Project 6: Dive Deep

Chris Peterson
General Assembly - Data Science Intensive - SM3
March 10, 2017

This is at the back side of the 80/20 rule.

The marginal gains are small compared to the timed being invested.

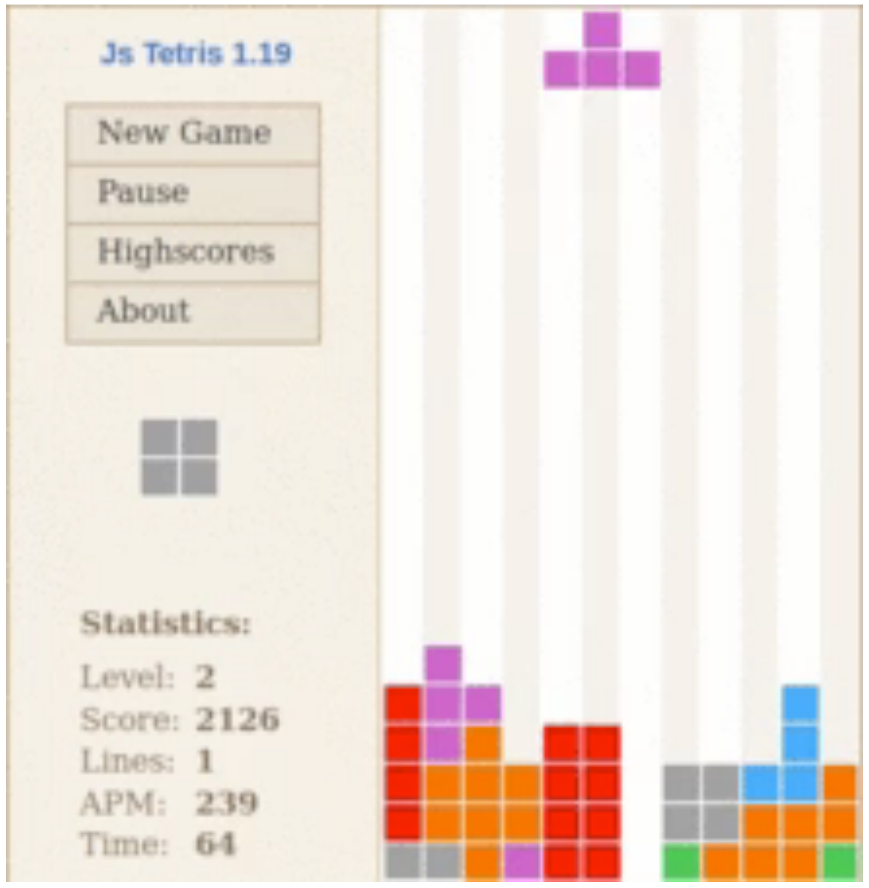
At least to me, it is important to be thorough.

My ultimate goal for the project to make it available via an API and allow it to grow by integrating new categories and documents. Then ultimately, incorporate new sources of documents.

I am continuing with Wikipedia, because I enjoy the randomness of what is included and documented.

Random Things included in this database:

Now we're going to step up the pace a little bit with the Prancercise Trot



Statistics:
Level: 2
Score: 2126
Lines: 1
APM: 239
Time: 64

Improvements on the final version of Project 6

Problem 1: API calls are limited to 500 responses.
Solution: Recursive function call with unlabeled arguments.

The API returns an element of the with the label "cmcontinue". It contains an string that allows the request to be continued for another 500 responses.

Solution: complete.

Problem 2: Service Architecture

The solution needs to be:

- Scalable
- Inexpensive
- Robust
- Independent of host
- Persistant

I’m choosing the Docker container system on AWS. At any point I should be able to choose to move to another service provider, with minimal effort.

Persistence is the next challenge.

Problem 3: Postgres on Docker

Beyond the base image, files in docker are ephemeral. The solution is using docker containers dedicated to interfacing a volume to other docker containers. Postgres, in particular, was challenging because it's. storage is a little more complex.

The best solution:

```
sudo docker create -v /mnt/postgres/postgres_files/ --name postgres_aws library/postgres /bin/true
sudo docker run -p 5432:5432 -e POSTGRES_PASSWORD=aPassword -d --volumes-from postgres_aws postgres
```

There is a little magic in there. I will examine it further later.

Problem 4: Data Acquisition

Downloading through the api is consistent but slow.

Problem 5: Rate limiting steps

Downloading is slow because it is a serial process. Moving to AWS provides an opportunity to parallelize the download. Redis Queue is a simple system to load a job into a queue then have ‘workers’ on other systems execute the function.

This needs to be fully implemented.

Problem 6: Modeling

When data acquisition is complete, what model should be used.
Model screening was done with train/test/validation data sets.
The categories used are in the table to the right. The validation category is a subcategory of main category.

category_name	# of	Validation Category
sports cars	593	muscle cars
Arcade games	1644	Cancelled arcade
desserts	161	cookies
chemistry	120	Industrial gases
physics	45	physicists
psychology	288	Popular psychology
cat breeds	94	Natural cat breeds
Earth sciences	105	Geochemistry
Sandwiches	122	American sandwiches
Breads	127	Bread dishes
Automotive technologies	163	Vehicle dynamics
belief	72	Ignorance
hygiene	95	Ritual purification
sports terminology	178	martial arts terminology
shoes	92	Sneaker culture
influenza	40	Influenza researchers
Children	6	Sons of Odin
Physical_quantities	232	Density
Submarine sandwich	42	
Association football trophies	58	
Physical exercise	190	hydrotherapy
health care	85	medical robotics
machine learning	189	Classification

Problem 6: Modeling Results

Best parameters as predicted by GridSearchCV

Model	**Training Accuracy**	**Test Accuracy**	**Validation Accuracy**	**params**	**Time to build model**	**Time to predict one page**
Cosine Similarity	94.5%	91.7%	72.3%		18.0 sec	4.1 ms
K Nearest Neighbors	90.2%	89.3%	64.0%		79 ms	1.45 ms
Multi Layer Perceptron	97.7%	94.8%	70.8%	activation= logistic, solver = 'adm', hidden_layer_size = (100,)	26.3 sec	3.7 ms
Random Forest Classifier	99.7%	91.5%	67.3%	estimators = 30	1.6 sec	120 µs
xgBoost	99.7%	92.7%	66.8%	n_estimators = 1000, max_depth = 5	39 min	6.6 ms

Problem 7: TF-IDF Features

Numbers, punctuation and other mark up features contributed many non-word items being included as features.
First pass of cleaning using:

```
temp = page.replace('\n',' ') #remove new lines
temp = temp.lower()
temp = re.sub(r'[^\w\s]',' ',temp) #remove all non-word, non-whitespace characters.
temp = re.sub(r'\[[0-9]{1,10}\]',' ', temp) #Remove strings of numbers
```

The number of features was reduced from ~130,000 to ~85,000.
More cleaning could be done, but it isn't needed at the moment.

Stemming and lemmatization are computationally expensive, but do not significantly reduce the feature set.

Problem 8: Alternate to truncatedSVD

The models were tested using tfidf vectors transformed with truncatedSVD with 500 features.

Neural Networks appear to work with large numbers of features to make accurate predictions.

As a first pass, sklearn’s Multi Layer Perceptron Classifier using tiff vectors as input made good predictions (train: 99.7%, test: 95.5%, Validation: 72.0%).

Keras with sparse matrices needs to be explored.

Problem 9: Data beyond Wikipedia

If this project is exposed to data sources other than wikipedia, the database needs to be restructured. The keys for the tables are based on the wikipedia page ids.

Adding other data sources will require adding fields for about the origin of the data, and the keys will need to be generated by Postgres.

Problem 10: Unicode and n-grams

Unicode and n-grams are sources of extra meaning and should be included. Currently, bánh mí doesn’t return the same search results as banh mi. Either bigrams or better handling of unicode should correct this problem. Multiple encoding schemes might improve search results and classifications.

This might mean rewriting all the modules in Python 3.

Problem 11: The Footprints

Of course, this problem has been solved before.

LSA on wikipedia is an example included in Gensim.

And all the data acquisition can be bypassed by using data dump from Wikipedia.

Implementing Gensim on AWS a stretch goal for this project.