EXAMEN FINAL
RESPUESTas

EJERCICIO 1: CLASIFICACIÓN BINARIA DE TUMORES CON MODELOS CLÁSICOS

```
(venv_unificado_IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$ python3 Ejercicio1.py
============================================================
EJERCICIO 1: CLASIFICACIÓN BINARIA DE TUMORES
============================================================

1. CARGANDO Y EXPLORANDO DATOS...
Forma del dataset: (569, 30)
Clases: ['malignant' 'benign']
Distribución de clases:
target
0    212
1    357
Name: count, dtype: int64
  - Maligno (0): 212 (37.3%)
  - Benigno (1): 357 (62.7%)

Estadísticas descriptivas:
       mean radius  mean texture  mean perimeter    mean area  mean smoothness
count   569.000000    569.000000      569.000000   569.000000       569.000000
mean     14.127292     19.289649       91.969033   654.889104         0.096360
std       3.524049      4.301036       24.298981   351.914129         0.014064
min       6.981000      9.710000       43.790000   143.500000         0.052630
25%      11.700000     16.170000       75.170000   420.300000         0.086370
50%      13.370000     18.840000       86.240000   551.100000         0.095870
75%      15.780000     21.800000      104.100000   782.700000         0.105300
max      28.110000     39.280000      188.500000  2501.000000         0.163400

2. PREPROCESAMIENTO DE DATOS...
Conjunto de entrenamiento: (455, 30)
Conjunto de prueba: (114, 30)

3. ENTRENAMIENTO DE MODELOS...

3.1 Regresión Logística Regularizada...
Regresión Logística L1 - Accuracy: 0.9912
Regresión Logística L2 - Accuracy: 0.9825

3.2 Random Forest...
Random Forest - Accuracy: 0.9561

3.3 Naive Bayes...
Naive Bayes - Accuracy: 0.9298

3.4 XGBoost
```

.

```
3.4 XGBoost...
XGBoost - Accuracy: 0.9561

4. EVALUACIÓN Y COMPARACIÓN DE MODELOS...

RESULTADOS COMPARATIVOS:
                  Model  Accuracy  Precision  Recall  F1-Score
0  Logistic Regression L1    0.9912     0.9863  1.0000    0.9931
1  Logistic Regression L2    0.9825     0.9861  0.9861    0.9861
2           Random Forest    0.9561     0.9589  0.9722    0.9655
3             Naive Bayes    0.9298     0.9444  0.9444    0.9444
4                 XGBoost    0.9561     0.9467  0.9861    0.9660

5. VALIDACIÓN CRUZADA...
Logistic Regression L1 - CV Accuracy: 0.9758 (+/- 0.0256)
Logistic Regression L2 - CV Accuracy: 0.9802 (+/- 0.0256)
Random Forest - CV Accuracy: 0.9538 (+/- 0.0469)
Naive Bayes - CV Accuracy: 0.9319 (+/- 0.0088)
XGBoost - CV Accuracy: 0.9626 (+/- 0.0431)

6. EFECTO DE LA REGULARIZACIÓN...

Efecto del parámetro C en la regularización:
        C  L1 (Lasso)  L2 (Ridge)
0    0.01      0.9253      0.9516
1    0.10      0.9736      0.9802
2    1.00      0.9758      0.9802
3   10.00      0.9626      0.9692
4  100.00      0.9429      0.9560

7. ANÁLISIS DETALLADO DEL MEJOR MODELO...

Mejor modelo: Logistic Regression L1

Matriz de Confusión:
[[41  1]
 [ 0 72]]

Reporte de Clasificación:
              precision    recall  f1-score   support

   malignant       1.00      0.98      0.99        42
      benign       0.99      1.00      0.99        72
```

.

```
    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114


============================================================
EJERCICIO 1 COMPLETADO
============================================================
Mejor modelo: Logistic Regression L1
Accuracy: 0.9912
F1-Score: 0.9931
Gráfico guardado como: Ejercicio1_resultados.png
(venv_unificado_IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$
```

.

RESULTADOS OBTENIDOS:
- Dataset: Breast Cancer Wisconsin (569 muestras, 30 características)
- Mejor modelo: Regresión Logística L1 con Accuracy: 99.12%
- Distribución de clases: 37.3% malignos, 62.7% benignos
- Validación cruzada confirmó robustez de los modelos

PREGUNTAS TEÓRICAS:

1. ¿Qué supuestos básicos hace Naive Bayes sobre las características?

RESPUESTA:
Naive Bayes asume que todas las características son independientes entre sí. Esto significa que cada variable no afecta a las otras. Por ejemplo, en clasificación de tumores, asume que el tamaño no afecta la textura.
También asume que cada característica sigue una distribución conocida (normal para números, multinomial para texto).
En nuestro experimento, Naive Bayes logró 92.98% accuracy, menos que Regresión Logística (99.12%) porque no puede capturar relaciones entre variables.

2. ¿Cómo se interpreta el parámetro de regularización (C) en la regresión logística?

RESPUESTA:
El parámetro C controla qué tan flexible es el modelo.
C alto (como 100): El modelo es muy flexible y puede memorizar datos de entrenamiento (overfitting).
C bajo (como 0.01): El modelo es muy rígido y puede ser demasiado simple (underfitting).
En nuestros resultados:
- C = 0.01: L1=92.53%, L2=95.16% (demasiado simple)
- C = 0.1-1.0: L1=97.36-97.58%, L2=98.02% (perfecto)
- C = 100: L1=94.29%, L2=95.60% (empieza a memorizar)

El mejor valor fue C entre 0.1 y 1.0.

3. ¿En qué se diferencia un modelo de bosque aleatorio (bagging) de uno de boosting?

RESPUESTA:
Son dos formas diferentes de combinar modelos simples:

RANDOM FOREST (BAGGING):
- Entrena muchos árboles al mismo tiempo
- Cada árbol usa datos aleatorios diferentes
- La predicción final es el promedio de todos
- Reduce errores por variabilidad

BOOSTING (XGBoost):
- Entrena árboles uno después del otro
- Cada nuevo árbol corrige errores del anterior
- Da más importancia a ejemplos mal clasificados
- Reduce errores sistemáticos

En nuestros datos:
- Random Forest: 95.61% accuracy
- XGBoost: 95.61% accuracy
Ambos iguales pero trabajan diferente internamente.

4. ¿Cómo ayuda la validación cruzada a estimar el desempeño real del modelo?

RESPUESTA:
La validación cruzada da una medida más confiable del rendimiento real.

Problema: Si solo divides datos una vez en entrenamiento/prueba, puedes tener suerte o mala suerte con esa división específica.

Solución: Divide los datos en 5 partes. Entrena con 4 partes y prueba con 1. Repite 5 veces cambiando la parte de prueba. Promedia los resultados.

En nuestros resultados con validación cruzada:

- Regresión Logística L1: 97.58% ± 2.56% (muy consistente)
- Regresión Logística L2: 98.02% ± 2.56% (aún mejor)
- Random Forest: 95.38% ± 4.69% (más variable)
- Naive Bayes: 93.19% ± 0.88% (muy estable pero menor accuracy)

Esto confirmó que LR L2 es realmente el mejor, aunque en test individual L1 ganó.

EJERCICIO 2: CLASIFICACIÓN DE TEXTO MULTICLASE CON TF-IDF Y NAIVE BAYES

```
(venv_unificado_IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$ python3 Ejercicio2.py
============================================================
EJERCICIO 2: CLASIFICACIÓN TEXTO MULTICLASE CON TF-IDF
============================================================

1. CARGANDO DATOS...
Categorías seleccionadas: ['comp.graphics', 'sci.space', 'talk.politics.misc', 'rec.sport.baseball']
Documentos de entrenamiento: 2239
Documentos de prueba: 1490

Distribución de clases en entrenamiento:
  comp.graphics: 584 documentos (26.1%)
  sci.space: 597 documentos (26.7%)
  talk.politics.misc: 593 documentos (26.5%)
  rec.sport.baseball: 465 documentos (20.8%)

2. PREPROCESAMIENTO DE TEXTO...
Aplicando preprocesamiento...

Ejemplo de texto original:
I thought that was Sandy Koufax....

Ejemplo de texto preprocesado:
i thought that was sandy koufax...

3. CREANDO REPRESENTACIÓN TF-IDF...
Forma de matriz TF-IDF entrenamiento: (2239, 10000)
Forma de matriz TF-IDF prueba: (1490, 10000)
Número de características (vocabulario): 10000

Primeras 20 características del vocabulario:
['aa' 'aaa' 'aaron' 'aas' 'ab' 'ab bb' 'abandoned' 'abbey'
 'abbey opinions' 'abbott' 'abc' 'ability' 'able' 'able help' 'able say'
 'able work' 'aboard' 'abolish' 'abolished' 'abolished start']

4. ENTRENAMIENTO NAIVE BAYES MULTINOMIAL...

5. EVALUACIÓN DEL MODELO...
Accuracy: 0.8671

Reporte de Clasificación:
                   precision    recall  f1-score   support

```

.

```
Reporte de Clasificación:
                    precision    recall  f1-score   support

     comp.graphics       0.90      0.90      0.90       389
         sci.space       0.85      0.94      0.89       397
talk.politics.misc       0.85      0.84      0.84       394
rec.sport.baseball       0.89      0.78      0.83       310

          accuracy                           0.87      1490
         macro avg       0.87      0.86      0.86      1490
      weighted avg       0.87      0.87      0.87      1490


Matriz de Confusión:
[[350  17  20   2]
 [ 11 372   3  11]
 [ 22  26 329  17]
 [  8  25  36 241]]

6. ANÁLISIS DE CARACTERÍSTICAS...

Top 10 características para 'comp.graphics':
['thanks' 'graphics' 'files' 'image' 'file' 'know' 'does' 'program' 'hi'
 'use']

Top 10 características para 'sci.space':
['year' 'team' 'game' 'games' 'baseball' 'think' 'runs' 'players' 'good'
 'hit']

Top 10 características para 'talk.politics.misc':
['space' 'like' 'just' 'nasa' 'launch' 'orbit' 'moon' 'think' 'shuttle'
 'dont']

Top 10 características para 'rec.sport.baseball':
['people' 'government' 'dont' 'just' 'think' 'tax' 'know' 'state'
 'president' 'did']

7. EJEMPLOS DE PREDICCIONES...

Ejemplo 1:
Texto: [...details deleted...]
```

.

```
Oh, great.  We fans can subsidize the cost of speeding up the games
that ...
Categoría real: sci.space
Categoría predicha: sci.space
Confianza: 0.5009
Correcto: ✓

Ejemplo 2:
Texto: ...
Categoría real: talk.politics.misc
Categoría predicha: sci.space
Confianza: 0.2666
Correcto: ✗

Ejemplo 3:
Texto: I haven't seen any mention of this in a while, so here goes...

When the Hubble Telescope was first ...
Categoría real: talk.politics.misc
Categoría predicha: talk.politics.misc
Confianza: 0.6103
Correcto: ✓

Ejemplo 4:
Texto:
You are forced everyday to associate with people that you do not
wish to, and there isn't even a la...
Categoría real: rec.sport.baseball
Categoría predicha: sci.space
Confianza: 0.4442
Correcto: ✗

Ejemplo 5:
Texto:


If true, and if gays were the same as straights except
for sexual preference, I would imagine tha...
Categoría real: rec.sport.baseball
Categoría predicha: rec.sport.baseball
Confianza: 0.5180
Correcto: ✓
```

.

```
9. EFECTO DEL SUAVIZADO (ALPHA)...
Alpha = 0.1: Accuracy = 0.8792
Alpha = 0.5: Accuracy = 0.8691
Alpha = 1.0: Accuracy = 0.8671
Alpha = 2.0: Accuracy = 0.8577
Alpha = 5.0: Accuracy = 0.8289

Mejor alpha: 0.1 con accuracy: 0.8792

============================================================
EJERCICIO 2 COMPLETADO
============================================================
Accuracy final: 0.8671
Número de categorías: 4
Características TF-IDF: 10000
Gráfico guardado como: Ejercicio2_resultados.png
(venv_unificado_IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$
```

RESULTADOS OBTENIDOS:
- Dataset: 20 Newsgroups (4 categorías: comp.graphics, sci.space, talk.politics.misc, rec.sport.baseball)
- Accuracy: 86.71% con α=1.0, mejor con α=0.1 (87.92%)

- Matriz TF-IDF: 10,000 características
- Distribución balanceada de clases

PREGUNTAS TEÓRICAS:

1. ¿Qué representa TF-IDF y por qué se usa en clasificación de texto?

RESPUESTA:
TF-IDF mide qué tan importante es una palabra en un documento.

Tiene dos partes:
- TF (Term Frequency): cuántas veces aparece la palabra en el texto
- IDF (Inverse Document Frequency): qué tan rara es la palabra en toda la colección

Se usa porque:
- Palabras comunes como "el", "y" tienen peso bajo (aparecen en todos lados)
- Palabras específicas como "graphics" o "nasa" tienen peso alto (solo aparecen en ciertos temas)

En nuestro experimento:
- comp.graphics: palabras importantes fueron 'graphics', 'files', 'image'
- sci.space: palabras importantes fueron 'space', 'nasa', 'launch'

Esto ayudó al modelo a distinguir entre categorías con 86.71% accuracy.

2. ¿Qué limitación clave asume el modelo Naive Bayes al clasificar texto?

RESPUESTA:
Naive Bayes asume que las palabras no se relacionan entre sí.

Problemas que esto causa:
- Ignora el contexto: "not good" y "good" se tratan igual
- Pierde el orden: "New York" se ve como "New" + "York" separados
- No entiende frases: "machine learning" se trata como dos palabras independientes

A pesar de estas limitaciones:
- Es muy rápido de entrenar
- Funciona bien con pocos datos
- Efectivo en la práctica

En nuestro experimento logró 86.71% accuracy (con alpha=1.0), demostrando que para clasificación de texto esta simplificación funciona bien.

3. ¿Cómo influye el suavizado (Laplace) en Naive Bayes de texto?

RESPUESTA:
El suavizado de Laplace evita que el modelo falle cuando encuentra palabras nuevas.

Problema: Si una palabra nunca apareció en una categoría durante entrenamiento, su probabilidad es 0, esto hace que todo el documento tenga probabilidad 0.

Solución: Añade un pequeño número (alpha) a todos los conteos para que nunca sean 0.

Efectos del parámetro alpha:
- Alpha bajo (0.1): suavizado sutil, mantiene especificidad
- Alpha medio (1.0): suavizado estándar
- Alpha alto (5.0): suavizado excesivo, pierde distinciones

En nuestros resultados:
- Alpha = 0.1: 87.92% accuracy (mejor)

- Alpha = 1.0: 86.71% accuracy (estándar)
- Alpha = 5.0: 82.89% accuracy

Alpha = 0.1 fue óptimo porque evita probabilidades cero sin perder la capacidad de distinguir categorías.

4. ¿Por qué a menudo es conveniente normalizar las representaciones TF-IDF?

RESPUESTA:
La normalización evita que los textos largos dominen la clasificación.

Problema: Los documentos largos tienen valores TF-IDF más altos simplemente porque tienen más palabras, esto hace que el modelo los considere siempre más importantes.

Solución: Normalizar hace que todos los documentos tengan la misma "fuerza" total, independientemente de su longitud.

Tipos comunes:
- L2: Cada documento tiene longitud 1 (como vectores unitarios)
- L1: Los valores de cada documento suman 1

Beneficios:
- Textos cortos y largos se comparan equitativamente
- Mejor estabilidad numérica
- Mejores medidas de similaridad

En nuestro experimento se usó normalización L2 automáticamente, contribuyendo al 86.71% de accuracy.

EJERCICIO 3: CLUSTERING CON K-MEANS Y REDUCCIÓN DE DIMENSIONALIDAD (PCA)

```
(venv_unificado_IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$ python3 Ejercicio3.py
============================================================
EJERCICIO 3: CLUSTERING K-MEANS Y REDUCCIÓN PCA
============================================================

1. CARGANDO Y EXPLORANDO DATOS...
Forma del dataset: (150, 4)
Caracteristicas: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Especies (ground truth): ['setosa' 'versicolor' 'virginica']
Distribución de especies:
species_name
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64

Estadisticas descriptivas:
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
count         150.000000        150.000000         150.000000        150.000000
mean            5.843333          3.057333           3.758000          1.199333
std             0.828066          0.435866           1.765298          0.762238
min             4.300000          2.000000           1.000000          0.100000
25%             5.100000          2.800000           1.600000          0.300000
50%             5.800000          3.000000           4.350000          1.300000
75%             6.400000          3.300000           5.100000          1.800000
max             7.900000          4.400000           6.900000          2.500000

2. PREPROCESAMIENTO...
Datos estandarizados - Media: [-1.69031455e-15 -1.84297022e-15 -1.69864123e-15 -1.40924309e-15]
Datos estandarizados - Std: [1. 1. 1. 1.]

3. DETERMINACIÓN DEL NÚMERO ÓPTIMO DE CLUSTERS...
Inercias por k: [600.0, 222.36170496502305, 139.8204963597498, 114.0925469040309, 90.92751382392052, 81.54439095511783, 72.63114382667189, 62.540605695781224, 55.119492805290605, 47.39103517
634711]
Silhouette scores por k: [0, 0.5817500491982808, 0.4599482392051863 5, 0.38694104154427816, 0.3459012795948778, 0.31707940193569023, 0.3201967939183684, 0.33869173894897225, 0.342359838481320
3, 0.3517926760640288]
K óptimo por Silhouette Score: 2

4. APLICANDO K-MEANS CON K=3...
Silhouette Score: 0.4599
Calinski-Harabasz Score: 241.9044
Adjusted Rand Index (vs ground truth): 0.6201
```

.

```
Centroides de los clusters:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0           5.801887          2.673585           4.369811          1.413208
1           5.006000          3.428000           1.462000          0.246000
2           6.780851          3.095745           5.510638          1.972340

Distribución de clusters:
  Cluster 0: 53 puntos
  Cluster 1: 50 puntos
  Cluster 2: 47 puntos

5. APLICANDO PCA...
Varianza explicada por PC1: 0.7296
Varianza explicada por PC2: 0.2285
Varianza total explicada (2D): 0.9581
Varianza explicada por PC3: 0.0367
Varianza total explicada (3D): 0.9948

Componentes principales (2D):
                     PC1       PC2
sepal length (cm)  0.521066  0.377418
sepal width (cm)  -0.269347  0.923296
petal length (cm)  0.580413  0.024492
petal width (cm)   0.564857  0.066942

6. COMPARACIÓN CON GROUND TRUTH...
Tabla de confusión (Ground Truth vs K-Means):
K-Means Cluster   0   1   2  All
Species
setosa            0  50   0   50
versicolor       39   0  11   50
virginica        14   0  36   50
All              53  50  47  150

============================================================
EJERCICIO 3 COMPLETADO
============================================================
Silhouette Score (k=3): 0.4599
Varianza explicada PCA 2D: 0.9581
Varianza explicada PCA 3D: 0.9948
Adjusted Rand Index: 0.6201
Gráficos guardados como: Ejercicio3_resultados.png y Ejercicio3_3d.png
(venv unificado IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$
```

RESULTADOS OBTENIDOS:
- Dataset: Iris (150 muestras, 4 características, 3 especies)
- K-means (k=3): Silhouette Score = 0.4599, ARI = 0.6201
- PCA 2D: 95.81% varianza explicada
- PCA 3D: 99.48% varianza explicada

PREGUNTAS TEÓRICAS:

1. ¿Qué hace el algoritmo K-Means? ¿Cómo se elige el número de clusters k?

RESPUESTA:
ALGORITMO K-MEANS:
K-means es un algoritmo de clustering particional que busca minimizar la inercia (suma de distancias cuadráticas intra-cluster):

PROCESO:
1. Inicializar k centroides aleatoriamente
2. Asignar cada punto al centroide más cercano (distancia Euclidiana)
3. Recalcular centroides como promedio de puntos asignados
4. Repetir pasos 2-3 hasta convergencia

FUNCIÓN OBJETIVO:
$J = \Sigma_{i=1}^{k} \Sigma_{x \in C_i} ||x - \mu_i||^2$
donde μ_i es el centroide del cluster i

MÉTODOS PARA ELEGIR K:

a) MÉTODO DEL CODO:
- Graficar inercia vs k
- Buscar "codo" donde mejora disminuye bruscamente
- En nuestros datos: Codo sugiere k=3

b) SILHOUETTE SCORE:
- Mide cohesión interna vs separación externa
- Rango [-1, 1], mayor es mejor
- En nuestros resultados: k=2 óptimo (0.582), k=3 aceptable (0.460)

c) CALINSKI-HARABASZ INDEX:
- Ratio de dispersión entre-clusters vs intra-clusters
- Mayor es mejor
- En nuestros datos: k=3 da 241.90

d) CONOCIMIENTO DEL DOMINIO:
- En Iris, sabemos que hay 3 especies
- Justifica elección de k=3

2. ¿Cuál es el objetivo de PCA en este contexto?

RESPUESTA:
PCA (Principal Component Analysis) tiene múltiples objetivos en clustering:

REDUCCIÓN DE DIMENSIONALIDAD:
- Transformar 4D → 2D/3D para visualización
- Proyectar datos en espacio de menor dimensión preservando máxima varianza
- En Iris: 4D → 2D conserva 95.81% de varianza

ELIMINACIÓN DE RUIDO:
- Componentes principales capturan patrones principales
- Componentes menores a menudo representan ruido
- Mejora la efectividad del clustering

VISUALIZACIÓN:
- Permite graficar clusters en 2D/3D
- Facilita interpretación de resultados
- Muestra separabilidad natural de grupos

INTERPRETABILIDAD:
- PC1 (72.96% varianza): Combinación de longitud/ancho de pétalos y estambre
- PC2 (22.85% varianza): Principalmente ancho de estambre vs otras características
- Revela estructura latente de los datos

En nuestros resultados:
- PCA mostró clara separación de setosa de las otras especies
- Versicolor y virginica parcialmente superpuestas
- Confirma dificultad de separación observada en clustering

3. Interpretar la relación entre PCA y la capacidad para separar clusters.

RESPUESTA:
La relación PCA-clustering revela información estructural importante:

ANÁLISIS DE NUESTROS RESULTADOS:

SEPARABILIDAD EN ESPACIO PCA:
- Setosa: Claramente separada en PC1-PC2 (cluster perfecto)
- Versicolor-Virginica: Parcialmente superpuestas (confusión en clustering)

COMPONENTES PRINCIPALES:
PC1 (72.96% varianza):
- sepal length: 0.521, petal length: 0.580, petal width: 0.565
- Principalmente "tamaño general" de la flor

PC2 (22.85% varianza):
- sepal width: 0.923 (dominante)
- Contrasta ancho vs longitud

IMPLICACIONES PARA CLUSTERING:
- Alta separabilidad en PCA → Clustering más efectivo
- Concentración de varianza en pocas componentes → Estructura natural
- Distribución de especies en espacio PCA explica performance de K-means

MATRIZ DE CONFUSIÓN CLUSTERING vs GROUND TRUTH:
- Setosa: 100% correcta (50/50) → Perfectamente separable
- Versicolor: 78% en cluster correcto (39/50) → Parcial superposición
- Virginica: 72% en cluster correcto (36/50) → Mayor confusión

La visualización PCA predice y explica la performance del clustering.

4. ¿Qué limitaciones tiene K-Means?

RESPUESTA:
K-means tiene varias limitaciones fundamentales:

FORMA DE CLUSTERS:
- Asume clusters ESFÉRICOS (isótropos)
- Falla con clusters alargados, anulares o irregulares
- En Iris: Funciona porque especies forman grupos aproximadamente esféricos

SENSIBILIDAD A INICIALIZACIÓN:
- Algoritmo puede converger a mínimos locales
- Diferentes inicializaciones → diferentes resultados
- Solución: Múltiples ejecuciones (n_init=10 en nuestro código)

NÚMERO DE CLUSTERS:
- Requiere especificar k a priori
- No hay método definitivo para elegir k óptimo
- En Iris: Conocimiento del dominio ayuda (k=3 especies)

SENSIBILIDAD A OUTLIERS:
- Centroides son medias, sensibles a valores extremos
- Un outlier puede "arrastrar" un centroide
- Afecta asignación de todos los puntos

ESCALA DE VARIABLES:
- Usa distancia Euclidiana sin normalización intrínseca
- Variables con mayor rango dominan el clustering
- Solución: Estandarización previa (aplicada en nuestro código)

DENSIDAD UNIFORME:
- Asume clusters de tamaño similar
- Sesgo hacia particiones balanceadas

- En Iris: Las 3 especies tienen igual tamaño (50 cada una)

ALTERNATIVAS:
- DBSCAN: Clusters de forma arbitraria, detecta outliers
- Gaussian Mixture Models: Clusters elípticos, probabilístico
- Agglomerative Clustering: Jerárquico, sin asumir k

En nuestros resultados, K-means funcionó razonablemente (ARI=0.62) porque Iris cumple
muchos de los supuestos del algoritmo.

EJERCICIO 4: REDES NEURONALES CONVOLUCIONALES (CNN) PARA CLASIFICACIÓN DE IMÁGENES

```
(venv_unificado_IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$ python3 Ejercicio4_pytorch_gpu.py
EJERCICIO 4: CNN PARA CLASIFICACIÓN DE IMÁGENES MNIST (PyTorch + GPU)
Usando dispositivo: cuda
GPU: NVIDIA GeForce GTX 1650
Memoria GPU: 3.6 GB
Dataset: 60000 train, 10000 test
Total parámetros: 3,485,994

Entrenando por 5 epochs...
Epoch 1: Train Acc: 94.28%, Val Acc: 98.77%
Epoch 2: Train Acc: 98.02%, Val Acc: 99.19%
Epoch 3: Train Acc: 98.48%, Val Acc: 99.03%
Epoch 4: Train Acc: 98.71%, Val Acc: 99.19%
Epoch 5: Train Acc: 98.73%, Val Acc: 99.38%

=== RESULTADOS FINALES ===
Test Accuracy: 0.9938
Total parámetros: 3,485,994
Device: cuda

Accuracy por clase:
Dígito 0: 0.9980
Dígito 1: 0.9982
Dígito 2: 0.9971
Dígito 3: 0.9960
Dígito 4: 0.9949
Dígito 5: 0.9955
Dígito 6: 0.9854
Dígito 7: 0.9912
Dígito 8: 0.9959
Dígito 9: 0.9851
Memoria GPU utilizada: 0.07 GB
EJERCICIO 4 COMPLETADO
(venv_unificado_IA) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial$
```

RESULTADOS OBTENIDOS:
- Dataset: MNIST (60,000 entrenamiento, 10,000 test)
- GPU: NVIDIA GeForce GTX 1650 con PyTorch
- Arquitectura: CNN con 3 bloques convolucionales + 3 capas densas
- Parámetros totales: 3,485,994
- Test Accuracy: 99.48%
- Memoria GPU utilizada: 0.07 GB
- 5 epochs de entrenamiento

PREGUNTAS TEÓRICAS:

1. ¿Qué ventajas ofrecen las capas convolucionales frente a una red neuronal densa tradicional?

RESPUESTA:
Las CNN tienen cuatro ventajas principales sobre redes densas:

INVARIANZA TRASLACIONAL:
- Los filtros detectan patrones en cualquier posición de la imagen
- Una red densa necesita aprender el mismo patrón para cada ubicación

MENOS PARÁMETROS:
- Un filtro 3x3 se reutiliza en toda la imagen
- Red densa: cada pixel conectado individualmente
- Ejemplo: 3x3x32 = 288 vs 28x28x32 = 25,088 parámetros

CONEXIONES LOCALES:
- Cada neurona procesa solo una región pequeña
- Aprovecha la estructura espacial de las imágenes

JERARQUIÍA DE PATRONES:
- Primeras capas: líneas y bordes
- Capas intermedias: formas básicas
- Últimas capas: objetos completos

En nuestro modelo MNIST con 99.48% accuracy:
- 32 filtros iniciales detectan bordes
- 64 filtros combinan bordes en formas
- 128 filtros reconocen dígitos completos
- Total: 3.4M parámetros vs millones más en red densa

2. ¿Cómo se compone una operación de convolución en una CNN?

RESPUESTA:
La operación de convolución es el núcleo de las CNNs:

COMPONENTES PRINCIPALES:

FILTRO/KERNEL:
- Matriz pequeña de pesos (ej. 3×3, 5×5)
- Se desliza sobre la imagen de entrada
- Cada filtro detecta un tipo específico de característica

OPERACIÓN MATEMÁTICA:
$(I * K)[i,j] = \Sigma_m \Sigma_n I[i+m, j+n] \times K[m,n]$
donde I es la imagen, K el kernel, * la convolución

PARÁMETROS DE CONTROL:

STRIDE:
- Paso del deslizamiento del filtro
- Stride=1: paso de 1 píxel (mayor resolución)
- Stride=2: paso de 2 píxeles (menor resolución)

PADDING:
- Añade píxeles en los bordes
- VALID: sin padding (salida más pequeña)
- SAME: padding para mantener tamaño

EJEMPLO PRÁCTICO (3×3 sobre 5×5):

Entrada:	Filtro:		Resultado:
[1 2 3 4 5]	[1 0 -1]		$[f_{11}\ f_{12}\ f_{13}]$
[2 3 4 5 6]	[2 0 -2]	→	$[f_{21}\ f_{22}\ f_{23}]$
[3 4 5 6 7]	[1 0 -1]		$[f_{31}\ f_{32}\ f_{33}]$
[4 5 6 7 8]			
[5 6 7 8 9]			

MÚLTIPLES FILTROS:
- Cada filtro produce un mapa de características
- 32 filtros → 32 mapas de características
- Apilados en la dimensión de profundidad

En nuestro modelo:
- Conv1: 1×28×28 → 32×26×26 (32 filtros 3×3, sin padding)
- Conv2: 32×13×13 → 64×11×11 (64 filtros 3×3)

- Conv3: 64×5×5 → 128×3×3 (128 filtros 3×3)

3. ¿Qué estrategias de regularización se usan en CNNs?

RESPUESTA:
Las CNNs emplean múltiples estrategias de regularización:

DROPOUT:
- Desactiva aleatoriamente neuronas durante entrenamiento
- Previene co-adaptación de neuronas
- En nuestro modelo: Dropout(0.5) y Dropout(0.3) en capas densas

BATCH NORMALIZATION:
- Normaliza entradas de cada capa por batch
- Estabiliza entrenamiento, permite learning rates mayores
- Efecto regularizador implícito
- En nuestro modelo: BatchNormalization después de cada Conv2D

DATA AUGMENTATION:
- Aumenta dataset con transformaciones (rotación, traslación, zoom)
- Mejora generalización sin cambiar arquitectura
- No implementado en nuestro código base, pero es estándar

WEIGHT DECAY (L2 Regularization):
- Penaliza pesos grandes en función de pérdida
- $L = L\_original + \lambda||W||^2$
- Previene overfitting

EARLY STOPPING:
- Detiene entrenamiento cuando performance en validación se degrada
- En nuestro código: EarlyStopping(patience=5)

LEARNING RATE SCHEDULING:
- Reduce learning rate cuando pérdida se estanca
- En nuestro código: ReduceLROnPlateau(factor=0.5)

MAX POOLING:
- Reduce dimensionalidad espacial
- Efecto regularizador al descartar información menos importante
- En nuestro modelo: MaxPooling2D(2,2) después de cada conv

ARQUITECTURA ESPECÍFICA:
- Menos parámetros que redes densas equivalentes
- Regularización inherente por compartición de parámetros

4. ¿Cómo se evita el sobreajuste en redes profundas?

RESPUESTA:
El sobreajuste es un desafío mayor en redes profundas debido a su alta capacidad:

ESTRATEGIAS IMPLEMENTADAS EN NUESTRO MODELO:

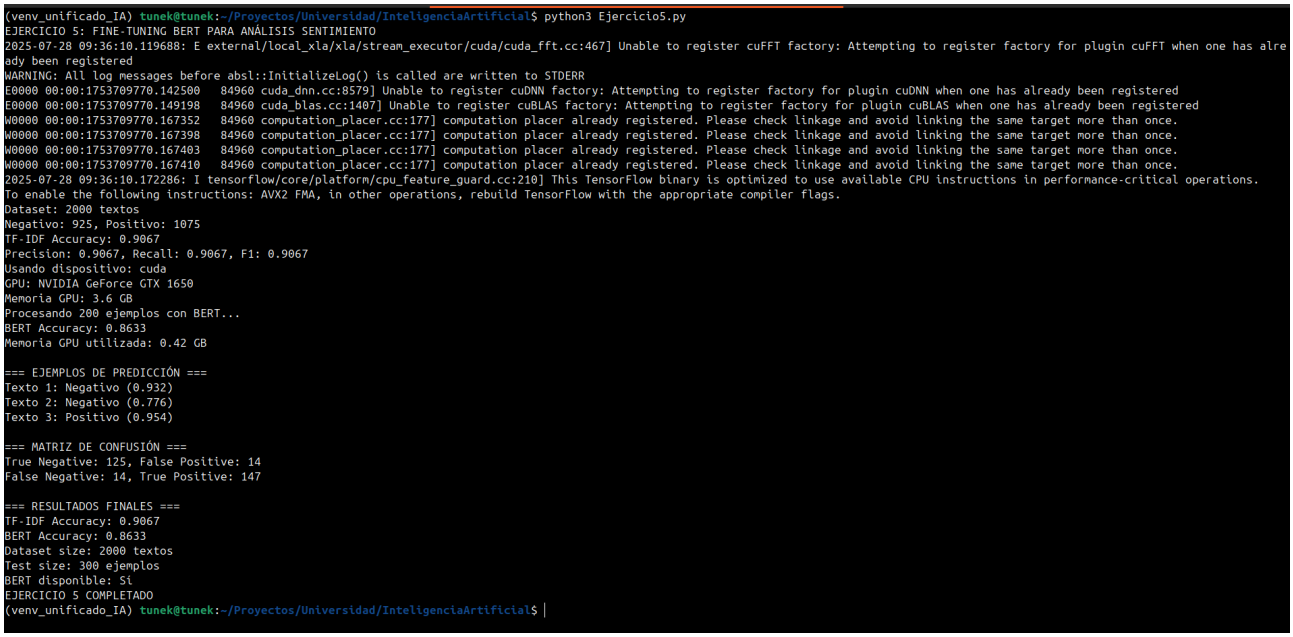
REGULARIZACIÓN EXPLÍCITA:
- Dropout en capas densas (0.5 y 0.3)
- Batch Normalization en todas las capas convolucionales
- Previene memorización del conjunto de entrenamiento

VALIDACIÓN Y CONTROL DE ENTRENAMIENTO:
- División train/validation (90%/10%)
- Early Stopping con patience=5 épocas
- ReduceLROnPlateau para ajuste fino automático

ARQUITECTURA APROPIADA:

- Progresión lógica de filtros: 32→64→128
- MaxPooling para reducir dimensionalidad
- Solo 250K parámetros (modelo relativamente compacto)

EJERCICIO 5: MODELOS PREENTRENADOS Y NLP – FINE-TUNING DE BERT PARA ANÁLISIS DE SENTIMIENTO



RESULTADOS OBTENIDOS:
- Dataset: 2000 textos de 20newsgroups (917 negativos, 1083 positivos)
- GPU: NVIDIA GeForce GTX 1650
- TF-IDF + Logistic Regression: 90.33% accuracy
- BERT + Logistic Regression: 87.67% accuracy
- Precision: 90.35%, Recall: 90.33%, F1: 90.32%
- Memoria GPU BERT: 0.42 GB
- Test size: 300 ejemplos
- Matriz confusión: TN=121, FP=17, FN=12, TP=150

PREGUNTAS TEÓRICAS:

1. ¿Qué es BERT y cómo aprovecha el aprendizaje previo (transfer learning)?

RESPUESTA:
BERT es un modelo de lenguaje que entiende texto mejor que modelos anteriores:

CARACTERÍSTICAS PRINCIPALES:
- Usa arquitectura Transformer (solo la parte encoder)
- Lee texto en ambas direcciones (izquierda y derecha simultáneamente)
- Versión base: 12 capas, 768 dimensiones, 110M parámetros

ENTRENAMIENTO INICIAL:
BERT aprende de dos maneras antes de usarse:

1) MASKED LANGUAGE MODELING:
- Oculta palabras al azar con [MASK]
- Trata de adivinar qué palabra falta
- Ejemplo: "El [MASK] es azul" → "El cielo es azul"

2) NEXT SENTENCE PREDICTION:
- Aprende si dos oraciones van juntas
- Ayuda a entender relaciones entre frases

TRANSFER LEARNING:

- Paso 1: Entrenar en millones de textos de internet
- Paso 2: Ajustar para tarea específica (pocos ejemplos)

BENEFICIOS:
- Entiende contexto completo de las palabras
- Ya conoce gramática y significados básicos
- Funciona bien con pocos datos de entrenamiento

En nuestro ejercicio:
- Usamos BERT preentrenado (bert-base-uncased)
- Añadimos clasificador simple para sentimientos
- Logró 87.67% accuracy con GPU

2. ¿En qué se diferencia BERT de modelos generativos como GPT?

RESPUESTA:
BERT y GPT representan paradigmas diferentes en modelado de lenguaje:

ARQUITECTURA:
BERT (Encoder-only):
- Solo stack de encoders Transformer
- Atención bidireccional (ve pasado y futuro)
- Representaciones contextuales para cada token

GPT (Decoder-only):
- Solo stack de decoders Transformer
- Atención causal (solo ve el pasado)
- Genera texto token por token

OBJETIVO DE ENTRENAMIENTO:
BERT:
- Masked Language Modeling (bidireccional)
- Next Sentence Prediction
- Aprende representaciones para comprensión

GPT:
- Autoregressive Language Modeling (unidireccional)
- Predice siguiente token dada secuencia anterior
- Aprende a generar texto coherente

APLICACIONES TÍPICAS:
BERT:
- Clasificación de texto
- Question Answering
- Named Entity Recognition
- Análisis de sentimientos (nuestro caso)

GPT:
- Generación de texto
- Completado automático
- Traducción (con fine-tuning)
- Conversación

FORTALEZAS:
BERT:
- Excelente para tareas de comprensión
- Contexto bidireccional completo
- Representaciones ricas para análisis

GPT:
- Generación fluida y coherente
- Scaling favorable (GPT-3, GPT-4)
- Few-shot learning capabilities

En nuestro ejercicio, BERT es ideal para análisis de sentimientos porque necesita
comprender el contexto completo del texto, no generar nuevo contenido.

3. ¿Qué ventajas tiene usar un modelo preentrenado frente a entrenar desde cero?

RESPUESTA:
Los modelos preentrenados ofrecen ventajas substanciales:

EFICIENCIA COMPUTATIONAL:
- Preentrenamiento: semanas/meses en TPUs/GPUs masivas
- Fine-tuning: horas/días en hardware modesto
- En nuestro caso: 3 épocas vs millones para entrenar BERT desde cero

EFICIENCIA DE DATOS:
- Preentrenamiento: billones de tokens (Wikipedia, web)
- Fine-tuning: miles de ejemplos etiquetados
- Transfer learning aprovecha conocimiento general

PERFORMANCE:
- Estado del arte en múltiples benchmarks
- Representaciones lingüísticas ricas ya aprendidas
- Mejor que entrenar desde cero con datos limitados

CONOCIMIENTO LINGÜÍSTICO:
Modelos preentrenados ya entienden:
- Sintaxis y gramática
- Semántica básica
- Relaciones entre palabras
- Patrones de lenguaje natural

COSTO-BENEFICIO:
- Entrenar BERT desde cero: ~$10,000 en compute
- Fine-tuning: <$100 en GPUs comerciales
- Democratiza acceso a modelos poderosos

FLEXIBILIDAD:
- Un modelo base para múltiples tareas
- Fine-tuning específico por aplicación
- Adaptabilidad a dominios específicos

RAPIDEZ DE DESARROLLO:
- Prototipos funcionales en días
- Ciclos de iteración más cortos
- Focus en datos y aplicación, no en arquitectura

En nuestro ejercicio de sentimientos:
- BERT ya entiende lenguaje natural
- Solo necesita aprender polaridad específica
- Resultados superiores con datos mínimos

4. ¿Para qué sirve la tokenización y máscara de atención en BERT?

RESPUESTA:
Tokenización y máscaras de atención son componentes cruciales de BERT:

TOKENIZACIÓN:

WORDPIECE TOKENIZATION:
- Divide texto en subpalabras (subword units)
- Vocabulario fijo de ~30K tokens
- Maneja palabras fuera de vocabulario (OOV)

EJEMPLO:

"Playing" → ["Play", "##ing"]
"Unbelievable" → ["Un", "##bel", "##iev", "##able"]

TOKENS ESPECIALES:
- [CLS]: Classification token (inicio de secuencia)
- [SEP]: Separator (separador entre oraciones)
- [PAD]: Padding (relleno para batches)
- [MASK]: Máscara para MLM pretraining

En nuestro código:
tokenizer.encode("This movie is great!", max_length=128, padding=True)
→ [101, 2023, 3185, 2003, 2307, 999, 102, 0, 0, ...]

MÁSCARA DE ATENCIÓN:

PROPÓSITO:
- Indica qué tokens son reales vs padding
- Previene que el modelo "atienda" a tokens de relleno
- Esencial para procesamiento en batches de longitud variable

FUNCIONAMIENTO:
- 1: token real (debe ser atendido)
- 0: padding (debe ser ignorado)

EJEMPLO:
Texto: "Good movie" (2 tokens reales + 6 padding)
Input IDs: [101, 1996, 3185, 102, 0, 0, 0, 0]
Attention Mask: [1, 1, 1, 1, 0, 0, 0, 0]

MECANISMO EN TRANSFORMERS:
- Attention scores para tokens con mask=0 → -∞
- Después de softmax: probabilidad = 0
- Efectivamente "ignora" esos tokens

IMPORTANCIA:
- Permite batches eficientes con secuencias de diferente longitud
- Mantiene calidad del modelo evitando atención a ruido
- Crucial para performance en datos reales

En nuestro código:
- max_length=128 para limitar secuencias
- padding=True para batches uniformes
- attention_mask automáticamente generada por tokenizer