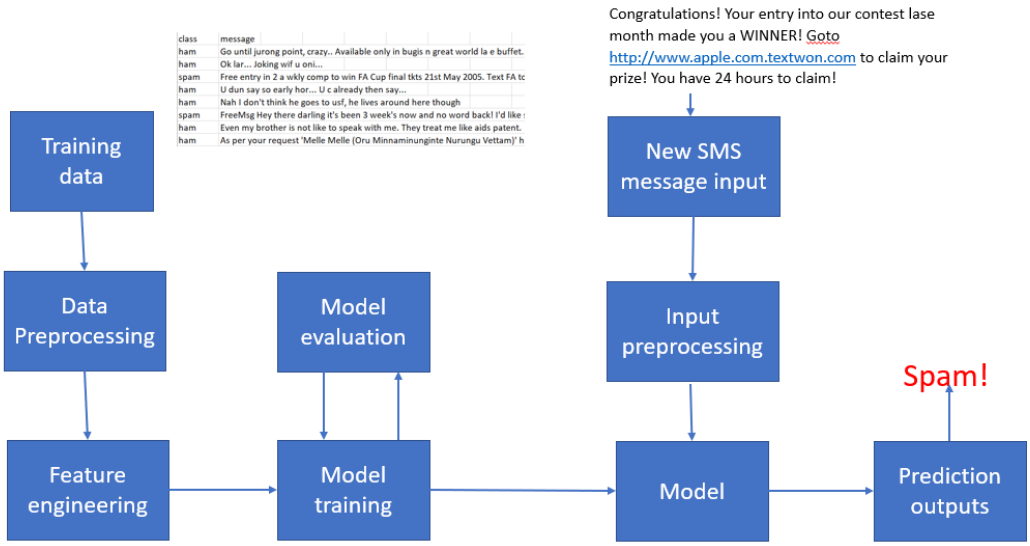


Develop a NLP Model in Python & Deploy It with Flask, Step by Step

Flask API, Document Classification, Spam Filter

Susan Li Follow

Dec 17, 2018 · 6 min read

By far, we have developed many machine learning models, generated numeric predictions on the testing data, and tested the results. And we did everything offline. In reality, generating predictions is only part of a machine learning project, although it is the most important part in my opinion.

Considering a system using machine learning to detect spam SMS text messages. Our ML systems workflow is like this: Train offline -> Make model available as a service -> Predict online.

- A classifier is trained offline with spam and non-spam messages.
- The trained model is deployed as a service to serve users.

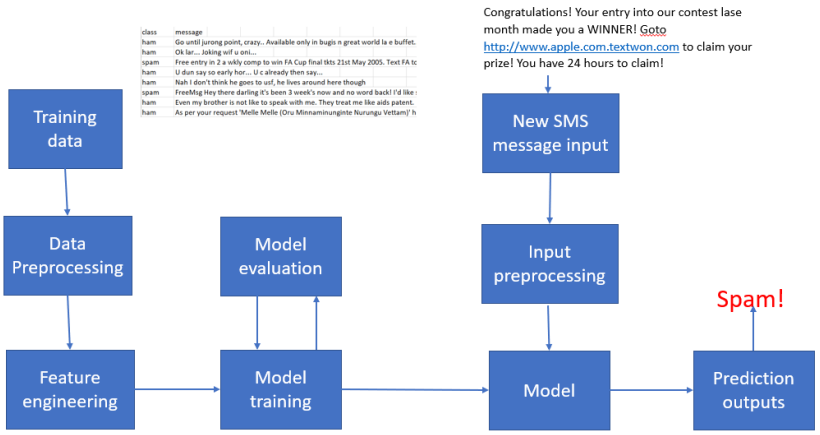


Figure 1

When we develop a machine learning model, we need to think about how to deploy it, that is, how to make this model available to other users.

Kaggle and Data science bootcamps are great for learning how to build and optimize models, but they don't teach engineers how to take them to the next step, where there's a major difference between building a model, and actually getting it ready for people to use in their products and services.

In this article, we will focus on both: building a machine learning model for spam SMS message classification, then create an API for the model, using Flask, the Python micro framework for building web applications.This API allows us to utilize the predictive capabilities through HTTP requests. Let's get started!

ML Model Building

The data is a collection of SMS messages tagged as spam or ham that can be found here. First, we will use this dataset to build a prediction model that will accurately classify which texts are spam.

Naive Bayes classifiers are a popular statistical technique of e-mail filtering. They typically use bag of words features to identify spam e-mail. Therefore, We'll build a simple message classifier using Naive Bayes theorem.

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectc
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

df = pd.read_csv('spam.csv', encoding="latin-1")
df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], ax
df['label'] = df['class'].map({'ham': 0, 'spam': 1})
X = df['message']
y = df['label']
cv = CountVectorizer()
X = cv.fit_transform(X) # Fit the Data
```

NB_spam.py

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1587
1	0.93	0.92	0.92	252
avg / total	0.98	0.98	0.98	1839

Figure 2

Not only Naive Bayes classifier is easy to implement but also provides very good result.

After training the model, it is desirable to have a way to persist the model for future use without having to retrain. To achieve this, we add the following lines to save our model as a .pkl file for the later use.

```
from sklearn.externals import joblib
joblib.dump(clf, 'NB_spam_model.pkl')
```

And we can load and use saved model later like so:

```
NB_spam_model = open('NB_spam_model.pkl','rb')
clf = joblib.load(NB_spam_model)
```

The above process called “persist model in a standard format”, that is, models are persisted in a certain format specific to the language in development.

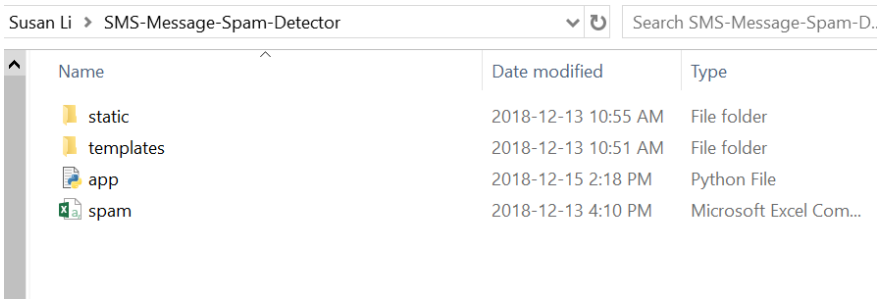
And the model will be served in a micro-service that expose endpoints to receive requests from client. This is the next step.

Turning the Spam Message Classifier into a Web Application

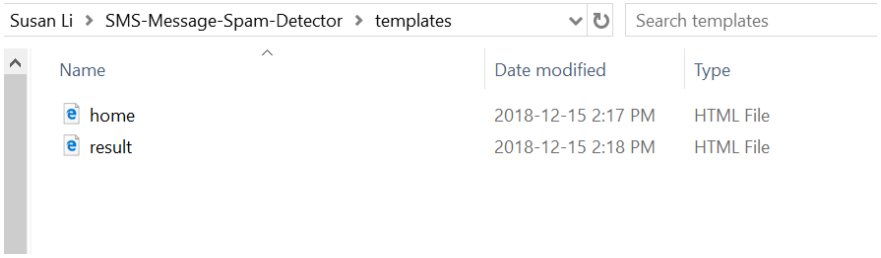
Having prepared the code for classifying SMS messages in the previous section, we will develop a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of spam or not spam.

First, we create a folder for this project called `SMS-Message-Spam-Detector` , this is the directory tree inside the folder. We will explain each file.

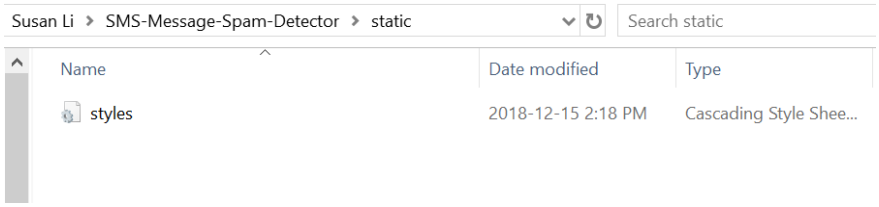
```
spam.csv
app.py
templates/
        home.html
        result.html
static/
        style.css
```



SMS-Message-Spam-Detector folder



templates folder



static folder

SMS Message Spam Detector folder

The sub-directory `templates` is the directory in which Flask will look for static HTML files for rendering in the web browser, in our case, we have two html files: `home.html` and `result.html` .

app.py

The `app.py` file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying SMS messages:

```
from flask import Flask,render_template,url_for,reques
import pandas as pd
import pickle
from sklearn.feature_extraction.text import CountVectc
from sklearn.naive_bayes import MultinomialNB
from sklearn.externals import joblib


app = Flask(__name__)

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

@app.route('/predict',methods=['POST'])
def predict():
        df= pd.read_csv("spam.csv", encoding="latin-1"
        df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed:
        # Features and Labels
        df['label'] = df['class'].map({'ham': 0, 'spam
        X = df['message']
        y = df['label']

        # Extract Feature With CountVectorizer
        cv = CountVectorizer()
        X = cv.fit_transform(X) # Fit the Data
        from sklearn.model_selection import train_test
        X_train, X_test, y_train, y_test = train_test_
        #Naive Bayes Classifier
        from sklearn.naive_bayes import MultinomialNB

        clf = MultinomialNB()
```

app.py

- We ran our application as a single module; thus we initialized a new Flask instance with the argument `__name__` to let Flask know that it can find the HTML template folder (`templates`) in the same directory where it is located.

- Next, we used the route decorator (`@app.route('/')`) to specify the URL that should trigger the execution of the `home` function.
- Our `home` function simply rendered the `home.html` HTML file, which is located in the `templates` folder.
- Inside the `predict` function, we access the spam data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
- we used the `POST` method to transport the form data to the server in the message body. Finally, by setting the `debug=True` argument inside the `app.run` method, we further activated Flask's debugger.
- Lastly, we used the `run` function to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the `if` statement with `__name__ == '__main__'` .

home.html

The following are the contents of the `home.html` file that will render a text form where a user can enter a message:

```
<!DOCTYPE html>
<html>
<head>
        <title>Home</title>
        <!-- <link rel="stylesheet" type="text/css" hr
        <link rel="stylesheet" type="text/css" href="{
</head>
<body>

        <header>
                <div class="container">
                <div id="brandname">
                        Machine Learning App with Flas
                </div>
                <h2>Spam Detector For SMS Messages</h2

        </div>
        </header>

        <div class="ml-container">

                <form action="{{ url_for('predict')}}"
                <p>Enter Your Message Here</p>
                <!-- <input type="text" name="comment"
                <textarea name="message" rows="4" cols
```

home.html

style.css

In the header section of `home.html`, we loaded `styles.css` file. CSS is to determine how the look and feel of HTML documents. `styles.css` has to be saved in a sub-directory called `static`, which is the default directory where Flask looks for static files such as CSS.

```css
body{
        font:15px/1.5 Arial, Helvetica,sans-serif;
        padding: 0px;
        background-color:#f4f3f3;
}

.container{
        width:100%;
        margin: auto;
        overflow: hidden;
}

header{
        background:#03A9F4;#35434a;
        border-bottom:#448AFF 3px solid;
        height:120px;
        width:100%;
        padding-top:30px;

}

.main-header{
                        text-align:center;
                        background-color: blue;
                        height:100px;
                        width:100%;
                        margin:0px;
                }
#brandname{
        float:left;
        font-size:30px;
        color: #fff;
        margin: 10px;
}

header h2{
```

style.css

result.html

we create a `result.html` file that will be rendered via the `render_template('result.html', prediction=my_prediction)` line return inside the `predict` function, which we defined in the `app.py` script to display the text that a user submitted via the text field. The `result.html` file contains the following content:

```
<!DOCTYPE html>
<html>
<head>
        <title></title>
    <link rel="stylesheet" type="text/css" href="{{ ur
</head>
<body>

        <header>
                <div class="container">
                <div id="brandname">
                        ML App
                </div>
                <h2>Spam Detector For SMS Messages</h2

        </div>
        </header>
        <p style="color:blue;font-size:20;text-align:
        <div class="results">


```

result.html

From `result.htm` we can see that some code using syntax not normally found in HTML files: `{% if prediction ==1%},{% elif prediction == 0%},{% endif %}` This is jinja syntax, and it is used to access the prediction returned from our HTTP request within the HTML file.

We are almost there!

Once you have done all of the above, you can start running the API by either double click `appy.py` , or executing the command from the Terminal:

```
cd SMS-Message-Spam-Detector
python app.py
```

You should get the following output:

```
* Restarting with stat
* Debugger is active!
* Debugger PIN: 386-058-881
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 3

Now you could open a web browser and navigate to http://127.0.0.1:5000/, we should see a simple website with the content like so:

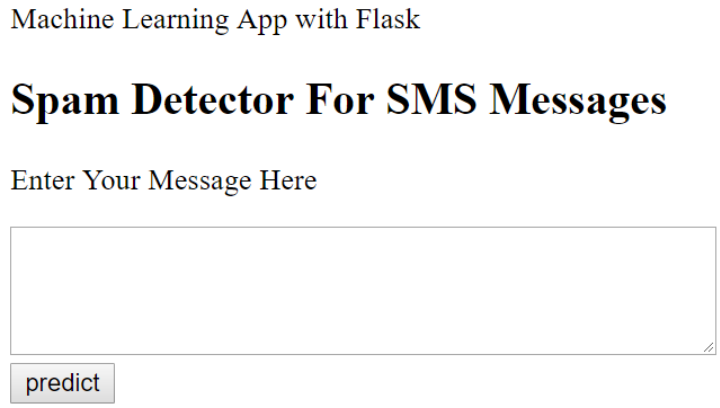
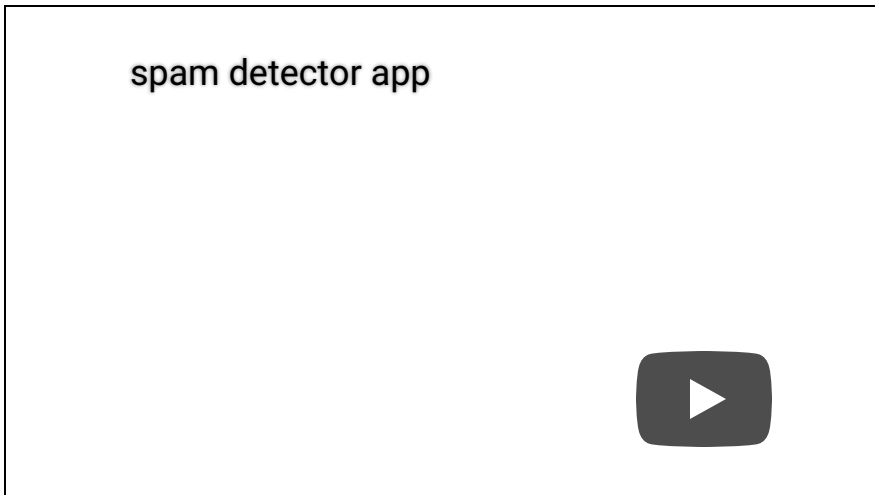


Figure 4

Let's test our work!



spam_detector_app

Congratulations! We have now created an end-to-end machine learning (NLP) application at zero cost. If you look it back, the overall process is not complicated at all. With a little bit patience and desire to learn, anyone can do it. All the open-source tools make every thing possible.

More importantly, we are able to extend our knowledge of machine learning theory to a useful and practical web application and lets us make our SMS spam message classifier available to the outside world!

The complete working source code is available at this repository.
Have a great week!

Reference:

Book: Python Machine Learning