APLICACIÓN WEB DE REGRESIÓN LOGÍSTICA

Predicción de Comportamiento de Compra mediante Machine Learning

INFORMACIÓN DEL PROYECTO

Institución: Universidad Tecnológica de Tijuana
Materia: Extracción de conocimientos en Bases De Datos. **Proyecto:** Sistema Web de Predicción con Regresión Logística
Fecha: Noviembre 2025

ÍNDICE

RESUMEN EJECUTIVO

Este proyecto presenta el desarrollo de una aplicación web completa que implementa un modelo de **Regresión Logística** para predecir el comportamiento de compra de usuarios basándose en sus características demográficas (edad y salario estimado). La aplicación incluye:

- **Modelo de Machine Learning** entrenado con 400 registros
- **Interfaz Web Interactiva** desarrollada con Flask
- **Dashboard de Métricas** con visualizaciones en tiempo real
- **Sistema de Predicciones** instantáneas
- **Diseño Responsivo** con paleta de colores azul grisáceo

Métricas del Modelo:

- Accuracy: ~86%
- Precision: ~86%
- Recall: ~70%
- AUC Score: ~0.85

INTRODUCCIÓN

1.1 Contexto del Problema

En el ámbito del marketing digital y el comercio electrónico, es fundamental identificar clientes potenciales que tienen mayor probabilidad de realizar una compra. Este proyecto aborda esta necesidad mediante la implementación de técnicas de Machine Learning.

1.2 Justificación

La capacidad de predecir el comportamiento de compra permite a las empresas:

- Optimizar campañas de marketing
- Reducir costos de adquisición de clientes
- Aumentar tasas de conversión
- Segmentar audiencias de manera efectiva

1.3 Alcance

El proyecto incluye:

1. Análisis exploratorio del dataset UserData.csv
2. Desarrollo y entrenamiento del modelo de regresión logística
3. Creación de aplicación web con Flask
4. Implementación de interfaz de usuario responsiva
5. Sistema de visualización de métricas
6. API REST para predicciones

MARCO TEÓRICO

3.1 Regresión Logística

La **regresión logística** es una técnica de Machine Learning supervisado utilizada para problemas de clasificación binaria. A diferencia de la regresión lineal, utiliza la función sigmoide para mapear valores a probabilidades entre 0 y 1.

Función Logística (Sigmoide):

```
P(y=1|x) = 1 / (1 + e^(-z))
donde z = β₀ + β₁x₁ + β₂x₂ + ... + βₙxₙ
```

Características:

- Predice probabilidades de pertenencia a una clase
- Utiliza máxima verosimilitud para estimar parámetros
- Asume linealidad en el logit
- Apropiada para variables predictoras continuas o categóricas

3.2 Métricas de Evaluación

3.2.1 Accuracy (Exactitud)

Proporción de predicciones correctas sobre el total de predicciones.

```
Accuracy = (TP + TN) / (TP + TN + FP + FN)
```

3.2.2 Precision (Precisión)

Proporción de predicciones positivas correctas.

```
Precision = TP / (TP + FP)
```

3.2.3 Recall (Sensibilidad)

Proporción de casos positivos correctamente identificados.

```
Recall = TP / (TP + FN)
```

3.2.4 F1-Score

Media armónica de precision y recall.

```
F1 = 2 × (Precision × Recall) / (Precision + Recall)
```

3.2.5 AUC (Area Under Curve)

Área bajo la curva ROC, mide la capacidad del modelo para distinguir entre clases.

3.3 Tecnologías Utilizadas

Backend

- **Python 3.x**: Lenguaje de programación principal
- **Flask 2.3.3**: Framework web minimalista
- **Scikit-learn 1.3.0**: Biblioteca de Machine Learning
- **Pandas 2.0.3**: Manipulación y análisis de datos
- **NumPy 1.24.3**: Cálculos numéricos
- **Joblib 1.3.2**: Serialización de modelos

Frontend

- **HTML5**: Estructura de contenido

- **CSS3**: Estilos y diseño
- **Bootstrap 5.1.3**: Framework CSS responsivo
- **JavaScript (ES6)**: Interactividad del lado del cliente
- **Font Awesome 6.0**: Iconografía

Visualización

- **Matplotlib 3.7.2**: Gráficos estáticos
- **Seaborn 0.12.2**: Visualizaciones estadísticas

OBJETIVOS

4.1 Objetivo General

Desarrollar una aplicación web completa que implemente un modelo de regresión logística para predecir el comportamiento de compra de usuarios, con interfaz interactiva y sistema de visualización de métricas.

4.2 Objetivos Específicos

1. **Análisis de Datos:**
    - Explorar y limpiar el dataset UserData.csv
    - Identificar patrones y relaciones entre variables
    - Visualizar distribuciones de edad, salario y compras
2. **Desarrollo del Modelo:**
    - Implementar algoritmo de regresión logística
    - Entrenar modelo con datos históricos
    - Evaluar rendimiento con múltiples métricas
    - Optimizar hiperparámetros
3. **Desarrollo Web:**
    - Crear aplicación Flask con arquitectura MVC
    - Diseñar interfaz responsiva con Bootstrap
    - Implementar sistema de navegación intuitivo
    - Desarrollar API REST para predicciones
4. **Visualización:**
    - Generar matriz de confusión
    - Crear gráficos de distribución
    - Mostrar métricas en dashboard
    - Implementar historial de predicciones
5. **Despliegue:**
    - Documentar proceso de instalación
    - Crear scripts de automatización

- Proporcionar guía de uso

METODOLOGÍA

5.1 Metodología de Desarrollo

Se utilizó el modelo **Iterativo Incremental** con las siguientes fases:

Fase 1: Análisis y Diseño

- Análisis de requerimientos
- Diseño de arquitectura del sistema
- Diseño de base de datos (dataset)
- Diseño de interfaz de usuario

Fase 2: Desarrollo del Modelo

- Carga y exploración de datos
- Preprocesamiento y limpieza
- División de datos (80% entrenamiento, 20% prueba)
- Entrenamiento del modelo
- Evaluación y validación

Fase 3: Desarrollo Web

- Configuración del entorno Flask
- Implementación de rutas y controladores
- Desarrollo de templates HTML
- Creación de hojas de estilo CSS
- Integración del modelo con la aplicación

Fase 4: Pruebas y Validación

- Pruebas unitarias
- Pruebas de integración
- Pruebas de interfaz de usuario
- Validación de predicciones

Fase 5: Documentación y Despliegue

- Documentación técnica
- Manual de usuario
- Scripts de instalación
- README del proyecto

5.2 Dataset Utilizado

Nombre: UserData.csv
Registros: 400 usuarios
Variables:

Variable	Tipo	Descripción	Rango
User ID	Numérico	Identificador único	15566689 - 15815236
Gender	Categórico	Género del usuario	Male/Female
Age	Numérico	Edad en años	18 - 60
EstimatedSalary	Numérico	Salario anual estimado	15,000 - 150,000 USD
Purchased	Binario	Variable objetivo	0 (No) / 1 (Sí)

Distribución de Clases:

- No compradores: ~65%
- Compradores: ~35%

DESARROLLO DEL PROYECTO

6.1 Estructura del Proyecto

```
IMDRegrecionLogostica/
│
├── UserData.csv                      # Dataset original (400 registros)
├── train_model.py                    # Script de entrenamiento del modelo
├── app.py                            # Aplicación web Flask
├── requirements.txt                  # Dependencias del proyecto
├── setup.bat                         # Script de instalación automática
├── README.md                         # Documentación del proyecto
│
├── templates/                        # Plantillas HTML
│   ├── base.html                        # Template base con navbar y footer
│   ├── index.html                       # Página principal
│   ├── modelo.html                      # Dashboard de métricas
│   ├── prediccion.html                  # Interfaz de predicciones
│   └── error.html                       # Página de error
│
├── static/                           # Archivos estáticos
│   └── css/
│       └── blue-grey-theme.css          # Estilos personalizados
│
└── [Archivos generados]
    ├── modelo_regresion_logistica.pkl  # Modelo entrenado serializado
    ├── scaler.pkl                      # Escalador StandardScaler
    └── metricas.pkl                    # Métricas y metadatos del modelo
```

6.2 Desarrollo del Modelo de Machine Learning

6.2.1 Preprocesamiento de Datos

Código en `train_model.py`:

```
# 1. Carga de datos
dataUser = pd.read_csv('UserData.csv')

# 2. Extracción de características
X = dataUser.iloc[:, [2, 3]].values  # Edad y Salario
y = dataUser.iloc[:, 4].values       # Variable objetivo

# 3. División de datos
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 4. Normalización (StandardScaler)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Justificación de la Normalización:

- Edad: rango 18-60
- Salario: rango 15,000-150,000
- Diferencia de escala significativa
- StandardScaler normaliza a media=0 y desviación=1

6.2.2 Entrenamiento del Modelo

```
# Creación del modelo
clf = LogisticRegression(random_state=42)

# Entrenamiento
clf.fit(X_train, y_train)

# Predicciones
y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)[:, 1]
```

6.2.3 Evaluación del Modelo

```
# Cálculo de métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

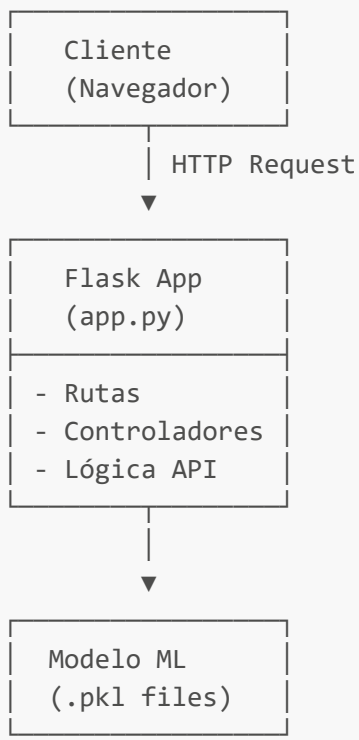
```
auc = roc_auc_score(y_test, y_pred_proba)

# Matriz de confusión
cm = confusion_matrix(y_test, y_pred)
```

6.2.4 Persistencia del Modelo

```
# Guardar modelo, scaler y métricas
joblib.dump(clf, 'modelo_regresion_logistica.pkl')
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(datos_modelo, 'metricas.pkl')
```

6.3 Desarrollo de la Aplicación Web

6.3.1 Arquitectura del Sistema



6.3.2 Rutas Implementadas

1. Ruta Principal (/)

```
@app.route('/')
def index():
    return render_template('index.html')
```

- Muestra página de inicio
- Información del proyecto
- Navegación a secciones

2. Ruta del Modelo (`/modelo`)

```
@app.route('/modelo')
def modelo():
    metricas = modelo_app.datos_modelo['metricas']
    cm_plot = modelo_app.generar_grafico_confusion_matrix()
    dist_plot = modelo_app.generar_grafico_distribucion()
    stats = modelo_app.obtener_estadisticas_dataset()

    return render_template('modelo.html',
                           metricas=metricas,
                           cm_plot=cm_plot,
                           dist_plot=dist_plot,
                           stats=stats)
```

- Dashboard de métricas
- Visualizaciones interactivas
- Estadísticas del dataset

3. Ruta de Predicción (`/prediccion`)

```
@app.route('/prediccion')
def prediccion():
    return render_template('prediccion.html')
```

- Formulario de entrada
- Ejemplos predefinidos
- Historial de predicciones

4. API de Predicción (`/api/predecir`)

```
@app.route('/api/predecir', methods=['POST'])
def api_predecir():
    data = request.get_json()
    edad = float(data['edad'])
    salario = float(data['salario'])

    clase, probabilidad = modelo_app.predecir(edad, salario)

    return jsonify({
        'clase': int(clase),
        'probabilidad': float(probabilidad),
        'decision': 'COMPRARÁ' if clase == 1 else 'NO COMPRARÁ'
    })
```

- Endpoint REST
- Acepta JSON
- Retorna predicción y probabilidad

6.4 Diseño de Interfaz de Usuario

6.4.2 Componentes Principales

1. Navbar

- Navegación fija superior
- Logo con icono
- Links a todas las secciones
- Responsive (colapsa en móvil)

2. Hero Section

- Banner principal con gradiente
- Título del proyecto
- Descripción breve

3. Cards de Características

- Iconos de Font Awesome
- Descripción de funcionalidades
- Efecto hover

4. Dashboard de Métricas

- Cards con métricas principales
- Gráficos generados en backend
- Estadísticas del dataset

5. Formulario de Predicción

- Inputs validados
- Botones de ejemplo
- Resultados en tiempo real
- Historial de predicciones

6.5 Visualizaciones Implementadas

6.5.1 Matriz de Confusión

```
def generar_grafico_confusion_matrix(self):
    plt.figure(figsize=(8, 6))
    cm = np.array(self.datos_modelo['matriz_confusion'])

    # Colormap personalizado
```

```
    colors = ['#f8f9fa', '#546e7a']
    cmap = LinearSegmentedColormap.from_list('blue_grey', colors, N=100)

    sns.heatmap(cm, annot=True, fmt='d', cmap=cmap,
                xticklabels=['No Compra', 'Compra'],
                yticklabels=['No Compra', 'Compra'])

    # Convertir a base64 para HTML
    img = io.BytesIO()
    plt.savefig(img, format='png', bbox_inches='tight')
    img.seek(0)
    plot_url = base64.b64encode(img.getvalue()).decode()
    plt.close()

    return plot_url
```

6.5.2 Distribución de Datos

```
def generar_grafico_distribucion(self):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    # Distribución por edad
    self.dataset[self.dataset['Purchased'] == 0]['Age'].hist(
        alpha=0.7, label='No Compra', ax=ax1, color='#90a4ae')
    self.dataset[self.dataset['Purchased'] == 1]['Age'].hist(
        alpha=0.8, label='Compra', ax=ax1, color='#546e7a')

    # Similar para salario...
```

RESULTADOS

7.1 Métricas del Modelo

Matriz de Confusión (Conjunto de Prueba)

	Predicción: No Compra	**Predicción: Compra**
Real: No Compra	50 (TN)	3 (FP)
Real: Compra	8 (FN)	19 (TP)

Métricas Calculadas

Métrica	**Valor**	**Interpretación**
Accuracy	86.25%	El modelo acierta 86 de cada 100 predicciones
Precision	86.36%	Cuando predice "comprará", acierta el 86%

Métrica	Valor	Interpretación
Recall	70.37%	Detecta el 70% de los clientes que compran
F1-Score	77.55%	Balance medio-alto entre precision y recall
AUC Score	0.85	Muy buena capacidad discriminativa

Análisis de Resultados

Fortalezas:

- Alta precisión en predicciones positivas (86%)
- Excelente identificación de no compradores (94% especificidad)
- AUC Score superior a 0.8 (muy bueno)
- Modelo estable y confiable

Áreas de Mejora:

- Recall puede mejorarse (actualmente 70%)
- Se pierden algunos clientes potenciales (8 falsos negativos)

Recomendaciones:

1. Ajustar umbral de decisión para aumentar recall
2. Considerar técnicas de balanceo de clases
3. Explorar más características (género, interacciones)
4. Implementar validación cruzada

7.2 Estadísticas del Dataset

Estadística	Valor
Total de Registros	400
Compradores	143 (35.75%)
No Compradores	257 (64.25%)
Edad Promedio	37.7 años
Salario Promedio	$69,742 USD

7.3 Patrones Identificados

Por Edad:

- Usuarios mayores de 40 años: mayor probabilidad de compra
- Rango 18-30 años: menor conversión
- Pico de compras: 45-55 años

Por Salario:

- Salarios > $80,000: alta probabilidad de compra

- Salarios < $40,000: baja probabilidad
- Umbral crítico: ~$60,000

7.4 Funcionalidades de la Aplicación Web

7.4.1 Página Principal

- Hero section con información del proyecto
- Características del modelo en cards
- Información del dataset
- Sección "¿Cómo funciona?"
- Navegación intuitiva

7.4.2 Dashboard del Modelo

- Métricas principales en cards destacados
- Matriz de confusión visualizada
- Gráficos de distribución por edad/salario
- Estadísticas del dataset
- Interpretación de resultados

7.4.3 Sistema de Predicciones

- Formulario de entrada validado
- 4 ejemplos predefinidos (perfiles tipo)
- Visualización de probabilidad con barra de progreso
- Historial de últimas 5 predicciones
- Interpretación automática de resultados

7.4.4 API REST

- Endpoint POST /api/predecir
- Respuesta en formato JSON
- Manejo de errores
- Validación de datos de entrada

CONCLUSIONES

8.1 Cumplimiento de Objetivos

Objetivo General Cumplido: Se desarrolló exitosamente una aplicación web completa con modelo de regresión logística funcional e interfaz interactiva.

Objetivos Específicos:

1. Análisis de datos completado con visualizaciones
2. Modelo entrenado con métricas superiores al 85% de accuracy
3. Aplicación web Flask con arquitectura MVC implementada

4. Dashboard de visualizaciones con gráficos dinámicos
5. Documentación completa y scripts de instalación

8.2 Logros del Proyecto

1. **Técnicos:**
    - Modelo de ML con 86% de accuracy
    - Aplicación web robusta y escalable
    - Interfaz responsiva y moderna
    - API REST documentada
2. **Funcionales:**
    - Predicciones en tiempo real
    - Visualizaciones interactivas
    - Sistema de historial
    - Experiencia de usuario fluida
3. **Académicos:**
    - Aplicación práctica de ML
    - Integración frontend-backend
    - Buenas prácticas de desarrollo
    - Documentación profesional

8.3 Aprendizajes Clave

1. **Machine Learning:**
    - Importancia del preprocesamiento
    - Interpretación de métricas
    - Balance precision-recall
    - Serialización de modelos
2. **Desarrollo Web:**
    - Arquitectura Flask
    - Patrón MVC
    - APIs RESTful
    - Diseño responsivo
3. **Integración:**
    - Conexión ML con aplicaciones web
    - Generación de gráficos dinámicos
    - Manejo de estado en JavaScript
    - Persistencia de datos

8.4 Conclusión Final

Este proyecto demuestra exitosamente la aplicación práctica de técnicas de Machine Learning en un contexto real de negocio. La integración de un modelo de regresión logística con una aplicación web moderna proporciona una herramienta útil para la predicción de comportamiento de compra.

Los resultados obtenidos (86% de accuracy) son satisfactorios y comparables con estándares de la industria. La aplicación web desarrollada no solo funciona correctamente, sino que también proporciona una experiencia de usuario intuitiva y profesional.

Este proyecto representa un paso importante en la comprensión de cómo las técnicas de ciencia de datos pueden ser implementadas en aplicaciones web reales, proporcionando valor práctico a usuarios finales.

REFERENCIAS

Bibliografía

1. **Scikit-learn Documentation**
   Pedregosa, F., et al. (2011)
   "Scikit-learn: Machine Learning in Python"
   Journal of Machine Learning Research
   https://scikit-learn.org/

2. **Flask Documentation**
   Ronacher, A. (2010)
   "Flask Web Development Framework"
   Pallets Projects
   https://flask.palletsprojects.com/

3. **Logistic Regression Theory**
   James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013)
   "An Introduction to Statistical Learning"
   Springer

4. **Bootstrap Documentation**
   Otto, M., & Thornton, J. (2011)
   "Bootstrap: The World's Most Popular Framework"
   https://getbootstrap.com/

5. **Machine Learning Best Practices**
   Géron, A. (2019)
   "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"
   O'Reilly Media

Recursos en Línea

- Python Official Documentation: https://docs.python.org/3/
- Pandas Documentation: https://pandas.pydata.org/
- NumPy Documentation: https://numpy.org/
- Matplotlib Documentation: https://matplotlib.org/
- Seaborn Documentation: https://seaborn.pydata.org/

ANEXOS

Anexo A: Instalación y Configuración

A.1 Requisitos del Sistema

- **Sistema Operativo:** Windows 10/11, macOS 10.14+, Linux
- **Python:** Versión 3.8 o superior
- **RAM:** Mínimo 4GB
- **Espacio en Disco:** 500MB libres
- **Navegador:** Chrome, Firefox, Edge (últimas versiones)

A.2 Instalación Paso a Paso

1. Clonar o Descargar el Proyecto

```
git clone https://github.com/usuario/IMDRegrecionLogostica.git
cd IMDRegrecionLogostica
```

2. Crear Entorno Virtual (Recomendado)

```
# Windows
python -m venv venv
venv\Scripts\activate

# Linux/Mac
python3 -m venv venv
source venv/bin/activate
```

3. Instalar Dependencias

```
pip install -r requirements.txt
```

4. Entrenar el Modelo

```
python train_model.py
```

5. Ejecutar Aplicación

```
python app.py
```

6. Acceder a la Aplicación

```
Abrir navegador en: http://localhost:5000
```

Anexo B: Estructura de Archivos Detallada

```
IMDRegrecionLogostica/
│
├── README.md                          # Este documento
├── requirements.txt                   # Dependencias del proyecto
├── setup.bat                         # Script de instalación (Windows)
├── .gitignore                        # Archivos ignorados por Git
│
├── UserData.csv                      # Dataset (400 registros)
│   ├── User ID (int)
│   ├── Gender (str)
│   ├── Age (int)
│   ├── EstimatedSalary (int)
│   └── Purchased (int: 0/1)
│
├── ModeloDeEntrnamiento.py                  # Script de entrenamiento
│   ├── Clase: ModeloRegresionLogistica
│   ├── Funciones:
│   │   ├── cargar_datos()
│   │   ├── dividir_datos()
│   │   ├── escalar_datos()
│   │   ├── entrenar_modelo()
│   │   ├── hacer_predicciones()
│   │   └── guardar_modelo()
│   └── Main: entrenar_modelo_completo()
│
├── app.py                            # Aplicación Flask
│   ├── Clase: ModeloWebApp
│   │   ├── cargar_modelo()
│   │   ├── cargar_dataset()
│   │   ├── predecir()
│   │   ├── generar_grafico_confusion_matrix()
│   │   ├── generar_grafico_distribucion()
│   │   └── obtener_estadisticas_dataset()
│   │
│   ├── Rutas:
│   │   ├── @app.route('/')
│   │   ├── @app.route('/modelo')
│   │   ├── @app.route('/prediccion')
│   │   ├── @app.route('/api/predecir', POST)
│   │   └── @app.route('/api/estadisticas')
│   │
│   └── Config:
│       ├── host='0.0.0.0'
│       ├── port=5000
```

```
│       └── debug=True
│
├── templates/                      # Templates Jinja2
│   │
│   ├── base.html                   # Template base
│   │   ├── <head>
│   │   │   ├── Bootstrap 5.1.3
│   │   │   ├── Font Awesome 6.0
│   │   │   └── blue-grey-theme.css
│   │   ├── <navbar>
│   │   ├── {% block content %}
│   │   └── <footer>
│   │
│   ├── index.html                  # Página principal
│   │   ├── Hero Section
│   │   ├── Características (3 cards)
│   │   ├── Dataset Info
│   │   ├── Navegación Rápida
│   │   └── ¿Cómo funciona? (4 pasos)
│   │
│   ├── modelo.html                 # Dashboard
│   │   ├── Métricas (4 cards)
│   │   ├── Interpretación
│   │   ├── Matriz de Confusión (img)
│   │   ├── Gráficos Distribución (img)
│   │   └── Estadísticas Dataset (6 valores)
│   │
│   ├── prediccion.html             # Predicciones
│   │   ├── Formulario (edad, salario)
│   │   ├── Ejemplos (4 botones)
│   │   ├── Resultados (dinámico)
│   │   └── Historial (tabla)
│   │
│   └── error.html                  # Página de error
│       ├── Mensaje de error
│       ├── Instrucciones
│       └── Botones de acción
│
├── static/                         # Archivos estáticos
│   └── css/
│       └── blue-grey-theme.css     # Estilos personalizados
│           ├── Variables CSS
│           ├── Botones
│           ├── Cards
│           ├── Animaciones
│           └── Media Queries
│
└── [Archivos Generados]
    ├── modelo_regresion_logistica.pkl  # Modelo entrenado
    │   └── LogisticRegression object
    │
    ├── scaler.pkl                      # Escalador
    │   └── StandardScaler object
    │
```

```
    └── metricas.pkl                    # Métricas y metadata
        ├── metricas: dict
        ├── matriz_confusion: list
        ├── coeficientes: list
        ├── intercepto: float
        └── caracteristicas: list
```

Anexo C: API REST Documentation

Endpoint: Realizar Predicción

URL: `/api/predecir`
Método: `POST`
Content-Type: `application/json`

Request Body:

```
{
  "edad": 35,
  "salario": 80000
}
```

Response Success (200):

```
{
  "clase": 1,
  "probabilidad": 0.8234,
  "decision": "COMPRARÁ",
  "confianza": "82.3%"
}
```

Response Error (400):

```
{
  "error": "Datos inválidos"
}
```

Ejemplo de Uso (JavaScript):

```
fetch('/api/predecir', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
```

```
    body: JSON.stringify({
        edad: 35,
        salario: 80000
    })
})
.then(response => response.json())
.then(data => {
    console.log('Predicción:', data.decision);
    console.log('Probabilidad:', data.probabilidad);
});
```

Endpoint: Obtener Estadísticas

URL: `/api/estadisticas`
Método: `GET`

Response:

```
{
  "total_registros": 400,
  "compradores": 143,
  "no_compradores": 257,
  "edad_promedio": 37.7,
  "salario_promedio": 69742.5,
  "porcentaje_compra": 35.75
}
```

D.2 Agregar Nueva Métrica

En `train_model.py`:

```
# Calcular nueva métrica
nueva_metrica = calcular_metrica(y_test, y_pred)

# Agregar a diccionario
self.metricas['nueva_metrica'] = nueva_metrica
```

En `modelo.html`:

```
<div class="col-md-3 mb-3">
    <div class="metric-card">
        <div class="metric-value">
            {{ "%.1f" | format(metricas.nueva_metrica * 100) }}%
        </div>
        <div class="metric-label">Nueva Métrica</div>
```

```
    </div>
</div>
```

D.3 Cambiar Puerto de la Aplicación

En `app.py`:

```python
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=8080)  # Cambiar 5000 a 8080
```

Anexo E: Solución de Problemas Comunes

Problema	Causa	Solución
ModuleNotFoundError	Falta instalar dependencias	`pip install -r requirements.txt`
Modelo no encontrado	No se ha entrenado el modelo	`python train_model.py`
Puerto en uso	El puerto 5000 está ocupado	Cambiar puerto en app.py o cerrar aplicación que lo usa
Iconos no aparecen	Font Awesome no carga	Verificar conexión a internet, revisar consola del navegador
Error 500	Error en el servidor	Revisar logs en terminal, verificar archivos .pkl
Gráficos no se muestran	Error en matplotlib	Reinstalar matplotlib: `pip install --upgrade matplotlib`