



Manual de programador para el prototipo de algoritmo para la predicción de precipitaciones a partir de datos estadísticos recolectados de la estación meteorológica de la Universidad De Cundinamarca Seccional Ubaté (MCP)



Tabla de contenido

Introducción	4
Acerca de este Manual	4
Objetivos del Software	4
Especificaciones del software	6
Desarrollo y Estructura del Software	7
Arquitectura del Software	7
Componentes del sistema	7
Casos de uso del funcionamiento del sistema	8
Tecnologías utilizadas	9
Configuración del entorno de desarrollo	10
Requisitos del entorno de desarrollo	10
Versiones de dependencias utilizadas.....	10
Instalación de herramientas y dependencias.....	10
Configuración de la base de datos	12
Estructura del código fuente.....	13
Organización de carpetas	13
Comentarios y documentación dentro del código	14
Codificación de los algoritmos.....	15
Descripción de los algoritmos utilizados.....	15
Desarrollo de la interfaz de usuario.....	17
Creación de rutas para los módulos	18
Diseño y estructura de la interfaz	18



Preguntas Frecuentes (FAQ).....	21
Glosario.....	22
Referencias.....	23



Introducción

Acerca de este Manual

El presente manual contiene los aspectos técnicos del aplicativo MCPP y aspectos a cumplir necesarios para su uso, además de información para los usuarios interesados en conocer la predicción de precipitaciones mediante un modelo desarrollado para proporcionar información relevante sobre las posibles futuras lluvias, basándose en la toma de datos de la estación meteorológica de la Universidad de Cundinamarca Seccional Ubaté, basado en la ejecución del proyecto denominado “Desarrollo de un prototipo de algoritmo para la predicción de precipitaciones a partir de datos estadísticos recolectados de la estación meteorológica de la Universidad De Cundinamarca Seccional Ubaté” sustentando en una serie de objetivos que empiezan desde el análisis de datos, creación del algoritmo e implementarlo en el modelo hasta el desarrollo del aplicativo MCPP para la visualización de los datos y las predicciones de precipitaciones dicho modelo esta implementado en un aplicativo diseñado para ser intuitivo, agradable y útil para los usuarios que lo utilicen.

Este manual proporciona una guía importante para los usuarios que deseen conocer el funcionamiento del aplicativo MCPP (Modelo Comparativo Predictivo de Precipitaciones).

Objetivos del Software

El objetivo principal del presente software es proporcionar información a los usuarios acerca de los porcentajes de efectividad y predicciones de los modelos Random Forest, Naive Bayes, Support Vector Machine y MCPP, los cuales manejan técnicas de predicción diferentes, por lo cual permiten a los usuarios tomar decisiones basados en sus necesidades. Dicho software también tiene como objetivo proporcionar una herramienta para probar los modelos con datos



que el usuario posea, para así darle a conocer porcentajes de efectividad y predicciones futuras con base en los datos proporcionados.



Especificaciones del software

Es necesario cumplir con los siguientes requisitos previos a la utilización de la página para poder hacer uso óptimo de todas sus funciones:

- **Contar con el aplicativo instalado:** Es necesario contar con el aplicativo instalado para el funcionamiento del mismo.
- **Navegadores web:** El aplicativo puede ser visitado desde cualquier navegador moderno, entre ellos Opera, Microsoft Edge, Brave, Google Chrome, etc.
- **Sistema operativo:** Para poder acceder al aplicativo solo es necesario contar con un navegador, por lo cual no se requiere un sistema operativo en especial.
- **Cuenta de usuario:** Para acceder a las funcionalidades principales del aplicativo no es necesario contar con una cuenta de usuario, a excepción de contar con privilegios de administrador para acceder a la base de datos del modelo, en dicho caso se asignará una cuenta preestablecida a quienes tengan el derecho de acceso.
- **Tamaño de pantalla:** El sistema cuenta con un diseño adaptable para cualquier tamaño de pantalla, por lo cual puede ser utilizado tanto en dispositivos móviles como computadores.

Desarrollo y Estructura del Software

Arquitectura del Software

La arquitectura del aplicativo de predicción de precipitaciones está diseñada con el fin de ofrecer una experiencia agradable y precisa para los usuarios, esta se basa en tres módulos principales: el modulo principal, el módulo de probar modelo y el módulo de administrador. La información detallada sobre el diseño de estos módulos puede ser encontrado en el Manual de Usuario.

Componentes del sistema

Módulo Principal. Este módulo sirve como punto de entrada para los usuarios al aplicativo. Dentro del cual se pueden visualizar las predicciones dadas por el modelo MCP, Random Forest, Naive Bayes y Support Vector Machines. Además, contiene información sobre los datos alojados en la base de datos, con los cuales es posible realizar las predicciones, widgets de información general sobre el clima y un apartado inferior de información de interés acerca del proyecto y el aplicativo.

Módulo de probar modelo. Este módulo proporciona una herramienta sencilla e intuitiva para el usuario tenga la posibilidad de probar la efectividad del modelo MCP y también de los modelos Random Forest, Naive Bayes y SVM, una forma útil de comprobar el funcionamiento del aplicativo principal, brindando la posibilidad de obtener resultados con datos que el usuario tenga y así mismo asignar un porcentaje de prueba para el debido entrenamiento de los datos ingresados.

Módulo de administrador. Este módulo está enfocado exclusivamente a los usuarios con el rol especial de administrador, a los cuales se les dará una cuenta única previamente para

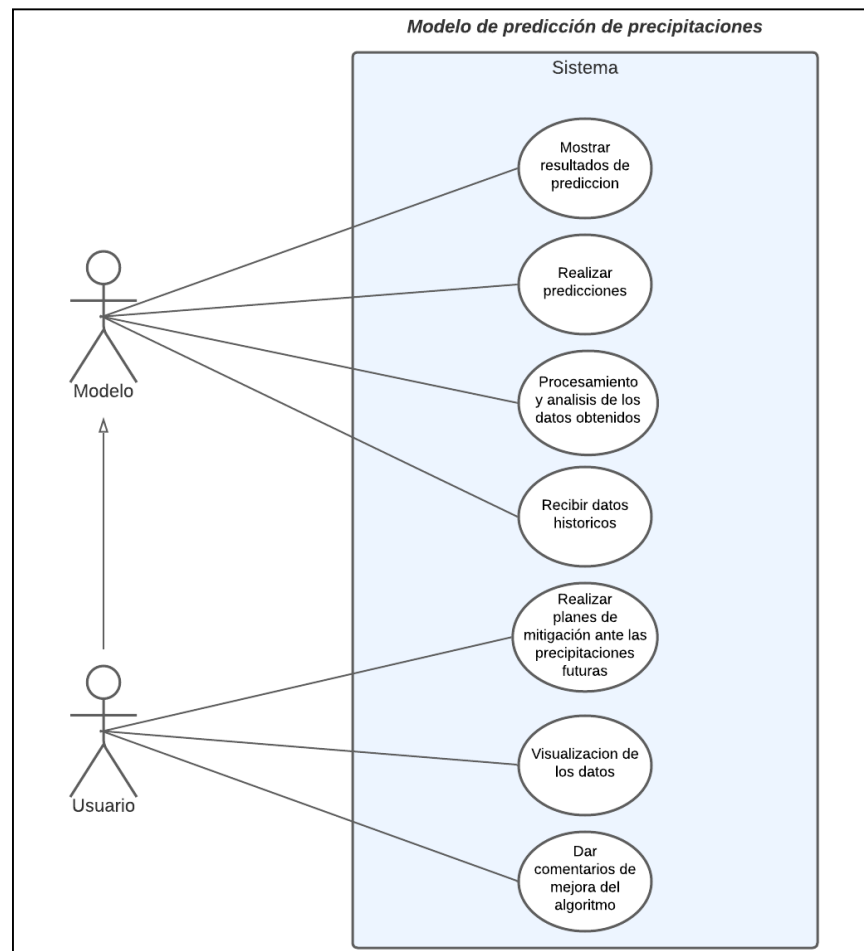
que tengan la posibilidad de visualizar, actualizar, ingresar o eliminar datos dentro de la base de datos del aplicativo (MongoDB).

Casos de uso del funcionamiento del sistema

Para mayor entendimiento y claridad del funcionamiento del sistema se generó un diagrama de casos de uso, con el fin de identificar las funciones tanto de los usuarios como del modelo MCP.

Figura 1

Casos de uso de la funcionalidad del sistema



Nota. Diagrama de casos de uso del sistema, mostrando los roles tanto del modelo como del usuario final.

Tecnologías utilizadas

El desarrollo de los modelos y del aplicativo estuvo representado por la utilización de diversas tecnologías que garantizan un rendimiento eficiente y una experiencia de usuario idónea. Entre dichas tecnologías se encuentran:

Lenguajes de programación. Python para la funcionalidad de los modelos y el backend general del aplicativo.

Frameworks. Flask para el backend y conexión con el diseño del aplicativo, Bootstrap para el diseño de la interfaz de usuario.

Bases de Datos. Mongo para el almacenamiento de los datos meteorológicos del entrenamiento de los modelos y de la estación meteorológica de la Universidad de Cundinamarca Seccional Ubaté.

Bibliotecas y librerías. Scikit-learn para el modelado y entrenamiento de los datos, Pandas y Numpy para la manipulación y análisis de los datos recolectados, matplotlib para graficas e interpretación de los datos.

Lenguajes de diseño. HTML para la estructuración del aplicativo MCP y CSS para el diseño y estilo del aplicativo.

Entorno de desarrollo. Jupyter Notebook para el análisis y desarrollo de los algoritmos y Visual Studio Code para la codificación de los modelos finales.



Configuración del entorno de desarrollo

Requisitos del entorno de desarrollo

Se recomienda el uso del entorno de desarrollo Visual Studio Code, pero cualquier IDE compatible con el lenguaje Python y permita la instalación de las librerías mencionadas es aceptable.

Versiones de dependencias utilizadas

A continuación, se detallan las versiones de las dependencias utilizadas en el desarrollo del aplicativo. Bibliotecas de Python:

- certifi 2023.7.22
- Flask 2.3.3
- Jinja2 3.1.2
- matplotlib 3.8.0
- numpy 1.26.0
- pandas 2.1.1
- pip 23.3.1
- pymongo 4.5.0
- scikit-learn 1.3.0
- scipy 1.11.2
- seaborn 0.12.2

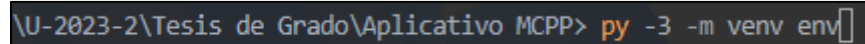
Instalación de herramientas y dependencias

Para la instalación de las librerías necesaria fue necesaria la creación de la carpeta del proyecto y abrirla dentro del IDE VSCode, de esta manera se abrió la terminal dentro del proyecto y se instalaron las librerías, empezando por pip, luego mediante el comando “pip

install” seguido del nombre de la librería requerida fue posible instalar las herramientas necesarias para el desarrollo, para esto fue necesaria la creación de un entorno virtual; el cual se creó con el comando “py -3 -m venv env”.

Figura 2

Creación del entorno virtual en VSCode



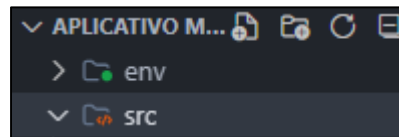
```
\U-2023-2\Tesis de Grado\Aplicativo MCP> py -3 -m venv env
```

Nota. La imagen muestra el comando para la creación del entorno virtual en el proyecto.

De esta manera se creó una carpeta denominada env, luego se creó una carpeta llamada src, la cual contiene todo el desarrollo del aplicativo y los modelos predictivos.

Figura 3

Creación de la carpeta src en VSCode



Nota. La imagen muestra la creación de la carpeta src junto al env en el proyecto.

Para el inicio del desarrollo dentro del entorno virtual se ingresó al mismo mediante la terminal con los comandos “Set-ExecutionPolicy Unrestricted -Scope Process” y “env\Scripts\activate”, de esta manera se instalaron las librerías necesarias, en este caso fueron las encontradas en el apartado de Versiones de dependencias utilizadas, esto mediante el comando “pip install + dependencia”

Figura 4*Instalación de la librería de Flask*

```
\U-2023-2\Tesis de Grado\Aplicativo MCP> pip install Flask
```

Nota. La imagen muestra la instalación de la librería de Flask como ejemplo de cómo se desarrolló la instalación de todas las dependencias.

Configuración de la base de datos

Para la base de datos se utilizó el motor de Mongo Atlas, el cual permite manejar gran cantidad de datos de forma flexible y sencilla. Se creó una colección denominada “Datos” que contiene los datos utilizados para las predicciones y otra colección llamada “Usuarios” que contiene la cuenta de usuario para el modulo administrador.

Figura 5*Base de datos en Mongo Atlas*

Nota. La imagen muestra las colecciones Datos y Usuarios dentro de la base de datos del aplicativo.

Para la conexión dentro del aplicativo se utilizó la librería pymongo y certifi para permitir la conexión del código fuente con la base de datos.

Figura 6

Conexión del aplicativo con Mongo

```
from pymongo import MongoClient
import certifi
from pymongo.collection import ReturnDocument
from pymongo import MongoClient

# Conexión con MongoDB
MONGO = 

# Utilización del certificado
certificado = certifi.where()

# Función para la conexión con la DB
def Conexion():
    try:
        client = MongoClient(MONGO, tlsCAFile = certificado)
        bd = client["bd_Precipitaciones"]
    except ConnectionError:
        print("Error de Conexión")
    return bd
```

Nota. La imagen muestra la conexión del aplicativo con Mongo.

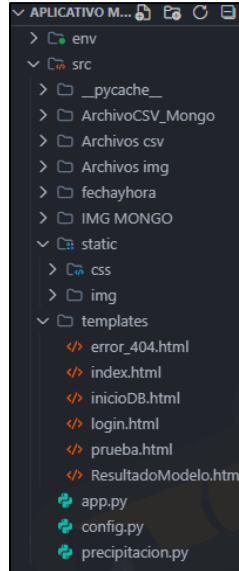
Estructura del código fuente

Organización de carpetas

Para la estructuración del código fuente se organizaron los archivos tipo Python dentro de la carpeta src, los archivos de tipo HTML dentro de una carpeta llamada “templates”, además se creó una carpeta llamada “static” que contiene los archivos tipo CSS dentro de una carpeta “css” y las imágenes del aplicativo dentro de “img”, sumado a esto se crearon carpetas específicas para alojar los gráficos, archivos css e imágenes generales del aplicativo.

Figura 7

Organización de carpetas para el aplicativo MCP



Nota. La imagen muestra la organización de las carpetas para el aplicativo MCP

Comentarios y documentación dentro del código

Para un correcto entendimiento del código se utilizaron comentarios dentro del mismo para entender todos los aspectos posibles en el desarrollo del aplicativo.

Figura 8

Ejemplo de comentarios dentro del código

```
# Se carga el conjunto de datos spam.data
## sep== separacion por comas
## Header== no tiene
df=pd.read_csv("../src/ArchivoCSV_Mongo/filename.csv", header=None, skiprows=1)
# separa los datos en "XRF" y en "YRF"
# "XRF" representa todas las columnas menos la ultima la cual es la de respuesta
# "YRF" representa la columna de respuesta
XRF = df.iloc[:, :-1]
YRF = df.iloc[:, -1]

# Normalizamos los datos con Logaritmos neperianos
XRF_normalizada=np.log1p(df.iloc[:, :-1])

# configuracion dl modelo con datos normalizados mediante a Logaritmos neperianos
XRF_train, XRF_test, YRF_train, YRF_test = train_test_split(XRF_normalizada, YRF, test_size=0.2, random_state=1)
```

Nota. La imagen muestra un ejemplo de los comentarios dejados a lo largo del código fuente para mayor entendimiento del lector.

De igual manera el proyecto está debidamente documentado para comprender razones, objetivos, importancia, desarrollo y conclusiones del presente desarrollo.

Codificación de los algoritmos

Descripción de los algoritmos utilizados

Se utilizaron 3 algoritmos principales, los cuales son Random Forest, Naive Bayes y Support Vector Machines, lo cuales fueron debidamente entrenados y adaptados al código fuente del aplicativo.

Figura 9

Algoritmo Random Forest implementado en el aplicativo

```
#Random forest
rf= RandomForestClassifier(n_estimators=1000,criterion='gini',max_depth=1000,min_samples_split=3,min_samples_leaf=1)
# Ajustar parametros de regresión lineal de datos
rf.fit(XRF_train,YRF_train)

YRF_prdss = rf.predict(XRF_test)

#Indice de ocurrencia
ocurrenciaRF=accuracy_score(YRF_test, YRF_prdss)
# Indice de f1_score
f1_RF=f1_score(YRF_test, YRF_prdss)
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
print(f1_RF)
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

# Se crea un dataframe para comparar con los valores reales
rtaRF = pd.DataFrame({'real': YRF_test,'predicciones': YRF_prdss})
lista=[YRF_test]
```

Nota. La imagen muestra la codificación del algoritmo de Random Forest

Figura 10

Algoritmo Support Vector Machines implementado en el aplicativo

```
# Support Vector Machines
datos=pd.read_csv('./src/ArchivoCSV_Mongo/filename.csv', header=None, skiprows=1)

# Separa Los datos en "XWMS" y en "YWWS"
# "XWMS" representa todas las columnas menos la ultima La cual es La de respuesta
# "YWWS" representa La columna de respuesta
XWMS = datos.iloc[:, :-1]
YWWS = datos.iloc[:, -1]

# División de los datos en un 80% para entrenamiento y un 20% para prueba
XWMS_train,XWMS_test, YWWS_train, YWWS_test = train_test_split(XWMS, YWWS, test_size=0.2, random_state=42)

clf = SVC(kernel = 'linear').fit(XWMS_train, YWWS_train)
YWWS_pred = clf.predict(XWMS_test)
ocurrenciaMVS=clf.score(XWMS_test, YWWS_test)
# Indice de f1_score
f1_MVS = f1_score(YWWS_test, YWWS_pred)
print("????????????????????????????????????????????????????????????????????????????????")
print(f1_MVS)
print("????????????????????????????????????????????????????????????????????????????????")

YWWS_pred = clf.predict(XWMS_test)

# Calculo de la matriz de correlación
correlation_matrix = datos.corr()

RTAWMS = pd.DataFrame({'Real': YWWS_test,'Predicho': YWWS_pred})
```

Nota. La imagen muestra la codificación del algoritmo de Support Vector Machines

Figura 11

Algoritmo Naive Bayes implementado en el aplicativo

```
# Modelo Naive Bayes

# Separa Los datos en "XW" y en "Y"
# "XW" representa todas las columnas menos la ultima La cual es La de respuesta
# "Y" representa La columna de respuesta

df_data = pd.read_csv('./src/ArchivoCSV_Mongo/filename.csv', header=None, skiprows=1)

# Haremos Feature Selection para mejorar Los resultados del algoritmo. Utilizando La clase SelectKBest para seleccionar Las mejores Caracteristicas

XW = df_data.iloc[:, :-1]
y = df_data.iloc[:, -1]

best = SelectKBest(k=4)
XW_new = best.fit_transform(XW, y)
XW_new.shape
selected = best.get_support(indices=True)
used_features = XW.columns[selected]

# División de los datos en conjuntos de prueba y entrenamiento, 80% para entrenamiento y 20% para prueba

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 6)
XW_train, XW_test, y_train, y_test = train_test_split(XW, y, test_size = 0.2, random_state = 6)

# Normalizacion de La data

bayes_naive = GaussianNB()
bayes_clasif = bayes_naive.fit(XW_train[used_features].values, y_train)
y_pred = bayes_naive.predict(XW_test[used_features])
My_pred = bayes_clasif.predict(X_test)

#Indice de ocurrencia
ocurrenciaWV=accuracy_score(y_test, y_pred)

f1_NB=f1_score(y_test, y_pred)
print("????????????????????????????????????????????????????????????????????????????????????????????")
print(f1_NB)
print("????????????????????????????????????????????????????????????????????????????????????????????")

RTAINW = pd.DataFrame({'Real': y_test, 'Predicho': y_pred})
```

Nota. La imagen muestra la codificación del algoritmo de Naive Bayes.

Seguido de esto de implemento la creación del algoritmo basado en la comparación de los tres anteriores, el cual más adelante fue denominado modelo MCP.

Figura 12

Fragmento de código de la creación del modelo MCP

```
lupo.append(1)
# RF=1 MVS=1 NV=0
elif(ocurrenciaRF>ocurrenciaAdimitica and ocurrenciaMVS>ocurrenciaAdimitica and ocurrenciaNV<ocurrenciaAdimitica):
    if(YRF_prdss[ni]==1 and YMVS_pred[ni]==1 and y_pred[ni]==1):
        lupo.append(1)
    elif(YRF_prdss[ni]==0 and YMVS_pred[ni]==1 and y_pred[ni]==1):
        if(ocurrenciaRF>ocurrenciaMVS):
            lupo.append(0)
        else:
            lupo.append(1)
    elif(YRF_prdss[ni]==1 and YMVS_pred[ni]==0 and y_pred[ni]==1):
        if(ocurrenciaRF>ocurrenciaMVS):
            lupo.append(1)
        else:
            lupo.append(0)
    elif(YRF_prdss[ni]==1 and YMVS_pred[ni]==1 and y_pred[ni]==0):
        lupo.append(1)
    elif(YRF_prdss[ni]==0 and YMVS_pred[ni]==0 and y_pred[ni]==0):
        lupo.append(0)
    elif(YRF_prdss[ni]==1 and YMVS_pred[ni]==0 and y_pred[ni]==0):
        if(ocurrenciaRF>ocurrenciaMVS):
            lupo.append(1)
        else:
            lupo.append(0)
    elif(YRF_prdss[ni]==0 and YMVS_pred[ni]==1 and y_pred[ni]==0):
        if(ocurrenciaRF>ocurrenciaMVS):
            lupo.append(0)
        else:
            lupo.append(1)
    elif(YRF_prdss[ni]==0 and YMVS_pred[ni]==0 and y_pred[ni]==1):
        lupo.append(0)
    else:
        print("no concuerda")
```

Nota. La imagen muestra un fragmento del código utilizado para la creación del modelo MCP

De esta manera fue posible obtener los resultados de predicciones lanzados por los tres modelos desarrollados en un proceso de algoritmia.

Desarrollo de la interfaz de usuario

Para el desarrollo de la interfaz del aplicativo se hizo uso de herramientas como Flask, Bootstrap, HTML, CSS y el mismo Python para la creación de cada ruta que contiene los tres módulos mencionados anteriormente.

Creación de rutas para los módulos

La creación de dichas rutas se realiza mediante el código “@app.route('/')”, como se puede visualizar en la figura 13.

Figura 13

Creación de la ruta prueba para el módulo de probar modelo

```
# Ruta para la pantalla de probar modelo
@app.route('/prueba')
def prueba():
    return render_template('prueba.html')
# Funcion para recibir el archivo y enviarlo a una carpeta llamada
# Archivos csv creada en la carpeta src
# Aqui se hara todo el proceso para leer el archivo seleccionado y aplicar los modelos
@app.route('/RTA', methods=['POST'])
def procesoAlgoritmo():
    if request.method == 'POST':
        # Obtenemos el archivo del input "archivo"
        f = request.files['archivo']
        filename = secure_filename(f.filename)
        # Guardamos el archivo en el directorio "Archivos PDF"
        f.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

    # test_size- valor de entrenamiento
    traine=int(request.form['test_size'])
    trainee=trainee*0.01
```

Nota. La imagen muestra la creación de la ruta para probar modelo.

De esta manera fue posible crear las rutas necesarias y realizar la configuración que cada una de ellas requería para su correcto funcionamiento.

Diseño y estructura de la interfaz

Para el diseño de la interfaz se utilizó esencialmente HTML y CSS, pero también fueron implementadas librerías de Bootstrap y tablas para mejorar el aspecto del aplicativo.

Figura 14

Fragmento de código en HTML del módulo principal

```
<!--Cuerpo del index-->
<div class="contenedor">
  <div class="area_1">
    <article>
      <section class="zona_1">
        <h3 class="titulo">Predicción de la ultima cadena de datos ingresados</h3>
        <br>
        <h4>Las predicciones de la fecha: {{Predicciones.Fecha}}</h4>
        <h4>En la hora: {{Predicciones.Hora}}</h4>
        <h4>Dicen que:</h4>
        <h1 class="porcentaje">{{Predicciones.lluvia}}</h1>
        <br>
        
      </section>
    </article>
  </div>
  <div class="area_2">
    <article class="zona_2">
      <h3 class="titulo">Actualización de predicciones segun el modelo:</h3>
      <div id="carouselExampleAutoplaying" class="carousel carousel-dark slide" data-bs-ride="carousel">
        <div class="carousel-inner">
          <div class="carousel-item active">
            <section class="area_model">
              <h3 class="modelo">Random Forest</h3>
              <abbr title="La exactitud mide cuántas predicciones acertadas realiza el modelo en comparación con el"
                <h4>Exactitud de las Predicciones (Accuracy)</h4>
              </abbr>
              <p class="porcentaje">{{ Predicciones.PorcentajeRF * 100 }}%</p>
              <abbr title="El F1 Score combina la precisión (cuántas predicciones son correctas) y la recuperación"
                <h4>Puntuación de Precisión y Recuperación (F1 Score)</h4>
              </abbr>
              <p class="porcentaje">{{ Predicciones.F1_RF * 100 }}%</p>
            </section>
          </div>
        </div>
      </div>
    </article>
  </div>
</div>
```

Nota. La imagen muestra un fragmento de código del módulo principal del aplicativo MCP

Para la mejora de los estilos se utilizó CSS.

Figura 15 Fragmento de código en CSS de los estilos del módulo principal

```
.subtitle{
  text-align: center;
}

.area_1, .area_2 {
  background-image: url('https://images.unsplash.com/photo-1517685352821-92');
  background-size: cover;
  background-attachment: fixed;
  flex-basis: calc(50% - 10px);
  padding: 0;
  background-color: #252525;
  margin-bottom: 20px;
  box-sizing: border-box;
  box-shadow: 0 0 5px #000;
  border-radius: 5px;
  color: #fff;
}

.porcentaje {
  font-size: 36px;
  color: #ffffff;
  font-weight: bold;
  text-shadow: 2px 2px 6px #000;
  margin: 10px 0;
  transition: transform 0.2s;
  cursor: pointer;
}

.porcentaje:hover {
  transform: scale(1.1);
}

.modelo{
  font-size: 30px;
  color: #ffffff;
  font-weight: bold;
  text-shadow: 2px 2px 6px #000;
}
```

Nota. La imagen muestra un fragmento de código de los estilos del módulo principal del aplicativo MCP.



Así, fue posible el desarrollo del aplicativo, combinando la unión del Backend que contiene el funcionamiento de los algoritmos para el resultado en forma de modelos con el Frontend, el cual está basado en brindar una experiencia de usuario idónea a la hora de consumir los servicios que ofrece el aplicativo.

Preguntas Frecuentes (FAQ)

1) La aplicación no se inicia correctamente ¿Cuáles podrían ser las causas?

Respuesta:

- Verifica que todas las dependencias estén instaladas.
- Asegúrate de que la configuración de la base de datos sea la correcta.
- Visualiza los mensajes de error lanzados por la terminal o el aplicativo para obtener soluciones adicionales.

2. El entorno virtual del aplicativo me da error ¿Cómo puedo solucionarlo?

Respuesta: Asegúrate de instalar el entorno virtual del proyecto en tu máquina, la carpeta de src puede copiarla y pegarla dentro de tu proyecto, de esta manera el aplicativo funcionara correctamente.

3. ¿Cómo puedo contribuir al desarrollo del proyecto?

Respuesta: Para contribuir con el proyecto puedes comunicarte con los desarrolladores Luis Rincón o Jeison Polanco, de esta manera podrás crear tu rama en el repositorio de GitHub y realizar contribuciones propias.



Glosario

MCP: Modelo Comparativo Predictivo de Precipitaciones.

Porcentaje de prueba: Cantidad de datos que se desea dar a los modelos para el entrenamiento de los datos.

Cadena de datos: Lista de elementos dados a el aplicativo para la constante actualización de las predicciones.

RF: Random Forest.

SVM: Support Vector Machine.

NB: Naive Bayes.

Po: Presión Atmosférica.

T: Temperatura.

U: Humedad Relativa.

Ff: Velocidad del viento.

RRR: Precipitación.

VSCode: Visual Studio Code



Referencias

Estación meteorológica de la Universidad de Cundinamarca Seccional Ubaté. (s. f.). [Conjunto de datos].

rp5.ru. (2023, 15 julio). *rp5.ru.* <https://rp5.ru/>

Python. (2023). Python.org. <https://www.python.org/>

Mark Otto Jacob Thornton, And Bootstrap. (s. f.). *Bootstrap.* [getbootstrap.com.](https://getbootstrap.com/)
<https://getbootstrap.com/>

Flask Documentation. (s. f.). *flask.palletsprojects.com.* <https://flask.palletsprojects.com/en/3.0.x/>

Polanco Montaña, J. S., & Rincón Gordo, L. C. (2023). *Desarrollo de un prototipo de algoritmo para la predicción de precipitaciones a partir de datos estadísticos recolectados de la estación meteorológica de la Universidad De Cundinamarca Seccional Ubaté* [Tesis de grado]. Universidad de Cundinamarca.