Guía de Laboratorio - Sesión 14: Comparativa de Algoritmos Basados en Instancias

Inteligencia Artificial (100000S14F)
Ingeniería de Software

Ciclo 2 Agosto 2025

Objetivos del Laboratorio

- Implementar practica de K-NN, CBR basico y SVM Radial
- Generar visualizaciones comparativas entre algoritmos
- Analizar resultados y extraer conclusiones practicas
- Desarrollar criterio para seleccion de algoritmos segun el problema

1. Configuracion Inicial

```
# Importaciones esenciales
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import time
import warnings
warnings.filterwarnings('ignore')

# Configuracion de estilos para graficos
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)
```

2. Ejercicio 1: Preparacion de Datos y Visualizacion

```python
def cargar_y_visualizar_datos():
    """Carga datasets y genera visualizaciones exploratorias"""

    # Cargar datasets
    iris = datasets.load_iris()
    wine = datasets.load_wine()
    cancer = datasets.load_breast_cancer()

    datasets_dict = {
        'iris': (iris.data, iris.target, iris.feature_names, iris.
            target_names),
        'wine': (wine.data, wine.target, wine.feature_names, wine.
            target_names),
        'cancer': (cancer.data, cancer.target, cancer.feature_names,
            cancer.target_names)
    }

    # Crear visualizaciones para cada dataset
    fig, axes = plt.subplots(3, 3, figsize=(20, 15))

    for idx, (nombre, (X, y, features, target_names)) in enumerate(
        datasets_dict.items()):
        # DataFrame para facilitar el plotting
        df = pd.DataFrame(X, columns=features[:4])
        df['target'] = y

        # Scatter plot de las dos primeras caracteristicas
        ax = axes[idx, 0]
        for target_idx in np.unique(y):
            mask = y == target_idx
            ax.scatter(X[mask, 0], X[mask, 1], label=target_names[
                target_idx], alpha=0.7)
        ax.set_title(f'{nombre.upper()} - Caracteristicas 1 vs 2')
        ax.set_xlabel(features[0])
        ax.set_ylabel(features[1])
        ax.legend()

        # Distribucion de clases
        ax = axes[idx, 1]
        counts = pd.Series(y).value_counts()
        ax.bar([target_names[i] for i in counts.index], counts.values)
        ax.set_title(f'{nombre.upper()} - Distribucion de Clases')
        ax.tick_params(axis='x', rotation=45)

        # Heatmap de correlacion
        ax = axes[idx, 2]
        corr_matrix = df.iloc[:, :5].corr()
        sns.heatmap(corr_matrix, annot=True, ax=ax, cmap='coolwarm',
            center=0)
        ax.set_title(f'{nombre.upper()} - Correlacion')

    plt.tight_layout()
    plt.savefig('exploracion_datos.png', dpi=300, bbox_inches='tight')
    plt.show()

    return datasets_dict
```

```
# Ejecutar la funcion
datasets = cargar_y_visualizar_datos()
print("Visualizaciones exploratorias generadas")
```

3. Ejercicio 2: Implementacion de Algoritmos

```
def implementar_algoritmos(X_train, X_test, y_train, y_test,
    nombre_dataset):
    """Implementa y compara los tres algoritmos"""

    resultados = {}

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

    # 1. K-NN con diferentes valores de K
    print(f"{nombre_dataset.upper()} - Probando K-NN:")
    knn_resultados = {}
    for k in [3, 5, 7, 10]:
        inicio = time.time()
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train_scaled, y_train)
        y_pred = knn.predict(X_test_scaled)
        precision = accuracy_score(y_test, y_pred)
        tiempo = time.time() - inicio

        knn_resultados[k] = {'precision': precision, 'tiempo': tiempo}
        print(f"   K={k}: Precision={precision:.4f}, Tiempo={tiempo:.4f
            }s")

    resultados['KNN'] = knn_resultados

    # 2. SVM Radial con diferentes parametros
    print(f"{nombre_dataset.upper()} - Probando SVM Radial:")
    svm_resultados = {}
    parametros_svm = [
        {'C': 0.1, 'gamma': 'scale'},
        {'C': 1.0, 'gamma': 'scale'},
        {'C': 10.0, 'gamma': 'scale'},
        {'C': 1.0, 'gamma': 'auto'}
    ]

    for params in parametros_svm:
        inicio = time.time()
        svm = SVC(kernel='rbf', **params, random_state=42)
        svm.fit(X_train_scaled, y_train)
        y_pred = svm.predict(X_test_scaled)
        precision = accuracy_score(y_test, y_pred)
        tiempo = time.time() - inicio

        clave = f"C={params['C']},gamma={params['gamma']}"
        svm_resultados[clave] = {'precision': precision, 'tiempo':
            tiempo}
```

```
        print(f"   {clave}: Precision={precision:.4f}, Tiempo={tiempo
            :.4f}s")

    resultados['SVM'] = svm_resultados

    # 3. CBR Basico
    print(f"{nombre_dataset.upper()} - Probando CBR Basico:")
    inicio = time.time()
    cbr = KNeighborsClassifier(n_neighbors=3)
    cbr.fit(X_train_scaled, y_train)
    y_pred = cbr.predict(X_test_scaled)
    precision = accuracy_score(y_test, y_pred)
    tiempo = time.time() - inicio

    resultados['CBR'] = {'precision': precision, 'tiempo': tiempo}
    print(f"   Precision={precision:.4f}, Tiempo={tiempo:.4f}s")

    return resultados

def ejecutar_experimentos_completos(datasets_dict, test_size=0.3):
    """Ejecuta experimentos para todos los datasets"""

    resultados_completos = {}

    for nombre, (X, y, features, target_names) in datasets_dict.items()
        :
        print(f"INICIANDO EXPERIMENTO: {nombre.upper()}")

        # Dividir datos
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=42, stratify=y
        )

        # Ejecutar algoritmos
        resultados = implementar_algoritmos(X_train, X_test, y_train,
            y_test, nombre)
        resultados_completos[nombre] = resultados

        # Generar matriz de confusion
        generar_matrices_confusion(X_train, X_test, y_train, y_test,
            nombre, target_names)

    return resultados_completos

def generar_matrices_confusion(X_train, X_test, y_train, y_test,
    nombre_dataset, target_names):
    """Genera matrices de confusion para cada algoritmo"""

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Modelos a comparar
    modelos = {
        'KNN (K=5)': KNeighborsClassifier(n_neighbors=5),
        'SVM (C=1.0)': SVC(kernel='rbf', C=1.0, gamma='scale',
            random_state=42),
        'CBR (K=3)': KNeighborsClassifier(n_neighbors=3)
```

```
    }

    fig, axes = plt.subplots(1, 3, figsize=(18, 5))

    for idx, (nombre_modelo, modelo) in enumerate(modelos.items()):
        modelo.fit(X_train_scaled, y_train)
        y_pred = modelo.predict(X_test_scaled)

        # Matriz de confusion
        cm = confusion_matrix(y_test, y_pred)
        sns.heatmap(cm, annot=True, fmt='d', ax=axes[idx],
                    xticklabels=target_names, yticklabels=target_names,
                    cmap='Blues')
        axes[idx].set_title(f'{nombre_dataset.upper()} - {nombre_modelo
            }')
        axes[idx].set_xlabel('Predicho')
        axes[idx].set_ylabel('Real')

    plt.tight_layout()
    plt.savefig(f'matrices_confusion_{nombre_dataset}.png', dpi=300,
        bbox_inches='tight')
    plt.show()

# Ejecutar experimentos
resultados = ejecutar_experimentos_completos(datasets)
print("Todos los experimentos completados")
```

4. Ejercicio 3: Analisis Visual de Resultados

```
def generar_graficas_comparativas(resultados):
    """Genera multiples graficas para comparar resultados"""

    # Preparar datos para graficas
    datos_graficas = []
    for dataset, algoritmos in resultados.items():
        for algo, valores in algoritmos.items():
            if algo in ['KNN', 'SVM']:
                for config, metricas in valores.items():
                    datos_graficas.append({
                        'Dataset': dataset,
                        'Algoritmo': algo,
                        'Configuracion': config,
                        'Precision': metricas['precision'],
                        'Tiempo': metricas['tiempo']
                    })
            else:  # CBR
                datos_graficas.append({
                    'Dataset': dataset,
                    'Algoritmo': algo,
                    'Configuracion': 'default',
                    'Precision': valores['precision'],
                    'Tiempo': valores['tiempo']
                })

    df = pd.DataFrame(datos_graficas)
```

```
# 1. Grafica de precision por algoritmo y dataset
plt.figure(figsize=(14, 10))

# Subplot 1: Comparacion de precision
plt.subplot(2, 2, 1)
precision_pivot = df.pivot_table(values='Precision', index='Dataset
    ', columns='Algoritmo', aggfunc='max')
precision_pivot.plot(kind='bar', ax=plt.gca())
plt.title('Comparacion de Precision Maxima por Algoritmo')
plt.ylabel('Precision')
plt.xticks(rotation=45)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

# Subplot 2: Comparacion de tiempos de ejecucion
plt.subplot(2, 2, 2)
tiempo_pivot = df.pivot_table(values='Tiempo', index='Dataset',
    columns='Algoritmo', aggfunc='mean')
tiempo_pivot.plot(kind='bar', ax=plt.gca())
plt.title('Tiempo Promedio de Ejecucion por Algoritmo')
plt.ylabel('Tiempo (segundos)')
plt.xticks(rotation=45)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

# Subplot 3: Heatmap de precisiones
plt.subplot(2, 2, 3)
heatmap_data = df.groupby(['Dataset', 'Algoritmo'])['Precision'].
    max().unstack()
sns.heatmap(heatmap_data, annot=True, cmap='YlGnBu', fmt='.3f', ax=
    plt.gca())
plt.title('Heatmap de Precision por Algoritmo y Dataset')

# Subplot 4: Grafico de dispersion Precision vs Tiempo
plt.subplot(2, 2, 4)
for algoritmo in df['Algoritmo'].unique():
    mask = df['Algoritmo'] == algoritmo
    plt.scatter(df[mask]['Tiempo'], df[mask]['Precision'], label=
        algoritmo, alpha=0.6, s=100)
plt.xlabel('Tiempo (segundos)')
plt.ylabel('Precision')
plt.title('Relacion Precision vs Tiempo de Ejecucion')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('comparativa_general.png', dpi=300, bbox_inches='tight'
    )
plt.show()

# 2. Graficas especificas por algoritmo
for algoritmo in ['KNN', 'SVM']:
    if algoritmo in df['Algoritmo'].values:
        plt.figure(figsize=(12, 8))
        algo_data = df[df['Algoritmo'] == algoritmo]

        # Para KNN: Efecto del parametro K
        if algoritmo == 'KNN':
```

```
        algo_data['K'] = algo_data['Configuracion'].str.extract
            ('(\\d+)').astype(float)
        for dataset in algo_data['Dataset'].unique():
            mask = algo_data['Dataset'] == dataset
            plt.plot(algo_data[mask]['K'], algo_data[mask]['
                Precision'],
                     'o-', label=dataset, markersize=8)

        plt.xlabel('Valor de K')
        plt.ylabel('Precision')
        plt.title('Efecto del Parametro K en K-NN')
        plt.legend()
        plt.grid(True, alpha=0.3)
        plt.savefig('efecto_k_knn.png', dpi=300, bbox_inches='
            tight')
        plt.show()

    # Para SVM: Comparacion de parametros C y gamma
    elif algoritmo == 'SVM':
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

        # Efecto de C
        c_data = algo_data[algo_data['Configuracion'].str.
            contains('gamma=scale')]
        for dataset in c_data['Dataset'].unique():
            mask = c_data['Dataset'] == dataset
            c_values = c_data[mask]['Configuracion'].str.
                extract('C=([\\d.]+)').astype(float)
            ax1.plot(c_values[0], c_data[mask]['Precision'], 'o
                -', label=dataset, markersize=8)

        ax1.set_xlabel('Valor de C')
        ax1.set_ylabel('Precision')
        ax1.set_title('Efecto del Parametro C en SVM (gamma=
            scale)')
        ax1.legend()
        ax1.grid(True, alpha=0.3)
        ax1.set_xscale('log')

        # Efecto de gamma
        gamma_data = algo_data[algo_data['Configuracion'].str.
            contains('C=1.0')]
        gamma_precision = gamma_data.groupby(['Dataset', '
            Configuracion'])['Precision'].mean().unstack()
        gamma_precision.T.plot(ax=ax2, marker='o')
        ax2.set_xlabel('Configuracion de Gamma')
        ax2.set_ylabel('Precision')
        ax2.set_title('Efecto del Parametro Gamma en SVM (C
            =1.0)')
        ax2.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        ax2.grid(True, alpha=0.3)
        ax2.tick_params(axis='x', rotation=45)

        plt.tight_layout()
        plt.savefig('efecto_parametros_svm.png', dpi=300,
            bbox_inches='tight')
        plt.show()
```

```
    return df

# Generar todas las graficas
df_resultados = generar_graficas_comparativas(resultados)
print("Graficas comparativas generadas")
```

5. Ejercicio 4: Analisis de Resultados y Conclusiones

```
def analizar_resultados_detallados(df_resultados):
    """Realiza analisis estadistico y genera conclusiones"""

    print("ANALISIS DETALLADO DE RESULTADOS")

    # Estadisticas descriptivas
    print("\n1. ESTADISTICAS DESCRIPTIVAS POR ALGORITMO:")
    stats = df_resultados.groupby('Algoritmo')['Precision'].describe()
    print(stats)

    # Mejor algoritmo por dataset
    print("\n2. MEJOR ALGORITMO POR DATASET:")
    mejor_por_dataset = df_resultados.loc[df_resultados.groupby('
        Dataset')['Precision'].idxmax()]
    print(mejor_por_dataset[['Dataset', 'Algoritmo', 'Configuracion', '
        Precision', 'Tiempo']])

    # Analisis de trade-off precision/tiempo
    print("\n3. ANALISIS DE TRADE-OFF PRECISION/TIEMPO:")
    for dataset in df_resultados['Dataset'].unique():
        dataset_data = df_resultados[df_resultados['Dataset'] ==
            dataset]
        mejor_precision = dataset_data['Precision'].max()
        mejor_tiempo = dataset_data.loc[dataset_data['Precision'].
            idxmax(), 'Tiempo']

        print(f"{dataset.upper()}:")
        print(f"   Mejor precision: {mejor_precision:.4f}")
        print(f"   Tiempo del mejor: {mejor_tiempo:.4f}s")

        # Encontrar el mas rapido con precision similar
        precision_umbral = mejor_precision * 0.95
        rapidos = dataset_data[dataset_data['Precision'] >=
            precision_umbral]
        if not rapidos.empty:
            mas_rapido = rapidos.loc[rapidos['Tiempo'].idxmin()]
            print(f"   Mas rapido con buena precision: {mas_rapido['
                Algoritmo']}")
            print(f"   Precision: {mas_rapido['Precision']:.4f}, Tiempo
                : {mas_rapido['Tiempo']:.4f}s")

    # Generar recomendaciones
    print("\nRECOMENDACIONES PRACTICAS")

    recomendaciones = {
        'KNN': {
            'escenarios': ['Datos balanceados', 'Necesidad de
                interpretabilidad', 'Recursos limitados'],
```

```
            'ventajas': ['Simple de implementar', 'No requiere
                entrenamiento', 'Facil de entender'],
            'desventajas': ['Sensible a la escala', 'Lento con muchos
                datos', 'Sensible a caracteristicas irrelevantes']
        },
        'SVM': {
            'escenarios': ['Problemas complejos no lineales', 'Alta
                dimensionalidad', 'Maxima precision requerida'],
            'ventajas': ['Efectivo en espacios de alta dimension', '
                Robusto contra overfitting', 'Maneja no linealidad'],
            'desventajas': ['Lento en entrenamiento', 'Dificil
                interpretacion', 'Sensible a parametros']
        },
        'CBR': {
            'escenarios': ['Dominios basados en experiencia', 'Sistemas
                 de recomendacion', 'Problemas con casos similares'],
            'ventajas': ['Explicabilidad', 'Aprendizaje incremental', '
                Adaptabilidad'],
            'desventajas': ['Depende de la base de casos', 'Costoso en
                recuperacion', 'Dificultad en adaptacion']
        }
    }

    for algo, info in recomendaciones.items():
        print(f"{algo}:")
        print(f"   Escenarios ideales: {', '.join(info['escenarios'])}"
            )
        print(f"   Ventajas: {', '.join(info['ventajas'])}")
        print(f"   Consideraciones: {', '.join(info['desventajas'])}")

    # Grafica final de resumen
    plt.figure(figsize=(10, 6))
    summary_data = df_resultados.groupby('Algoritmo').agg({
        'Precision': ['mean', 'std'],
        'Tiempo': ['mean', 'std']
    }).round(4)

    # Grafico de barras para precision media
    ax = summary_data['Precision']['mean'].plot(kind='bar', yerr=
        summary_data['Precision']['std'],
                                                capsize=5, color='
                                                    lightblue')
    plt.title('Precision Media por Algoritmo (con desviacion estandar)'
        )
    plt.ylabel('Precision Media')
    plt.xticks(rotation=45)
    plt.grid(True, alpha=0.3)

    # Anadir valores en las barras
    for i, v in enumerate(summary_data['Precision']['mean']):
        ax.text(i, v + 0.01, f'{v:.3f}', ha='center', va='bottom')

    plt.tight_layout()
    plt.savefig('resumen_final.png', dpi=300, bbox_inches='tight')
    plt.show()

    return summary_data
```

```
# Ejecutar analisis final
resumen_final = analizar_resultados_detallados(df_resultados)
print("Analisis completado")
```

6. Ejercicio 5: Entrega y Reflexion

```
def generar_reporte_final():
    """Genera un reporte final con los hallazgos principales"""

    print("CONCLUSIONES FINALES DEL LABORATORIO")

    conclusiones = """
    HALLazGOS PRINCIPALES:

    1. K-NN demostro ser efectivo en datasets peque os y balanceados
       como Iris,
       pero su rendimiento decae con datasets mas complejos.

    2. SVM Radial mostro la mayor precision general, especialmente en
       problemas
       complejos como el dataset de cancer, pero requiere mas tiempo de
           computo.

    3. CBR (en su version simplificada) ofrece un buen balance entre
       interpretabilidad
       y rendimiento, ideal cuando la explicabilidad es importante.

    4. La escalabilidad es un factor critico: K-NN se vuelve
       prohibitivo con
       grandes volumenes de datos, mientras que SVM maneja mejor la
          dimensionalidad.

    5. La seleccion de parametros es crucial: K optimo en K-NN y
       combinacion
       C-gamma en SVM pueden mejorar significativamente el rendimiento.

    RECOMENDACIONES PRACTICAS:

    - Para problemas simples e interpretables: USAR K-NN
    - Para maxima precision en problemas complejos: USAR SVM Radial
    - Cuando la explicabilidad es prioridad: USAR CBR
    - Siempre normalizar los datos antes de aplicar estos algoritmos
    - Realizar validacion cruzada para optimizar hiperparametros

    APRENDIZAJES CLAVE:

    - No existe el mejor algoritmo universal - depende del problema
    - El trade-off entre precision y tiempo de computo es inevitable
    - La preparacion de datos es tan importante como la seleccion del
      algoritmo
    - La experimentacion sistematica es esencial para tomar decisiones
      informadas
    """

    print(conclusiones)
```

```
    # Guardar resultados en CSV para analisis posterior
    df_resultados.to_csv('resultados_comparativa.csv', index=False)
    print("Resultados guardados en 'resultados_comparativa.csv'")

# Generar reporte final
generar_reporte_final()
```

Instrucciones de Entrega

Elementos a Entregar:

1. Codigo Python completo ejecutable
2. Todas las graficas generadas (minimo 5 diferentes)
3. Archivo CSV con resultados numericos
4. Breve informe (1 pagina) con conclusiones personales

Criterios de Evaluacion:

- Correcta ejecucion de todos los algoritmos (30 %)
- Calidad y variedad de visualizaciones (30 %)
- Analisis critico de resultados (25 %)
- Conclusiones y reflexiones (15 %)

Tiempo Estimado:

- 0-15 min: Configuracion y Ejercicio 1
- 15-45 min: Ejercicio 2 (implementacion)
- 45-75 min: Ejercicios 3-4 (analisis y graficas)
- 75-90 min: Ejercicio 5 (conclusiones y entrega)