How to build a text classifier web app.

Today we are going to train a model, build a web interface that allows people to submit data, send that data to our model, make a prediction on it, then return the results.

This is how your app directory should look.

```
MyProject/
|-- my_app/
|   |-- app.py
|   |-- build_model.py
|   |-- templates
|   |   |-- index.html
|   |   |-- submit.html
|   |   |-- predict.html
|   |-- data
|   |   |-- articles.csv
|   |   |-- model.pkl
```

Step 1: Build your model

This exercise isn't about model tuning, so just build a basic model and don't worry about tuning it.

1. In build_model.py, build a text classifier model using the articles.csv dataset. You can use the body field to get the text and use it to predict the section_name.

   Feel free to choose something different, but here's an option:

```
* Use the `TfidfVectorizer` on the `body` field

* Use `MultinomialNB` model to predict the `section_name`
```

A stub build_model.py is included, note that we have wrapped both the TfidfVectorizerand MultinomialNB in a single class, and created sklearn style methods for fitting and prediction. This allows us work with only one object, and keeps our interfaces lean and clean.

You should save the model as a pickle file. This is python's internal format for saving objects to disk. Almost all python objects can be pickled, and then reloaded in another python process. Note though, to reload a pickled object, you *must* import the class defining that object in the process that would like to do the loading.

1. Check that you can reload your model and vectorizer by running these lines of code:

```python
import cPickle as pickle
import pandas as pd
from build_model import TextClassifier, get_data

with open('data/model.pkl') as f:
    model = pickle.load(f)

X, y = get_data('data/articles.csv')

print "Accuracy:", model.score(X, y)
print "Predictions:", model.predict(X)
```

Step 2: Build Your Site

(Use the example_with_form in the flask_examples as a guide.)

1. In the `app.py` file, build a welcome homepage. It should have a link to a `/submit` page.
2. Build the `/submit` page as an html form which accepts text data.
3. Build the `/predict` page which will display the result of the model prediction.

Extra Credit: Submit Text with Ajax

In the previous solution the app needs to load a new page once the text is submitted, you can avoid this and provide a smoother experience with some javascript. Re-write the necessary pieces of the previous app to submit the text using jquery to

- Submit a request containing the text using `$.ajax`.
- Respond to the request in Flask, sending back json containing the model's prediction.
- Use javascript to write the model's prediction into the html when the ajax request is completed.