Universidad Autónoma de Yucatán

Facultad de Matemáticas

Licenciado en Ciencias de la Computación

Inteligencia Artificial

Proyecto aprendizaje supervisado

Profesor / Alumnos:

Alejandro Pasos Ruiz / Julian

Emiliano Ortiz Rivero / Angel Mariel

Osalde Salazar & Miguel Omar Solis

Rodriguez

Realizar experimentos usando varios algoritmos de machine learning para los datos y realizar ajuste de parametros y validacion cruzzada para dar sustento a los resultados

Mérida, Yucatán, Enero - Mayo 2025

Contents

1 Introducción

La enfermedad de Parkinson es un trastorno cerebral que deteriora las células nerviosas encargadas del control de los movimientos y aunque no hay medicamentos que frenen su avance, existen tratamientos para controlar los síntomas en cada etapa, por lo que es importante un diagnóstico preciso y un manejo adecuado de los medicamentos para mejorar la calidad de vida de los pacientes. El Parkinson, al ser el segundo trastorno cerebral más común después del Alzheimer y tener como principal factor de riesgo la edad, representa un problema para la salud pública, especialmente en sociedades occidentales con poblaciones con un porcentaje considerable de personas mayores.

Es por ello que el reporte tiene como objetivo analizar un conjunto de datos relacionados con la enfermedad de Parkinson para predecir el estado de salud de los pacientes a partir de diversas características, utilizando modelos de aprendizaje automático que permiten identificar patrones en los datos y hacer estimaciones sobre su condición. El documento está organizado en varias secciones, donde primero se explica el problema que se aborda, luego se describe el conjunto de datos con el que se trabaja, se detallan los experimentos realizados, se presentan los resultados obtenidos y se escriben algunas conclusiones sobre lo que se ha aprendido.

2 Descripción del Problema

La enfermedad de Parkinson es un trastorno complejo que afecta el movimiento, causando temblores, rigidez y dificultades para caminar, lo que presenta un problema para los médicos, ya que los síntomas pueden variar mucho entre las personas y son similares a los de otras enfermedades. Esto hace que el diagnóstico temprano sea difícil, lo que complica el tratamiento y el manejo de la enfermedad. Además, como los síntomas suelen empeorar con el tiempo, es importante poder detectar la enfermedad en sus primeras etapas para intervenir lo más pronto posible. Por esta razón, se pretende utilizar técnicas de aprendizaje automático para analizar datos relacionados con la enfermedad y predecir el estado de salud de los pacientes, lo que podría mejorar el diagnóstico y permitir un tratamiento más efectivo desde las primeras etapas de la enfermedad.

3 Descripción de los Datos

El conjunto de datos utilizado en este estudio proviene de un dataset en el que se incluyen varias características relacionadas con la enfermedad de Parkinson, como medidas de voz, temblores, y otras variables clínicas. El conjunto de datos se limpió y preprocesó para eliminar valores nulos y manejar valores atípicos. Además, se normalizaron las características para mejorar el rendimiento de los modelos de aprendizaje automático.

3.1 Limpieza del Dataset

A continuación, se muestra el código utilizado para limpiar y preprocesar el dataset:

```
# Importar librer as necesarias
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import joblib

# Cargar el dataset
df = pd.read_csv("updated_dataset.csv")
```

```
9
10 # Eliminar la columna 'name' ya que no aporta informaci n relevante
11 df.drop(columns=['name'], inplace=True)
12
13 # Verificar valores nulos y eliminar filas si es necesario
14 df.dropna(inplace=True)
15
16 # Identificar y manejar outliers con percentiles
17 for col in df.columns:
18     if col != "status":  # No modificar la variable objetivo
19         q1 = df[col].quantile(0.25)
20         q3 = df[col].quantile(0.75)
21         iqr = q3 - q1
22         lower_bound = q1 - 1.5 * iqr
23         upper_bound = q3 + 1.5 * iqr
24         df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
25
26 # Separar caracter sticas y variable objetivo
27 X = df.drop(columns=["status"])
28 y = df["status"]
29
30 # Normalizaci n de las caracter sticas
31 scaler = StandardScaler()
32 X_scaled = scaler.fit_transform(X)
33
34 # Guardar datos preprocesados y scaler
35 joblib.dump((X_scaled, y), "datos_limpiados.pkl")
36 joblib.dump(scaler, "scaler.pkl")
37
38 print("Preprocesamiento completo. Datos limpios guardados en 'datos_limpiados.pkl'")
```

4 Experimentos

Se realizaron varios experimentos utilizando diferentes modelos de aprendizaje automático, incluyendo Regresión Logística, Random Forest, MLP (Perceptrón Multicapa) y XGBoost. Los datos se dividieron en conjuntos de entrenamiento y prueba, y se utilizó validación cruzada para evaluar el rendimiento de los modelos. Además, se calcularon métricas como precisión, recall y accuracy para comparar el rendimiento de los modelos.

4.1 Modelos Utilizados

Regresión Logística: Un modelo lineal utilizado para problemas de clasificación binaria, que se basa en la relación lineal entre las características de entrada y la salida. Este modelo es adecuado para problemas en los que la relación entre las variables es lineal o casi lineal.

```
1 # Cargar los datos preprocesados
2 X_scaled, y = joblib.load("datos_limpiados.pkl")
3
4 # Divisi n en datos de entrenamiento y prueba
5 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
      random_state=42, stratify=y)
6
7 # Entrenar el modelo de Regresi n Log stica
8 log_reg = LogisticRegression(max_iter=500, random_state=42)
9 log_reg.fit(X_train, y_train)
10
11 # Predicci n y evaluaci n
12 y_pred = log_reg.predict(X_test)
13 accuracy = accuracy_score(y_test, y_pred)
```

```
report = classification_report(y_test, y_pred)

print("Accuracy de Regresi n Log stica:", accuracy)
print("Reporte de clasificaci n:\n", report)

# Guardar el modelo
joblib.dump(log_reg, "logistic_regression_model.pkl")
```

```
Accuracy de Regresión Logística: 0.5089820359281437
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.41      0.24      0.31        74
           1       0.54      0.72      0.62        93

    accuracy                           0.51       167
   macro avg       0.48      0.48      0.46       167
weighted avg       0.48      0.51      0.48       167

['logistic_regression_model.pkl']
```

Random Forest: Un método de ensamble que utiliza múltiples árboles de decisión para mejorar la precisión y reducir la variabilidad de los resultados. Cada árbol de decisión se entrena en un subconjunto de los datos y se combina con los resultados de los demás árboles para obtener un resultado final.

```
# Cargar los datos preprocesados
X_scaled, y = joblib.load("datos_limpiados.pkl")

# Divisi n en datos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
    random_state=42, stratify=y)

# Entrenar el modelo Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predicci n y evaluaci n
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy de Random Forest:", accuracy)
print("Reporte de clasificaci n:\n", report)

# Guardar el modelo
joblib.dump(rf_model, "random_forest_model.pkl")
```

```
Accuracy de Random Forest: 0.5808383233532934
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.53      0.43      0.48        74
           1       0.61      0.70      0.65        93

    accuracy                           0.58       167
   macro avg       0.57      0.57      0.56       167
weighted avg       0.57      0.58      0.57       167
```

```
['random_forest_model.pkl']
```

MLP (Perceptrón Multicapa): Una red neuronal con múltiples capas ocultas, que permite aprender representaciones complejas de los datos. Cada capa de la red neuronal procesa la información de la capa anterior y la envía a la capa siguiente, lo que permite aprender patrones y relaciones complejas en los datos.

```python
# Cargar los datos preprocesados
X_scaled, y = joblib.load("datos_limpiados.pkl")

# Divisi n en datos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
    random_state=42, stratify=y)

# Entrenar el modelo de Red Neuronal
mlp_model = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=500, random_state=42)
mlp_model.fit(X_train, y_train)

# Predicci n y evaluaci n
y_pred = mlp_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy de Red Neuronal:", accuracy)
print("Reporte de clasificaci n:\n", report)

# Guardar el modelo
joblib.dump(mlp_model, "mlp_model.pkl")
```

```
Accuracy de Red Neuronal: 0.5389221556886228
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.48      0.47      0.48        74
           1       0.59      0.59      0.59        93

    accuracy                           0.54       167
   macro avg       0.53      0.53      0.53       167
weighted avg       0.54      0.54      0.54       167

['mlp_model.pkl']
```

XGBoost: Un algoritmo de boosting que utiliza árboles de decisión para mejorar la precisión y reducir la variabilidad de los resultados. XGBoost es un algoritmo de aprendizaje automático que se basa en la técnica de boosting, que consiste en combinar los resultados de múltiples modelos para obtener un resultado final más preciso.

```python
# Cargar los datos preprocesados
X_scaled, y = joblib.load("datos_limpiados.pkl")

# Divisi n en datos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
    random_state=42, stratify=y)

# Entrenar el modelo XGBoost
xgb_model = XGBClassifier(eval_metric='mlogloss', random_state=42)
xgb_model.fit(X_train, y_train)

# Predicci n y evaluaci n
```

```
y_pred = xgb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy de XGBoost:", accuracy)
print("Reporte de clasificaci n:\n", report)

# Guardar el modelo
joblib.dump(xgb_model, "xgboost_model.pkl")
```

```
Accuracy de XGBoost: 0.6407185628742516
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.60      0.58      0.59        74
           1       0.67      0.69      0.68        93

    accuracy                           0.64       167
   macro avg       0.64      0.63      0.63       167
weighted avg       0.64      0.64      0.64       167

['xgboost_model.pkl']
```

4.2 Carga de datos de validación

```
# Cargar los datos preprocesados
X_scaled, y = joblib.load("datos_limpiados.pkl")

# Cargar modelos
models = {
    "Logistic Regression": joblib.load("logistic_regression_model.pkl"),
    "Random Forest": joblib.load("random_forest_model.pkl"),
    "MLP Neural Network": joblib.load("mlp_model.pkl"),
    "XGBoost": joblib.load("xgboost_model.pkl")
}
```

```
Accuracy de Random Forest: 0.5808383233532934
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.53      0.43      0.48        74
           1       0.61      0.70      0.65        93

    accuracy                           0.58       167
   macro avg       0.57      0.57      0.56       167
weighted avg       0.57      0.58      0.57       167

['random_forest_model.pkl']
```

4.3 Evaluación con Validación cruzada, Precisión y Recall

```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import make_scorer, precision_score, recall_score

```

```
# Diccionarios para almacenar resultados
cv_results = {}
precision_results = {}
recall_results = {}

# Evaluaci n con validaci n cruzada (5-fold) para cada m trica
for name, model in models.items():
    # Calcular Accuracy
    accuracy_scores = cross_val_score(model, X_scaled, y, cv=5, scoring='accuracy')
    cv_results[name] = accuracy_scores  # Se guarda directamente la lista de valores

    # Calcular Precision (Weighted)
    precision_scores = cross_val_score(
        model, X_scaled, y, cv=5,
        scoring=make_scorer(precision_score, average='weighted')
    )
    precision_results[name] = precision_scores

    # Calcular Recall (Weighted)
    recall_scores = cross_val_score(
        model, X_scaled, y, cv=5,
        scoring=make_scorer(recall_score, average='weighted')
    )
    recall_results[name] = recall_scores
```

5 Resultados

5.1 Gráficas de Resultados

A continuación, se muestran las gráficas de los resultados de la validación cruzada y las métricas de precisión y recall.

5.2 Evaluación con Validación Cruzada, Precisión y Recall

```
plt.figure(figsize=(10, 5))
sns.boxplot(data=cv_results.values())
plt.ylabel("Accuracy")
plt.title("Comparaci n de modelos con validaci n cruzada (5-fold)")
plt.xticks(ticks=range(len(models)), labels=models.keys(), rotation=15)
plt.show()
```

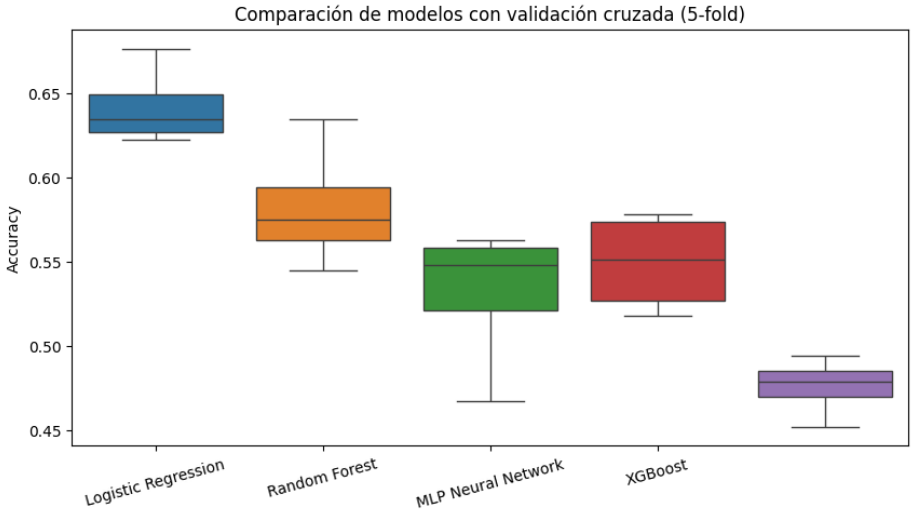

Figure 1: Comparación de modelos con validación cruzada (5-fold)

5.3 Evaluación con Validación Cruzada promediada

```
# Graficar resultados de validaci n cruzada
plt.figure(figsize=(10, 5))
model_names = list(cv_results.keys())  # Los nombres de los modelos (eje X)
accuracy_values = [np.mean(cv_results[name]) for name in model_names]  # Promedio de accuracy por modelo
bars = plt.bar(model_names, accuracy_values, color=['blue', 'green', 'red', 'purple'])
plt.ylabel("Accuracy")
plt.ylim(0,1)
plt.title("Comparaci n de modelos con validaci n cruzada (5-fold) promediados")

# Agregar los valores de accuracy sobre las barras
for bar in bars:
    yval = bar.get_height()  # Obtener la altura de cada barra (valor de accuracy)
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.02, round(yval, 2), ha='center', va='bottom')

plt.show()
```

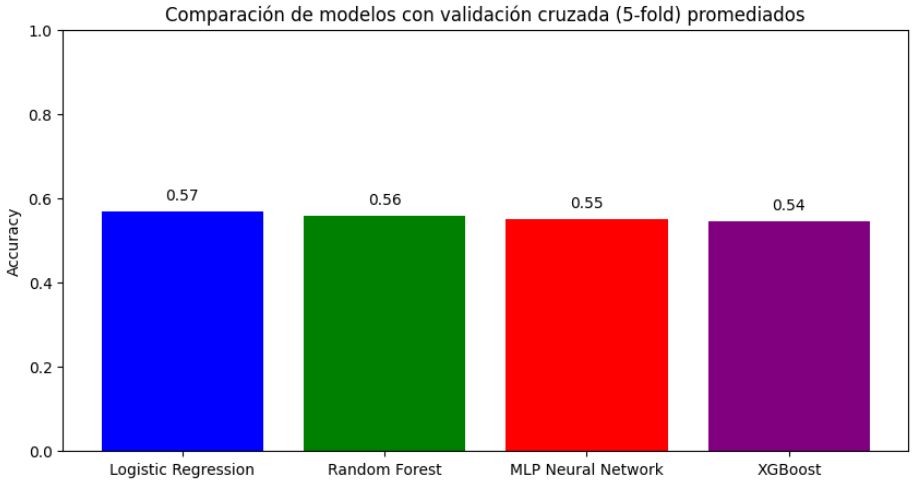

Figure 2: Comparación de modelos con validación cruzada (5-fold) promediados

5.4 Comparación de los modelos en Precisión

```
plt.figure(figsize=(10, 5))
sns.boxplot(data=precision_results.values())
plt.ylabel("Precision (Weighted)")
plt.title("Comparaci n de modelos - Precisi n")
plt.xticks(ticks=range(len(models)), labels=models.keys(), rotation=15)
plt.show()
```

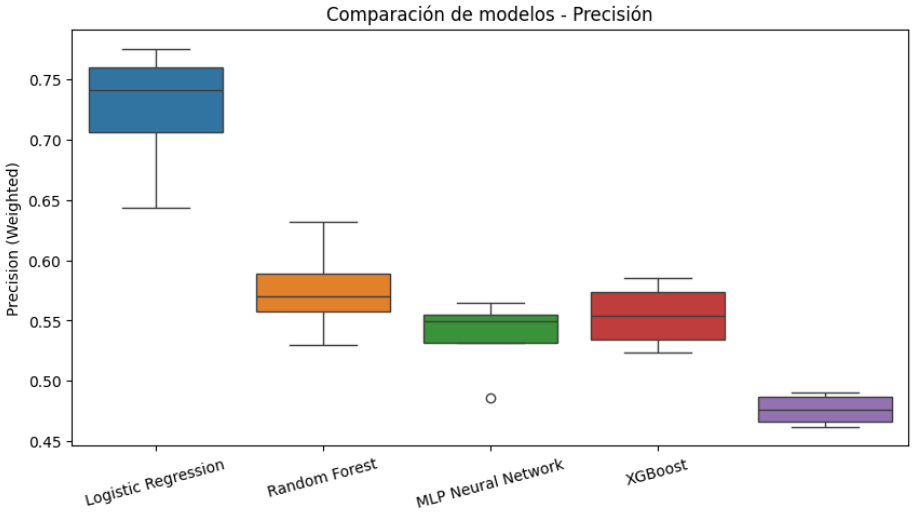

Figure 3: Comparación de modelos con precisión

5.5 Comparación de los modelos en Recall

```
plt.figure(figsize=(10, 5))
sns.boxplot(data=recall_results.values())
plt.ylabel("Recall (Weighted)")
plt.title("Comparaci n de modelos - Recall")
```

```
plt.xticks(range(len(models)), models.keys(), rotation=15)
plt.show()
```

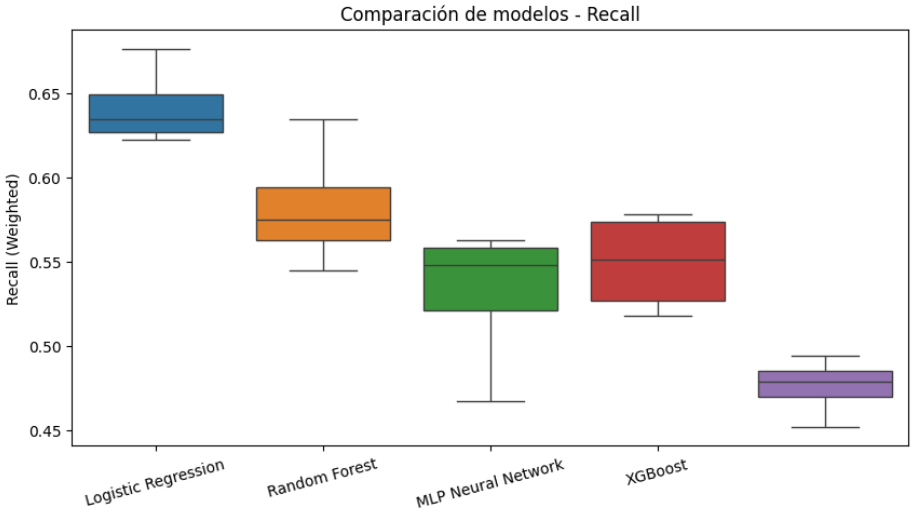

Figure 4: Comparación de modelos con recall

5.6 Comparaciones

En la siguiente tabla se presentan los resultados de las pruebas estadísticas entre los diferentes modelos.

```
# Pruebas estad sticas
stat_tests = {}
model_names = list(models.keys())
for i in range(len(model_names)):
    for j in range(i + 1, len(model_names)):
        model1, model2 = model_names[i], model_names[j]
        t_stat, p_ttest = ttest_rel(cv_results[model1], cv_results[model2])
        w_stat, p_wilcoxon = wilcoxon(cv_results[model1], cv_results[model2])
        stat_tests[f"{model1} vs {model2}"] = (p_ttest, p_wilcoxon)

# Mostrar pruebas estad sticas
print("\nResultados de pruebas estad sticas (p-values):")
for comparison, (p_ttest, p_wilcoxon) in stat_tests.items():
    print(f"{comparison}: t-test p={p_ttest:.5f}, Wilcoxon p={p_wilcoxon:.5f}")
```

Modelos Comparados	t-test	Wilcoxon
Logistic Regression vs Random Forest	0.70218	0.62500
Logistic Regression vs MLP Neural Network	0.31939	0.43750
Logistic Regression vs XGBoost	0.20003	0.18750
Random Forest vs MLP Neural Network	0.84336	0.81250
Random Forest vs XGBoost	0.58293	0.62500
MLP Neural Network vs XGBoost	0.45835	0.62500

6 Conclusiones

En este estudio, se aplicaron varios modelos de aprendizaje automático para predecir el estado de salud de pacientes con Parkinson. Los resultados indican que el modelo XGBoost es el más efectivo para esta tarea, alcanzando un accuracy del 64%. Sin embargo, es importante destacar que todos los modelos tuvieron un rendimiento relativamente bajo, lo que sugiere que podría ser necesario explorar otras técnicas o mejorar el preprocesamiento de los datos para obtener mejores resultados.

En futuros trabajos, se podría considerar la inclusión de más características o la aplicación de técnicas de selección de características para mejorar el rendimiento de los modelos. Además, sería interesante explorar el uso de técnicas de aprendizaje profundo para ver si se pueden obtener mejores resultados.