
<b>4</b>	DT	0.860000	0.860000	0.867709	0.859641

Esta tabla muestra el rendimiento de todos los algoritmos de clasificación implementados anteriormente, entrenados con los parámetros más óptimos probados.

El algoritmo que muestra un mejor rendimiento es el SVM con un kernel 'rbf' superior al resto de algoritmos en accuracy (94.66%), recall (94.66%), precision (94.85%) y F1-score (94.42%). El segundo mejor algoritmo es el Naive Bayes classifier con un 94.00% de accuracy, seguido del MLP (con una capa oculta de 30 unidades) con la misma accuracy y del random forest (n=100) con una accuracy del 92.66%. Los algoritmos que han mostrado un peor rendimiento son el KNN (accuracy = 91.33%) y el Decision Tree con boosting (accuracy = 86.00%).

En este ejercicio hemos explorado y analizado los datos RNA-seq de expresión de 3 tipos diferentes de cáncer de piel. Hemos demostrado que existen diferencias apreciables a simple vista entre los datos de expresión de los 3 tipos de cáncer mediante un PCA plot. Además, hemos entrenado y optimizado 6 algoritmos distintos mediante un 'Cross Validation 3 folds' para la predicción de la clasificación de las variables en estos 3 tipos y hemos comparado que algoritmos son los mejores y cuáles son sus fallos. Finalmente hemos obtenido una accuracy máxima de 94.66% con un SVM.