



UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732



Acreditación Institucional
Internacional

OTORGADA POR EL IAC CINDA ACUERDO 55 DEL 9 DE MAYO-VIGENCIA 5 AÑOS





UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

Faculty: systems engineer

Course: Deep Learning

Topic: Embed a model of machine learning in a web (cloud)

Professor: Luis Fernando Castellanos Guarín

Email: Luis.castellanosg@usantoto.edu.co

Phone: 321-4582098



CONTENIDO

- **Desarrollar una aplicación web mediante el uso de framework “*web flask*”**
- **Implementar una aplicación de aprendizaje automático en un servidor web www.pythonanywhere.com.**

¡Siempre
hacia lo alto!





Flask

web development,
one drop at a time

Framework Flask

Características:

- Es un microframework (núcleo pequeño, pero puede expandirse usando librerías).
- Curva de aprendizaje baja en comparación con otros frameworks en Python (**Django**)
- Escrito en **Python** (proporciona una interfaz cómoda para usar código Python)
- Creado en el 2010 (un framework medianamente maduro <10 años)
- Usado por plataformas web como LinkedIn y Pinterest.



Servidores públicos para proyectos de Python

<https://www.pythonanywhere.com/>

ó

<https://www.heroku.com/>

¡Siempre
hacia lo alto!



Dashboard de <https://www.pythonanywhere.com/>

pythonanywhere

Dashboard Consoles Files Web Tasks Databases

Welcome, [LuisFernandoCastellanosGuarin](#)

CPU Usage: 0% used – 0.00s of 100s. Resets in 4 hours, 47 minutes [More Info](#)

File storage: 0% full – 100.0 KB of your 512.0 MB quota

Recent Consoles [+](#) [5](#) [-](#)

[Python3.7 console 15525697](#)

New console:

[\\$ Bash](#) [>>> Python](#) [More...](#)

Recent Files [+](#) [5](#) [-](#)

You have no recently edited files.

[+ Open another file](#) [Browse files](#)

Recent Notebooks [+](#) [5](#) [-](#)

Your account does not support Jupyter Notebooks. [Upgrade your account](#) to get access!

All Web apps

You don't have any web apps.

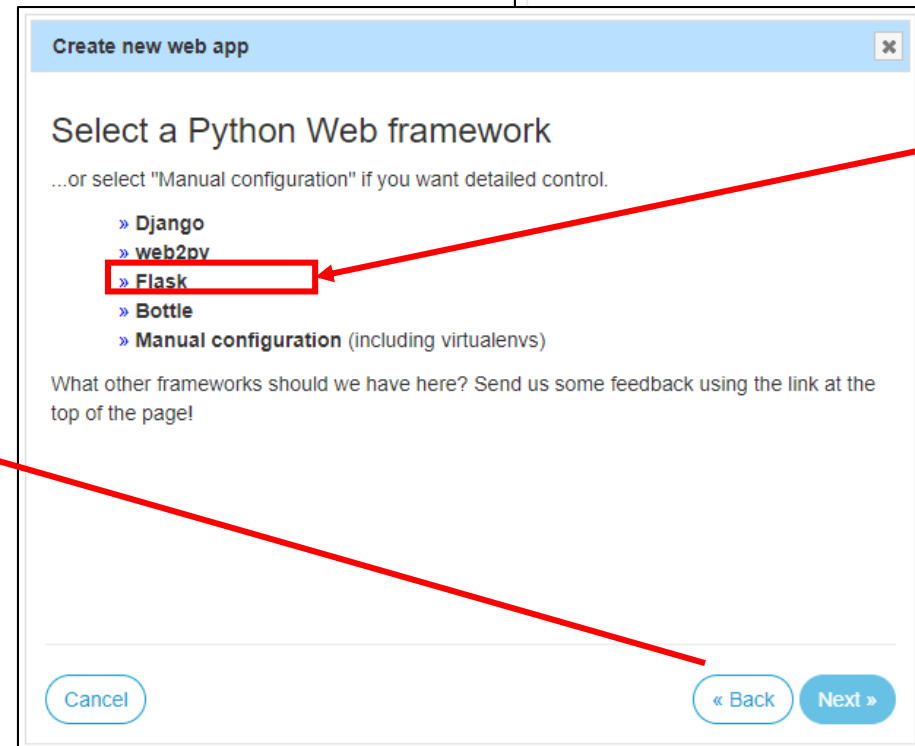
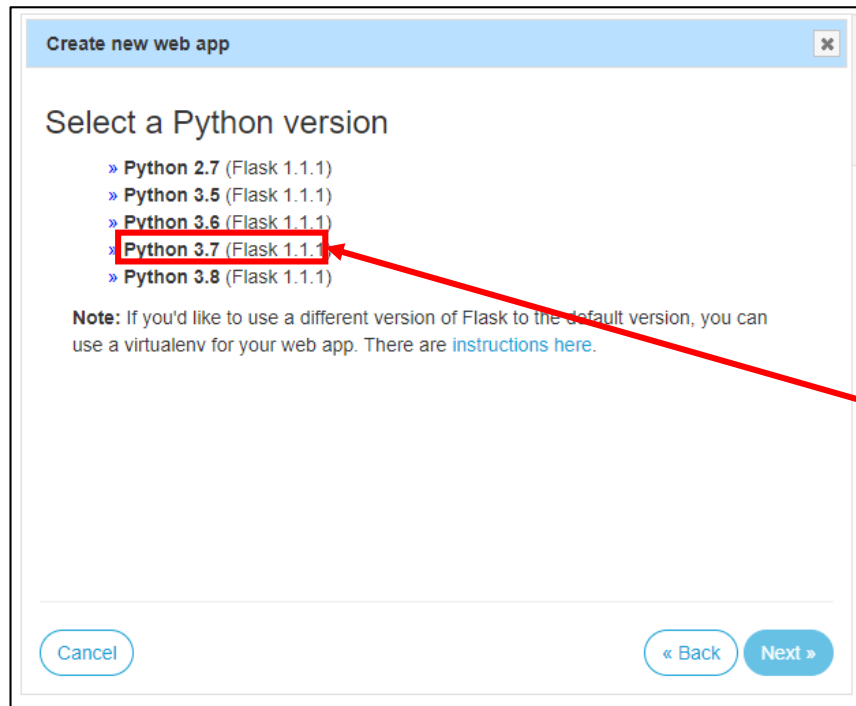
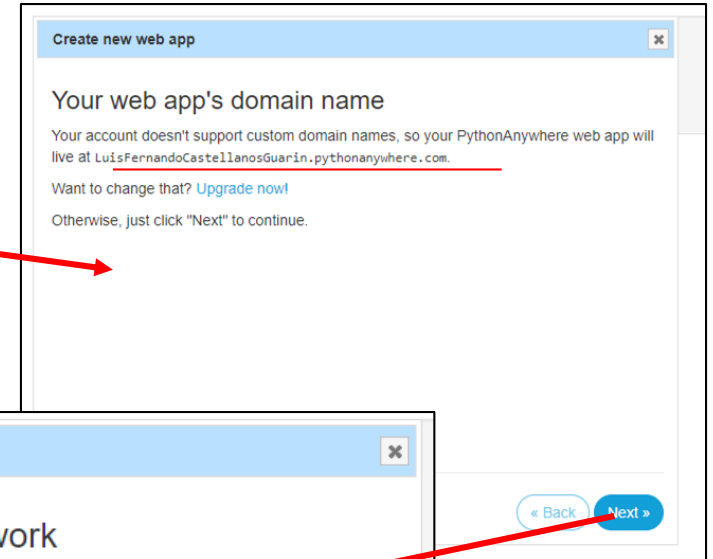
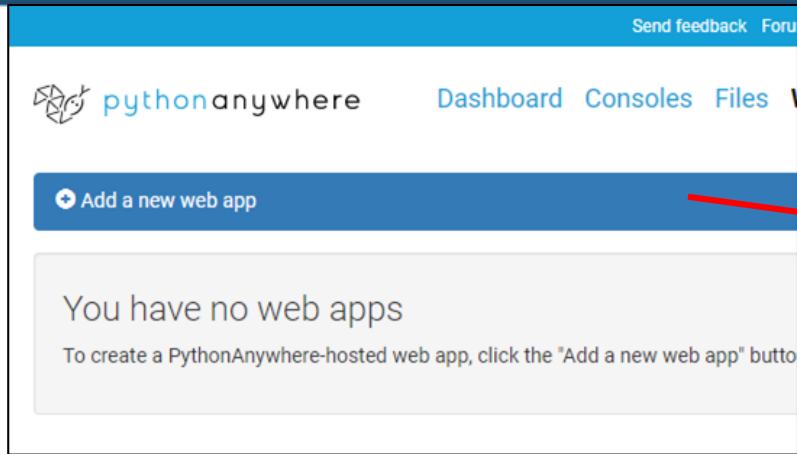
[Open Web tab](#)

Para crear una web app

¡Siempre
hacia lo alto!



Creando una nueva web apps





P1: Instalando flask en Raspberry PI

Python trae instalado Python 2.x y 3.x por defecto, por lo tanto para instalar flask:

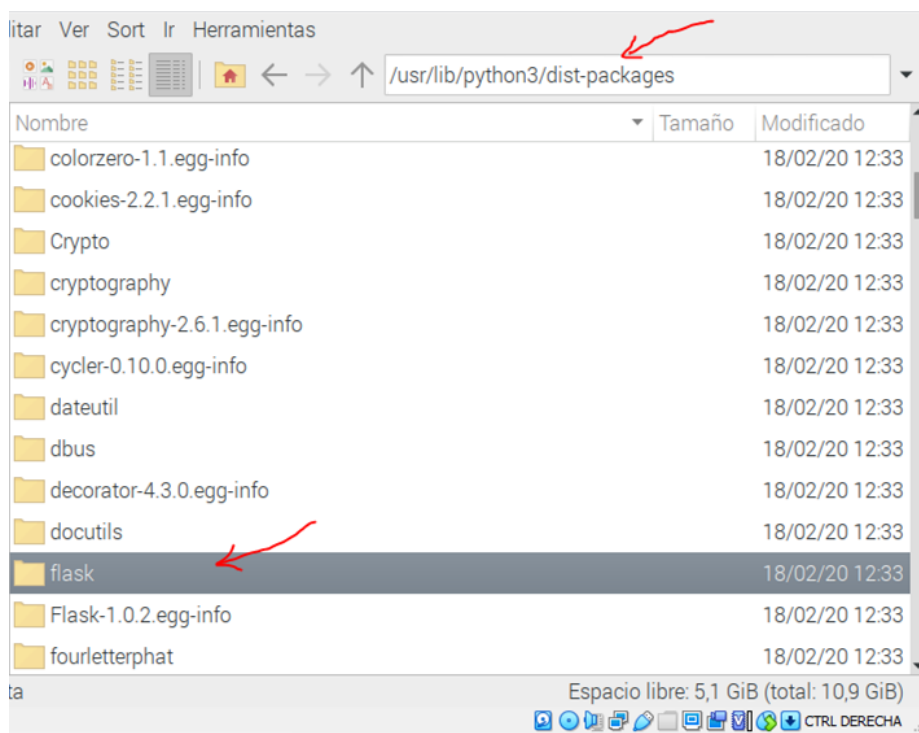
Ejecutamos el siguiente código desde la consola:

pip3 install flask

Verificamos en la ruta del S.O:

Usr/lib/python3/dist-packages

que se haya creado la carpeta denominada **flask**



iSiempre
hacia lo alto!



P2: Creando la primera aplicación en flask

Creamos un directorio en el **Home** una carpeta denominada **framework_flask**:

```
flask_app_1/  
  app.py  
  templates/  
    first_app.html
```

En el archivo **app.py**, contendrá el código principal que será ejecutado por el interprete de Python para iniciar la aplicación web de **flask**.

```
from flask import Flask, render_template  
  
app = Flask(__name__)  
@app.route('/')  
def index():  
    return render_template('first_app.html')  
  
if __name__ == '__main__':  
    app.run(debug=True)
```

En el directorio **templates** será donde **flask** buscara los archivos **HTML** estáticos para reproducir en el navegador web.

¡Siempre
hacia lo alto!



P2.1: Conociendo el app.py

- Iniciamos una nueva instancia de flask con el argumento **__name__** para que flask pueda saber que puede encontrar la carpeta de plantillas HTML (**templates**) en el mismo directorio que se encuentra.
- Usamos el definidor de ruta (**@app.route('/')**) para especificar la URL que debería desencadenar la ejecución de la función **index**.
- La función **index**, lo único que hará es renderizar el archivo HTML **first_app.html**, que se encuentra en la carpeta **templates**.
- La función **run** solo para iniciar la aplicación en el servidor donde este script es directamente ejecutado por el interprete de Python, que se garantiza mediante la sentencia **if** con **__name__=='__main__'**

```
from flask import Flask, render_template

app = Flask(__name__)
@app.route('/')
def index():
    return render_template('first_app.html')

if __name__ == '__main__':
    app.run(debug=True)
```




P2.2: creando el first_app.html

Código HTML básico y sencillo de entender.

first_app.html

```
<!doctype html>

<html>

  <head>
    <title>USTA-First app flask</title>
  </head>

  <body>
    <div>Hi, this course of Machine learning, this is my first Flask web app!    <hr>
      created by: xxxxxxxxxx  </div>
  </body>

</html>
```



P4.1: Mejorando la aplicación (formularios)

Necesitaremos la librería **WTForm** (<https://wtforms.readthedocs.io/en/latest/>).
Nos permitirá usar formularios en nuestras aplicaciones web de **flask**:

What's your name?

Hello Sebastian

¡Siempre
hacia lo alto!



P4.1: Mejorando la aplicación (formularios)

La librería **WTFORM** la instalaremos ejecutando el siguiente código desde la consola/terminal :

```
Sudo pip3 install wtforms
```

```
pi@raspberrypi:/media/cdrom0 $ pip3 install wtforms
Collecting wtforms
  Downloading https://files.pythonhosted.org/packages/9f/c8/dac5dce9908df1d9d48e
c0e26e2a250839fa36ea2c602cc4f85ccfeb5c65/WTForms-2.2.1-py2.py3-none-any.whl (166
kB)
    100% |#####| 174kB 678kB/s
Installing collected packages: wtforms
Successfully installed wtforms-2.2.1
pi@raspberrypi:/media/cdrom0 $
```

En la carpeta compartida de drive tenemos una subcarpeta denominada **flask_app_2**, con la siguiente distribución de carpetas y archivos:

```
flask_app_2/
  app.py
  static/
    styles.css
  templates/
    _formhelpers.html
    second_app.html
    hello.html
```

¡Siempre
hacia lo alto!



P4.1: Mejorando la aplicación (formularios)

Nueva versión del **app.py** que nos permitirá que las dos paginas HTML interactúen (**second_page.html** y **hello.html**) usando el método **POST** y por ultimo usamos el argumento **debug=true**, muy útil para desarrollar aplicaciones web.

```
from flask import Flask, render_template, request
from wtforms import Form, TextAreaField, validators

app = Flask(__name__)

class HelloForm(Form):
    sayhello = TextAreaField("",[validators.DataRequired()])

@app.route('/')

def index():
    form = HelloForm(request.form)
    return render_template('first_app.html', form=form)

@app.route('/hello', methods=['POST'])
def hello():
    form = HelloForm(request.form)
    if request.method == 'POST' and form.validate():
        name = request.form['sayhello']
        return render_template('hello.html', name=name)
    return render_template('first_app.html', form=form)

if __name__ == '__main__':
    app.run(debug=True)
```

¡Siempre
hacia lo alto!



P4.2: implementar macro del motor JINJA2.

Jinja 2 es un motor de plantillas para Python:

Permite al desarrollador producir páginas web, que contienen, por ejemplo, código html base y marcadores de posición para que Jinja 2 los llene.

Basado en el sistema de plantillas de **Django**, **Jinja** es uno de los más utilizados, ya que permite a los desarrolladores usar conceptos poderosos como **sandboxing** y **herencia** para permitir que una plantilla se reutilice fácilmente.

Para instalar el motor de pantillas ejecute el siguiente código desde el terminal/consola del S.O:

```
sudo pip3 install jinja2
```

```
pi@raspberrypi: /  
File Edit Tabs Help  
pi@raspberrypi:~$ pip3 install jinja2  
Collecting jinja2  
  Downloading https://files.pythonhosted.org/packages/65/e0/eb35e762802015cab1cc  
ee04e8a277b03f1d8e53da3ec3106882ec42558b/Jinja2-2.10.3-py2.py3-none-any.whl (125  
kB)  
    100% |#####| 133kB 662kB/s  
Collecting MarkupSafe>=0.23 (from jinja2)  
  Downloading https://files.pythonhosted.org/packages/b1/60/fa4afa6fb4547b46b24b  
c679dd312242e0e579b4ee5651a2e5f50f814319/MarkupSafe-1.1.1-cp35-cp35m-manylinux1_  
i686.whl  
Installing collected packages: MarkupSafe, jinja2  
Successfully installed MarkupSafe-1.1.1 Jinja2-2.10.3  
pi@raspberrypi:~$
```

¡Siempre
hacia lo alto!



P4.2: implementar macro del motor JINJA2.

Jinja es simple:

Tienes una plantilla con un montón de agujeros en ella. Luego, le pide al motor que llene la plantilla con los valores que le da en el tiempo de ejecución, y se le devuelve la respuesta, en forma de un documento **html**, listo para ser enviado al usuario

En la subcarpeta templates creamos el archivo HTML, denominado: **_formhelpers.html**, que importaremos en el archivo **second_app.html** para renderizar el campo del texto **textAreaFields**

```
{% macro render_field(field) %}
<dt>{{ field.label }}
<dd>{{ field(**kwargs)|safe }}
{% if field.errors %}
<ul class=errors>
{% for error in field.errors %}
<li>{{ error }}</li>
{% endfor %}
</ul>
{% endif %}
</dd>
</dt>
{% endmacro %}
```

¡Siempre
hacia lo alto!



P4.2: implementar macro del motor JINJA2.

Implementaremos una hoja de estilo en cascada (CSS) **style.css**, para mejorar la apariencia de los archivos HTML.

```
body {  
    font-size: 2em;  
}
```

El código HTML de **second_app.html**.

```
<!doctype html>  
<html>  
  <head>  
    <title>USTA -Second app</title>  
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">  
  </head>  
  <body>  
{% from "_formhelpers.html" import render_field %}  
<div>USTA - Faculty: systems engineering</div>  
<div>What's your name?</div>  
<form method=post action="/hello">  
  <table>  
    <tr>  
      <td>  
        {{ render_field(form.sayhello) }}  
      </td>  
      <td>  
        <input type=submit value='Say Hello' name='submit_btn'>  
      </td>  
    </tr>  
  </table>  
</form>  
</body>  
</html>
```

¡Siempre
hacia lo alto!



P4.3: Pagina resultante

Por ultimo creamos el archivo **hello.html** que será renderizado mediante el retorno de línea **render_template('hello.html', name=name)** dentro de la función **hello** que se definio en el script app.py

hello.html

```
<!doctype html>
<html>
  <head>
    <title>USTA - Second app</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>
    <div>
      Hello {{ name }}, welcome to course of machine learning
      <hr>
      Created by: xxxxx
    </div>
  </body>
</html>
```



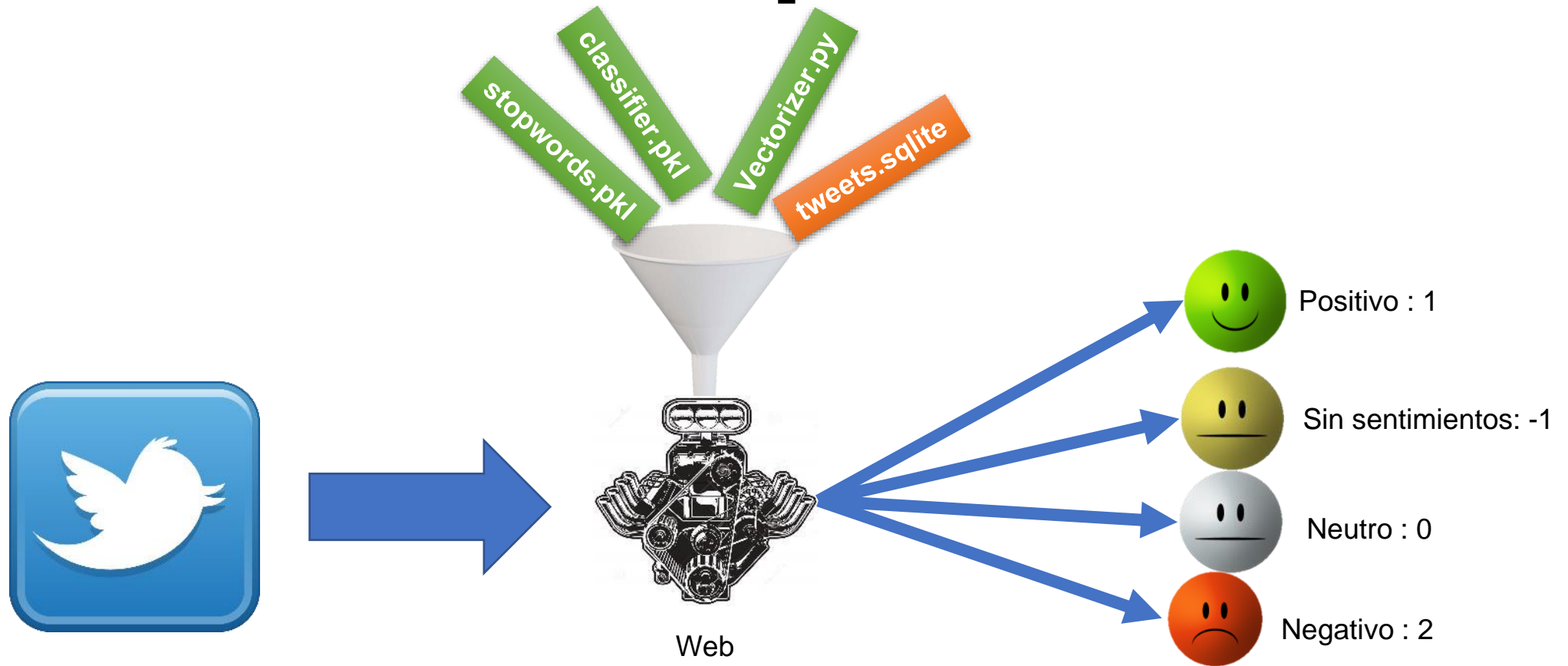
UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
T U N J A
VIGILADA MINEDUCACIÓN - SNIES 1732



Formando personas que transforman



Convertir el clasificador de twitter en una aplicación web



Crearemos una aplicación web que validaremos **tweets en español** y la aplicación mostrara la etiqueta de la clase predicha y la probabilidad de la predicción. Adicional el usuario podrá proporcionar un comentario sobre dicha predicción con dos opciones (Correcta o incorrecta).



P1: Configurar database SQLite

Creamos una base de datos **SQLite** llamada **tweets.sqlite** dentro del directorio de **twitterclassifier** y almacenara dos criticas de cine (prueba)

make_database_sqlite.py

```
import sqlite3
import os

if os.path.exists('tweets.sqlite'):
    os.remove('tweets.sqlite')

conn = sqlite3.connect('tweets.sqlite')
c = conn.cursor()
c.execute('CREATE TABLE tweets_db (tweet TEXT, sentiment INTEGER, date TEXT)')

example1 = 'que aburrido es estar en cuarentena...'
c.execute("INSERT INTO tweets_db (tweet, sentiment, date) VALUES (?, ?, DATETIME('now'))", (example1, 2))

example2 = 'Estoy feliz de estar con mi familia'
c.execute("INSERT INTO tweets_db (tweet, sentiment, date) VALUES (?, ?, DATETIME('now'))", (example2, 1))

conn.commit()
conn.close()
```





P1: Configurar database SQLite

Comprobemos si se guardo bien: abrimos la conexión y hacemos una consulta (SELECT)

```
conn = sqlite3.connect(tweets.sqlite')
c = conn.cursor()

c.execute("SELECT * FROM tweets_db WHERE date BETWEEN '2019-01-01 10:10:10' AND DATETIME('now')")
results = c.fetchall()
conn.close()

print(results)
```

```
[
  (que aburrido es estar en cuarentena... ', 1, '2019-10-08 22:01:44'),
  (Estoy feliz de estar con mi familia ', 0, '2019-10-08 22:01:44')
]
```

Descargar los 4 archivos (classifier.pkl, stopwords.pkl, vectorizer.py y tweets.sqlite) y guardarlos en el repositorio denominado:

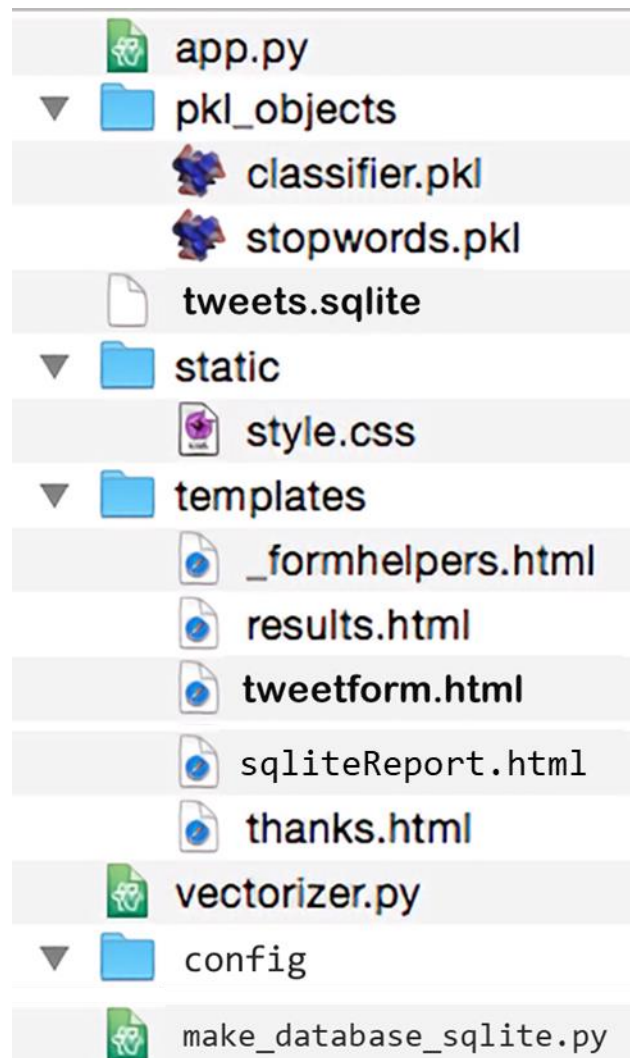
twitterclassifier

**iSiempre
hacia lo alto!**



P2: Archivos y carpetas

Árbol de directorios y archivos que se van a crear:



Los archivos están en la carpeta drive/.../documentos/

framework Flask/tweetsclassifier:

- **pk1_objects (classifier.pkl y stopwords.pkl)**
- **tweets.sqlite**
- **vectorizer.py**

Los demás archivos los vamos a analizar a continuación.

¡Siempre
hacia lo alto!



P2: Archivos y carpetas

Librerías que vamos a necesitar:
sklearn

```
sudo pip3 install sklearn
```

```
ModuleNotFoundError: No module named 'sklearn'
pi@raspberrypi:~/framework_Flask/tweetsclassifier $ pip3 install sklearn
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting sklearn
  Downloading https://www.piwheels.org/simple/sklearn/sklearn-0.0-py2.py3-none-any.whl
Collecting scikit-learn (from sklearn)
  Downloading https://files.pythonhosted.org/packages/98/a2/3d9427aa154136e4a8131227b6ed4d1315289f487d53514f4e916b869951/scikit_learn-0.22.2.post1-cp37-cp37m-manylinux1_i686.whl (6.3MB)
    100% |#####| 6.3MB 108kB/s
Collecting scipy>=0.17.0 (from scikit-learn->sklearn)
  Downloading https://files.pythonhosted.org/packages/42/49/9c113fc85e1d6a38eb2f4bf8702310a0ed00886def039c1a9a14532196df/scipy-1.4.1-cp37-cp37m-manylinux1_i686.whl (22.6MB)
    100% |#####| 22.6MB 35kB/s
Requirement already satisfied: numpy>=1.11.0 in /usr/lib/python3/dist-packages (from scikit-learn->sklearn) (1.16.2)
Collecting joblib>=0.11 (from scikit-learn->sklearn)
  Downloading https://files.pythonhosted.org/packages/28/5c/cf6a2b65a321c4a209efcdf64c2689efae2cb62661f8f6f4bb28547cf1bf/joblib-0.14.1-py2.py3-none-any.whl (294kB)
    100% |#####| 296kB 172kB/s
Installing collected packages: scipy, joblib, scikit-learn, sklearn
Successfully installed joblib-0.14.1 scikit-learn-0.22.2.post1 scipy-1.4.1 sklearn-0.0
pi@raspberrypi:~/framework_Flask/tweetsclassifier $
```

¡Siempre
hacia lo alto!



P3. Comprendiendo el app.py (parte 1)

```
from flask import Flask, render_template, request
from wtforms import Form, TextAreaField, validators
import pickle
import sqlite3
import os
import numpy as np

# import HashingVectorizer from local dir
from vectorizer import vect

app = Flask(__name__)

##### Preparing the Classifier
cur_dir = os.path.dirname(__file__)
clf = pickle.load(open(os.path.join(cur_dir,
    'pkl_objects',
    'classifier.pkl'), 'rb'))
db = os.path.join(cur_dir, 'tweets.sqlite')

def classify(document):
    label = {-1: 'Sin sentimiento', 0: 'Neutro', 1: 'Positivo', 2: 'Negativo'}
    X = vect.transform([document])
    y = clf.predict(X)[0]
    proba = np.max(clf.predict_proba(X))
    return label[y], proba

def train(document, y):
    X = vect.transform([document])
    clf.partial_fit(X, [y])

def sqlite_entry(path, document, y):
    conn = sqlite3.connect(path)
    c = conn.cursor()
    c.execute("INSERT INTO tweets_db (tweet, sentiment, date)"
        " VALUES (?, ?, DATETIME('now'))", (document, y))
    conn.commit()
    conn.close()

def sqlite_select(path):
    conn = sqlite3.connect(path)
    c = conn.cursor()
    c.execute("SELECT tweet, sentiment, date FROM tweets_db")
    results = c.fetchall()
    return results
```

El archivo **app.py** cambia un poco, en la primera parte implementaremos todo lo que necesitamos para clasificar y entrenar:

- El **vectorizer** que trabajamos en Google Colaboratory y con ello el **HashingVectorizer** (vectoriza y tokeniza palabras en un texto).
- Deserialización del clasificador de regresión logística (classifier.pkl)
- Creamos una función llamada **classify**, que le ingresa un texto y me retorna la etiqueta de clase.
- Creamos una función **train**, que usaremos para actualizar el clasificador, con las nuevas críticas y etiquetas.
- Creamos una función **sqlite_entry**, la usaremos para guardar las críticas que cargaran los usuarios en la página web.
- Creamos una función denominada **sqlite_select**, que nos permitirá visualizar los tweets que se han clasificado y han permitido reentrenar la IA.



P3.1 Comprendiendo el app.py (parte 2)

```
#####flask
class TweetForm(Form):
    tweet = TextAreaField("",
        [validators.DataRequired(),
         validators.length(min=15)])

@app.route('/')
def index():
    form = TweetForm(request.form)
    return render_template('tweetform.html', form=form)

@app.route('/sqliteReport', methods=['POST'])
def sqliteReport():
    dataset = sqlite_select(db)
    return render_template('sqliteReport.html', dataset=dataset)

@app.route('/results', methods=['POST'])
def results():
    form = TweetForm(request.form)
    if request.method == 'POST' and form.validate():
        tweet = request.form['tweet']
        y, proba = classify(tweet)
        return render_template('results.html',
            content=tweet,
            prediction=y,
            probability=round(proba*100, 2))
    return render_template('tweetform.html', form=form)

@app.route('/thanks', methods=['POST'])
def feedback():
    feedback = request.form['feedback_button']
    tweet = request.form['tweet']
    prediction = request.form['prediction']
    inv_label = {'Sin sentimiento':-1, 'Neutro':0, 'Positivo':1, 'Negativo':2}
    y = inv_label[prediction]
    train(tweet, y)
    sqlite_entry(db, tweet, y)

    return render_template('thanks.html')

if __name__ == '__main__':
    app.run(debug=True)
```

El archivo **app.py** cambia un poco, en la segunda parte implementaremos lo asociado a el framework de **flask**

- Creamos una clase llamada **ReviewForm** que instancia el **TextAreaField**, que será renderizado en el archivo **reviewform.html** (pagina de inicio de a aplicación). Que es renderizada por la función **index**.
- Con el parámetro **validators.length(min=15)**, obligamos al usuario a que ingrese mínimo 15 caracteres.
- La función **results**, tomamos el contenido del formulario (**TextAreaField**) y lo pasamos al clasificador para predecir el sentimiento y después se mostrara en **results.html**.
- La función **feedback**, toma la apreciación correcto o incorrecto, y vuelve a transformar el sentimiento predicho y vuelve a entrenar (**train**).
- La función **sqliteReport**, que permitirá consultar la base de datos de **sqlite** y mostrar la información en pantalla



P4: formulario de tweets (tweetform.html)

Página **index** de proyecto llama a la página **html** ubicada en **templates** denominada **tweetform.html**:

```
<!doctype html>
<html>
  <head>
    <title>clasificador de sentimientos Tweet </title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>

    <h2>Ingrese el tweet (español) para evaluar el sentimiento:</h2>
    <h3>Sin sentimiento, neutro, positivo, negativo</h3>

    {% from "_formhelpers.html" import render_field %}

    <form method=post action="/results">
      <dl>
        {{ render_field(form.tweet, cols='30', rows='10') }}
      </dl>
      <div>
        <input type=submit value='Evaluar tweet' name='submit_btn'>
      </div>
    </form>

    <div id='sqlite'>
      <form method=post action="/sqliteReport">
        <input type=submit value='Reporte de clasificaciones'>
      </form>
    </div>

  </body>
</html>
```

Importamos la plantilla **_formhelpers.html** (macro de jinja2), la función **render_field** se usa para renderizar un **TextAreaField**, donde el usuario proporcionará la crítica.



Al dar clic en “**evaluar tweet**” me llevará a la página **result.html**

Siempre
hacia lo alto!



P4: pagina de resultado (result.html) parte 1

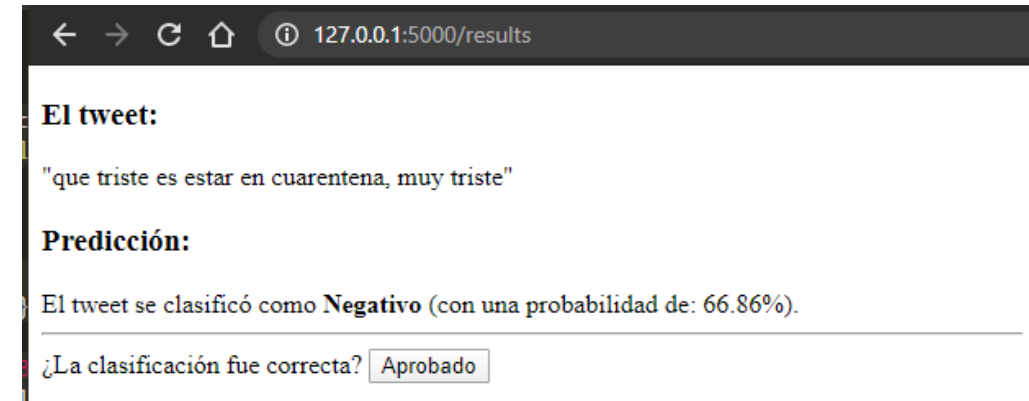
Pagina **result.html** tiene dos partes, la primera donde se muestra el resultado de la clasificación del tweet según la IA:

```
<!doctype html>
<html>
  <head>
    <title>clasificador de sentimientos Tweet</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
  </head>
  <body>

    <h3>El tweet:</h3>
    <div>"{{ content }}"</div>

    <h3>Predicción:</h3>
    <div>El tweet se clasificó como <strong>{{ prediction }}</strong>
      (con una probabilidad de: {{ probability }}%).</div>

    <hr>
```



¡Siempre
hacia lo alto!



P4: pagina de resultado (result.html) parte 2

Pagina **result.html** tiene dos partes, la segunda parte se le presenta la opción al usuario de calificar si el modelo de IA hizo bien su trabajo, de lo contrario podrá reentrenar con el apoyo del usuario.

```
<form action="/thanks" method="post">
    ¿La clasificación fue correcta?
    <input type="submit" value='Aprobado' name='feedback_button'>
    <input type="hidden" value='{{ prediction }}' name='prediction'>
    <input type="hidden" value='{{ content }}' name='tweet'>
</form>
<br>De lo contrario seleccione la opción correcta del sentimiento del tweet<br>
<form action="/thanks" method="post">
    <input type="submit" value='Sin sentimientos' name='feedback_button'><br>
    <input type="hidden" value='Sin sentimiento' name='prediction'>
    <input type="hidden" value='{{ content }}' name='tweet'>
</form>
<form action="/thanks" method="post">
    <input type="submit" value='Sentimientos neutros'
name='feedback_button'><br>
    <input type="hidden" value='Neutro' name='prediction'>
    <input type="hidden" value='{{ content }}' name='tweet'>
</form>
<form action="/thanks" method="post">
    <input type="submit" value='Sentimientos positivos'
name='feedback_button'><br>
    <input type="hidden" value='Positivo' name='prediction'>
    <input type="hidden" value='{{ content }}' name='tweet'>
</form>
<form action="/thanks" method="post">
    <input type="submit" value='Sentimientos negativos'
name='feedback_button'><br>
    <input type="hidden" value='Negativo' name='prediction'>
    <input type="hidden" value='{{ content }}' name='tweet'>
</form>
<br>
<hr>
<form action="/">
    <input type="submit" value='Evaluar otro tweet'>
</form>
</body>
</html>
```

The screenshot shows a web browser window with the URL 127.0.0.1:5000/results. The page displays the following content:

El tweet:

"que triste es estar en cuarentena, muy triste"

Predicción:

El tweet se clasificó como **Negativo** (con una probabilidad de: 66.86%).

¿La clasificación fue correcta?

De lo contrario seleccione la opción correcta del sentimiento del tweet

El usuario puede determinar que la clasificación fue errónea, y clasificar el tweet con el sentimiento "correcto".

Al momento de aprobar o reclasificar el modelo se reentrenara agregando el tweet y el sentimiento al modelo de IA y con ello evolucionara la capacidad de acertar en la clasificación.



P5: pagina de agradecimiento (thanks.html)

Pagina **thanks.html**, tiene dos botones que permitirán al usuario evaluar otro tweet o ver un reporte general de los tweets que se han clasificado en la pagina.

```
<!doctype html>
<html>
  <head>
    <title>clasificador de sentimientos Tweet</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>

<h3>Muchas gracias por su retroalimentación!</h3>

<div id='button'>
  <form action="/">
    <input type=submit value='Evaluar otro tweet'>
  </form>
</div>
<hr>
<div id='sqliteReport'>
  <form method=post action="/sqliteReport">
    <input type=submit value='Reporte de clasificaciones'>
  </form>
</div>

</body>
</html>
```



¡Siempre
hacia lo alto!



P6: reporte de sqlite (sqliteReport.html)

Pagina **sqliteReport.html**, consulta la base de datos de sqlite en la tabla tweets_db y realiza “SELECT tweet, sentiment, date FROM tweets_db” y visualiza los datos en la pagina.

```
<!doctype html>
<html>
  <head>
    <title>Sqlite clasificador de sentimientos Tweet</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  </head>
  <body>

<h3>Listado de clasificaciones realizadas</h3>
<hr>
<table border=1>
  <tr><td>Tweet</td><td>Sentimiento</td><td>Fecha</td></tr>

  {% for row in dataset %}
  <tr>
    <td>{{row[0]}}</td>
    <td>{{row[1]}}</td>
    <td>{{row[2]}}</td>

  </tr>
  {% endfor %}

</table>
<hr>

<div id='button'>
  <form action="/">
    <input type=submit value='Evaluar otro tweet'>
  </form>

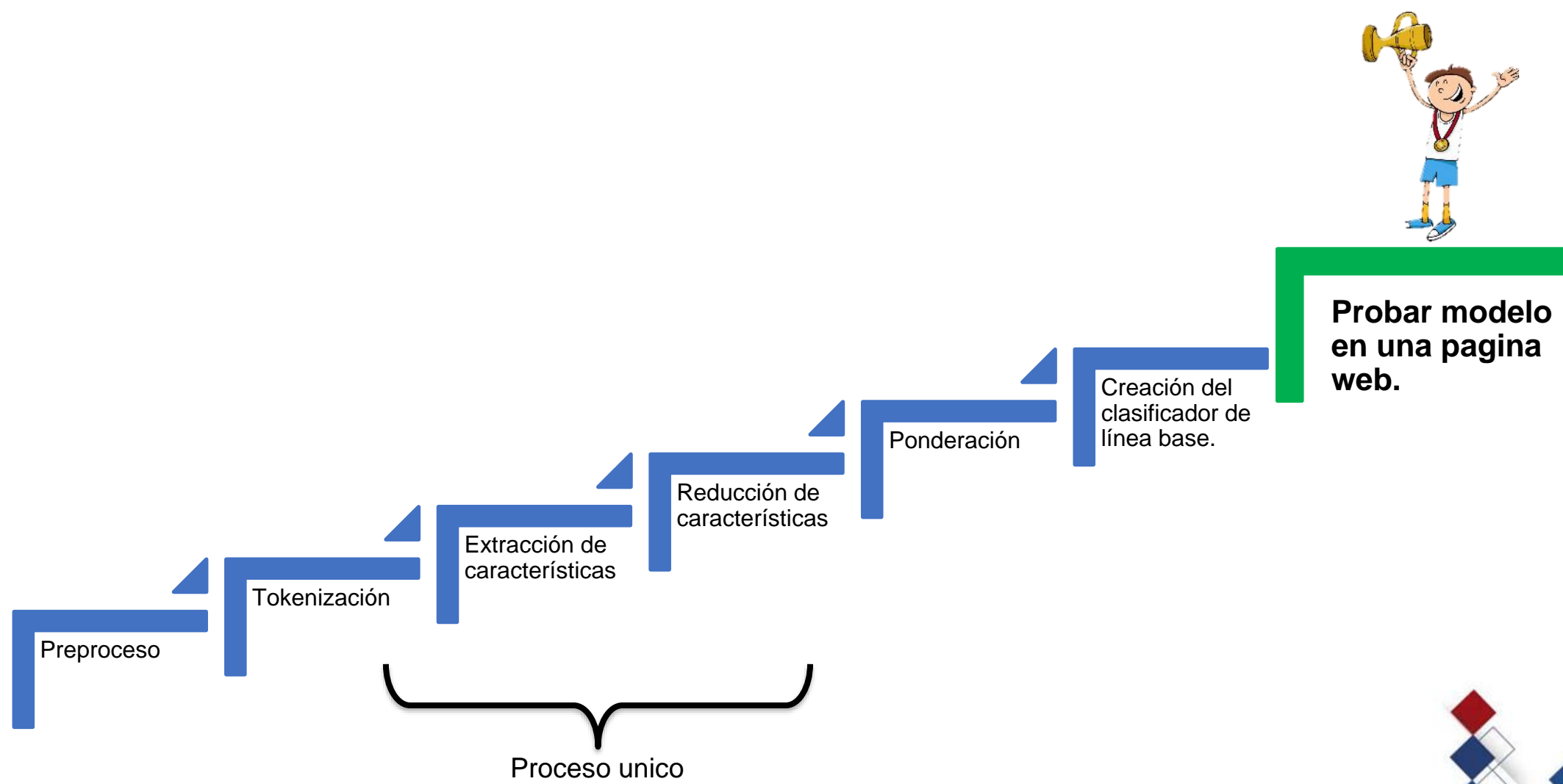
</div>
</body>
</html>
```

← → ↺ 🏠 ⓘ 127.0.0.1:5000/sqliteReport

Listado de clasificaciones realizadas

Tweet	Sentimiento	Fecha
que aburrido es estar en cuarentena...	2	2020-04-18 22:10:15
Estoy feliz de estar con mi familia	0	2020-04-18 22:10:15
probando lo positivo de vivir en cuarenta por ahora va genial	1	2020-04-19 22:16:03
probando lo positivo de vivir en cuarenta por ahora va genial	1	2020-04-19 22:17:04
que aburrido es estar en cuarentena...	2	2020-04-19 22:17:51
que aburrido es estar en cuarentena...	2	2020-04-19 22:18:20
que triste es estar en cuarentena, muy triste	2	2020-04-19 22:19:20
que triste es estar en cuarentena, muy triste	2	2020-04-19 23:14:52


iSiempre
hacia lo alto!




¡Siempre
hacia lo alto!







Subimos todos los archivos

 pythonanywhere [Dashboard](#) [Consoles](#) **Files** [Web](#) [Tasks](#) [Databases](#)

/home/LuisFernandoCastellanosGuarin/  mysite 13% full – 64.2 MB of your 512.0 MB quota
[Open Bash console here](#)












Directories

Enter new directory name [New directory](#)

- [__pycache__/](#) 
- [pkl_objects/](#) 
- [static/](#) 
- [templates/](#) 

Files

Enter new file name, eg hello.py [New file](#)

 app.py	   2020-04-21 02:33 2.5 KB
 tweets.sqlite	  2020-04-21 02:33 8.0 KB
 vectorizer.py	   2020-04-21 02:33 798 bytes

[Upload a file](#)
100MiB maximum size

Se deben cargar todos los archivos y sus respectivas carpetas.

¡Siempre
hacia lo alto!



Instalar todas las librerías por “bash console”

Este paso es mejor omitirlo para servidores pequeños como PythonAnyWhere (Solo tienes 500 megas),

SI ES SU CASO ADELANTE DOS DIAPOSITIVA

Creamos un ambiente virtual para poder instalar las librerías necesarias:

```
mkvirtualenv myvirtualenv --python=/usr/bin/python3.7
```

Para listar los ambiente virtuales creados:

```
lsvirtualenv
```

Para eliminar un ambiente virtual necesarias:

```
rmvirtualenv myvirtualenv
```

Una vez dentro del ambiente virtual: **(myvirtualenv) 04:16 ~/mysite**

```
pip install sklearn
```

```
pip install flask
```

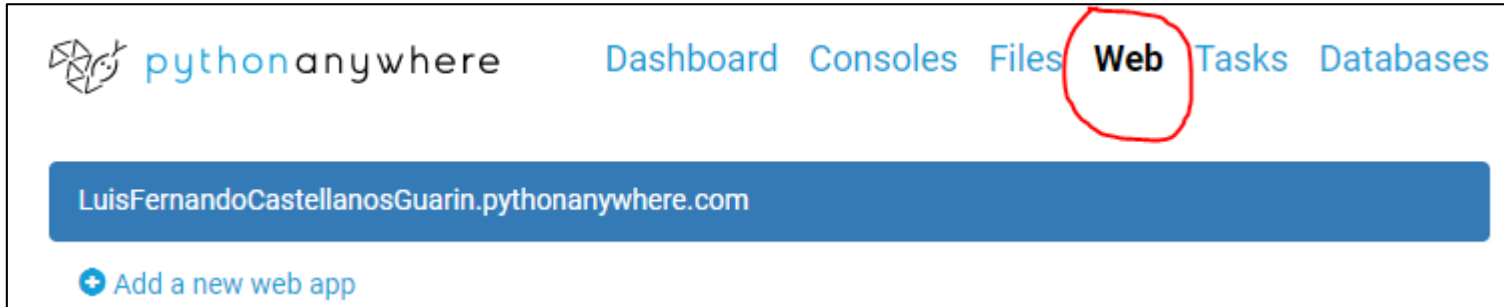
```
pip install wtforms
```

```
pip install jinja2
```



Agregar el virtualenv a la app web

Ingresamos a la app web.



Verificamos que la versión de Python sea la 3.8 y en la sección de virtualenv: Agregamos el siguiente texto:

```
/home/NOMBREDELUSUARIO/.virtualenvs/myvirtualenv
```

Donde el nombre del usuario sea el que se asignó a la plataforma PythonAnywhere.



¡Siempre
hacia lo alto!



Instalar todas las librerías por “bash console”

Si tienen problemas de espacio en disco, no usen ambientes virtuales (exclusivo para PythonAnyWhere – usuarios nuevos)

```
pip3.8 install --user sklearn
```

```
pip3.8 install --user flask
```

```
pip3.8 install --user wtforms
```

```
pip3.8 install --user jinja2
```

Recuerden las versiones que deben tener instaladas:

- sklearn => 0.22.2.post1
- Python (>= 3.5)
- NumPy (>= 1.11)
- SciPy (>= 0.17)
- Joblib (>= 0.11)

Para hacer upgrades:

```
pip3.8 install --user -U scikit-learn
```

¡Siempre
hacia lo alto!