SVM-RESEARCHER

Logotipo

Descripción generada automáticamente

*Código Fuente*

**SVM-RESEARCHER**

**Autores:**

1 Marco Fidel Mayta Quispe

2

3

**ÍNDICE**

**04**

**Código Fuente**

**05**

**Archivo: app.r**

**1. CÓDIGO FUENTE:**

La aplicación "SVM Researcher", desarrollada con Python y Streamlit, es una solución avanzada y completamente interactiva diseñada para facilitar el entrenamiento, evaluación y despliegue de modelos de clasificación basados en Máquinas de Soporte Vectorial (SVM), brindando a los usuarios herramientas intuitivas para explorar y optimizar modelos de aprendizaje automático. Su diseño contempla dos funcionalidades principales: entrenar modelos SVM desde cero con datos personalizados y cargar modelos previamente entrenados para realizar predicciones en tiempo real, lo que ofrece una gran flexibilidad tanto para experimentación académica como para su uso en entornos profesionales. Los usuarios pueden cargar archivos CSV utilizando distintos separadores, como comas, puntos y comas o tabulaciones, para adaptarse a diferentes formatos de datos, y seleccionar fácilmente la columna objetivo (variable dependiente) para el entrenamiento del modelo. La aplicación realiza automáticamente un preprocesamiento de los datos, incluyendo la codificación de variables categóricas mediante One-Hot Encoding, la normalización de características numéricas con ‘StandardScaler’, y la integración de las variables en un conjunto de datos preparado para el aprendizaje automático, asegurando consistencia y calidad en las entradas.

En cuanto a la configuración del modelo, "SVM Analyzer" permite a los usuarios personalizar parámetros avanzados como el tipo de kernel (‘linear’, ‘rbf’, ‘poly’, ‘sigmoid’), el valor de penalización (C) para regularización, los hiperparámetros gamma y el grado de polinomio para kernels específicos, así como la opción de balancear clases desbalanceadas mediante ajustes automáticos de pesos. El proceso de entrenamiento incluye una barra de progreso visual que no solo informa al usuario del avance, sino que mejora la experiencia de interacción, haciendo que el proceso sea más transparente y dinámico. Una vez entrenado el modelo, la aplicación evalúa su desempeño dividiendo los datos en conjuntos de entrenamiento y prueba, generando métricas clave como precisión, recall, F1-score y accuracy, las cuales son presentadas en un informe tabular claro y detallado, que facilita la interpretación de los resultados.

Adicionalmente, la aplicación permite descargar el modelo entrenado en un archivo ‘.pkl’, que incluye tanto el modelo SVM como los pasos de preprocesamiento necesarios para garantizar compatibilidad en usos futuros, además de permitir cargar estos modelos entrenados para realizar predicciones en nuevos datos. "SVM Researcher" también incluye una funcionalidad para realizar predicciones con entradas manuales, donde los usuarios pueden ingresar valores específicos para cada característica, asegurando que el preprocesamiento sea consistente con los datos utilizados en el entrenamiento, incluso en presencia de variables categóricas. Esta versatilidad y atención al detalle hacen de la aplicación una herramienta ideal para estudiantes, investigadores y profesionales que buscan explorar modelos SVM de manera eficiente, comprender su funcionamiento y aplicarlos en la resolución de problemas prácticos en múltiples contextos.

ARCHIVO: APP.PY

import streamlit as st

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

from sklearn.preprocessing import StandardScaler

import time

import pickle

import base64

import io

st.title("Aplicación  SVM")

st.write("Marco Mayta - Facultad de Ingeniería Estadística e Informática - UNA PUNO")

st.write("Sube un archivo CSV, selecciona el separador, elige la columna objetivo, y configura el modelo SVM.")

tab1, tab2 = st.tabs(["Entrenar Nuevo Modelo", "Cargar Modelo Existente"])

with tab1:

    separator = st.selectbox("Selecciona el separador de columnas en tu archivo CSV", [

                             ",", ";", " ", "|", "\t"], index=0)

    uploaded\_file = st.file\_uploader(

        "Sube un archivo CSV", type="csv", key="train\_data")

    if uploaded\_file is not None:

        df = pd.read\_csv(uploaded\_file, sep=separator)

        st.write("Datos cargados:")

        st.write(df.head())

        st.write("Columnas disponibles:", df.columns.tolist())

        st.subheader(

            "Paso 2: Selección de columna objetivo y limpieza de datos")

        target\_column = st.selectbox(

            "Selecciona la columna objetivo (clase a predecir)", df.columns)

        if target\_column not in df.columns:

            st.error(

                f"La columna objetivo '{target\_column}' no se encuentra en los datos. Por favor, verifica el archivo.")

        else:

            st.write("\*\*Preprocesamiento de datos\*\*")

            df\_features = df.drop(columns=[target\_column])

            if df\_features.select\_dtypes(include=['object']).empty:

                st.write(

                    "No se encontraron columnas categóricas, omitiendo One-Hot Encoding.")

            else:

                df\_features = pd.get\_dummies(df\_features, drop\_first=True)

                st.write(

                    "Datos después de aplicar One-Hot Encoding a las características:")

                st.write(df\_features.head())

            df = pd.concat([df\_features, df[target\_column]], axis=1)

            X = df.drop(columns=[target\_column]).values

            y = df[target\_column].values

            if X.shape[1] == 0:

                st.error(

                    "No hay columnas numéricas disponibles después de procesar los datos. Verifica el archivo y elige la columna objetivo correctamente.")

            else:

                scaler = StandardScaler()

                X = scaler.fit\_transform(X)

                st.subheader("Paso 4: Configuración avanzada del modelo SVM")

                kernel\_type = st.selectbox("Selecciona el tipo de kernel", [

                                           "linear", "rbf", "poly", "sigmoid"])

                C\_value = st.slider("Valor de penalización (C)",

                                    min\_value=0.01, max\_value=10.0, value=1.0)

                if kernel\_type in ["rbf", "poly"]:

                    gamma\_value = st.slider("Valor de gamma", min\_value=0.01, max\_value=1.0,

                                            value=0.1, help="Controla la forma del kernel RBF o polinómico.")

                if kernel\_type == "poly":

                    degree\_value = st.slider(

                        "Grado del polinomio", min\_value=1, max\_value=5, value=3)

                class\_weight\_opt = st.selectbox("¿Quieres balancear las clases?", [

                                                None, "balanced"], help="Usar 'balanced' ajusta el peso de las clases automáticamente.")

                X\_train, X\_test, y\_train, y\_test = train\_test\_split(

                    X, y, test\_size=0.3, random\_state=42)

                st.subheader("Paso 5: Entrenamiento y evaluación del modelo")

                if kernel\_type == "linear":

                    model = SVC(kernel=kernel\_type, C=C\_value,

                                class\_weight=class\_weight\_opt)

                elif kernel\_type == "rbf":

                    model = SVC(kernel=kernel\_type, C=C\_value,

                                gamma=gamma\_value, class\_weight=class\_weight\_opt)

                elif kernel\_type == "poly":

                    model = SVC(kernel=kernel\_type, C=C\_value, gamma=gamma\_value,

                                degree=degree\_value, class\_weight=class\_weight\_opt)

                else:

                    model = SVC(kernel=kernel\_type, C=C\_value,

                                class\_weight=class\_weight\_opt)

                with st.spinner('Entrenando modelo...'):

                    progress = st.progress(0)

                    for percent\_complete in range(100):

                        time.sleep(0.01)

                        progress.progress(percent\_complete + 1)

                    model.fit(X\_train, y\_train)

                st.success("Entrenamiento completado")

                y\_pred = model.predict(X\_test)

                report = classification\_report(

                    y\_test, y\_pred, output\_dict=True, zero\_division=0)

                report\_df = pd.DataFrame(report).transpose()

                st.write("\*\*Resultados del modelo:\*\*")

                st.dataframe(report\_df)

                st.write(f"\*\*Precisión (Accuracy)\*\*: {report['accuracy']:.2f}")

                st.write(

                    f"\*\*Precisión (weighted)\*\*: {report['weighted avg']['precision']:.2f}")

                st.write(

                    f"\*\*Recall (weighted)\*\*: {report['weighted avg']['recall']:.2f}")

                st.write(

                    f"\*\*F1-Score (weighted)\*\*: {report['weighted avg']['f1-score']:.2f}")

                st.subheader("Descargar Modelo Entrenado")

                model\_data = {

                    'model': model,

                    'scaler': scaler,

                    'feature\_names': df\_features.columns.tolist(),

                    'target\_column': target\_column,

                    'original\_columns': list(df.drop(columns=[target\_column]).columns),

                    'categorical\_columns': list(df.select\_dtypes(include=['object']).columns),

                    'numeric\_columns': list(df.select\_dtypes(exclude=['object']).columns)

                }

                buff = io.BytesIO()

                pickle.dump(model\_data, buff)

                buff.seek(0)

                b64 = base64.b64encode(buff.read()).decode()

                href = f'<a href="data:file/pickle;base64,{b64}" download="modelo\_svm.pkl">Descargar Modelo Entrenado</a>'

                st.markdown(href, unsafe\_allow\_html=True)

                st.subheader("Probar Modelo - Entrada Manual")

                with st.form(key='prediction\_form'):

                    st.write("Ingrese los valores para realizar una predicción:")

                    input\_data = {}

                    for col in df.drop(columns=[target\_column]).columns:

                        if col in df.select\_dtypes(include=['object']).columns:

                            unique\_values = df[col].unique().tolist()

                            input\_data[col] = st.selectbox(

                                f"Seleccione valor para {col}", unique\_values)

                        else:

                            input\_data[col] = st.number\_input(

                                f"Ingrese valor para {col}", value=0.0)

                    submit\_button = st.form\_submit\_button(

                        label='Realizar Predicción')

                if submit\_button:

                    input\_df = pd.DataFrame([input\_data])

                    if not input\_df.select\_dtypes(include=['object']).empty:

                        input\_df = pd.get\_dummies(input\_df, drop\_first=True)

                    for col in df\_features.columns:

                        if col not in input\_df.columns:

                            input\_df[col] = 0

                    input\_df = input\_df[df\_features.columns]

                    X\_new = scaler.transform(input\_df.values)

                    prediction = model.predict(X\_new)

                    st.success(f"La predicción es: {prediction[0]}")

    else:

        st.info("Por favor, sube un archivo CSV para continuar.")

with tab2:

    st.subheader("Cargar Modelo Existente")

    uploaded\_model = st.file\_uploader(

        "Sube un modelo guardado (.pkl)", type="pkl")

    if uploaded\_model is not None:

        try:

            model\_data = pickle.load(uploaded\_model)

            loaded\_model = model\_data['model']

            loaded\_scaler = model\_data['scaler']

            feature\_names = model\_data['feature\_names']

            target\_column = model\_data['target\_column']

            if 'original\_columns' in model\_data:

                original\_columns = model\_data['original\_columns']

                categorical\_columns = model\_data.get('categorical\_columns', [])

                numeric\_columns = model\_data.get('numeric\_columns', [])

            else:

                original\_columns = feature\_names

                categorical\_columns = []

                numeric\_columns = original\_columns

            st.success("Modelo cargado correctamente")

            st.subheader("Probar Modelo Cargado - Entrada Manual")

            with st.form(key='loaded\_model\_form'):

                st.write("Ingrese los valores para realizar una predicción:")

                input\_data = {}

                for col in original\_columns:

                    if col in categorical\_columns:

                        input\_data[col] = st.text\_input(

                            f"Ingrese valor para {col}")

                    else:

                        input\_data[col] = st.number\_input(

                            f"Ingrese valor para {col}", value=0.0)

                submit\_button = st.form\_submit\_button(

                    label='Realizar Predicción')

            if submit\_button:

                input\_df = pd.DataFrame([input\_data])

                if categorical\_columns:

                    input\_df = pd.get\_dummies(input\_df, drop\_first=True)

                for col in feature\_names:

                    if col not in input\_df.columns:

                        input\_df[col] = 0

                input\_df = input\_df[feature\_names]

                X\_new = loaded\_scaler.transform(input\_df.values)

                prediction = loaded\_model.predict(X\_new)

                st.success(f"La predicción es: {prediction[0]}")

        except Exception as e:

            st.error(f"Error al cargar el modelo: {str(e)}")

            st.write(

                "Por favor, asegúrese de que el archivo .pkl sea un modelo SVM válido.")

Logotipo

Descripción generada automáticamente