Part 2: Capstone Progress

```
Identify: Articulate Problem Statement/Specific goals & success crit
eria
Identify: Outline proposed methods & models
Parse: Identify risks & assumptions
Parse: Your Database solution
Parse: Your data source and some data challenges
Parse: Interesting data structures
Mine: Perform & summarize EDA (optional)
```

Identify: Articulate Problem Statement/Specific goals & success criteria

I am extending Project 6.

First, I need to improve on the piplines from project 6. There are a number of tweaks to improve behavior of the piplines.

Second, Improving the predictive categorization is major step. Starting with Cosine Similarity, the prediction on the train/test split is nearly 90% accuracy. XGboost performs about 92% performance. A wider range of classsifiers need be explored and optimized to improve accuracy and and generalization.

The ultimate goal is to accurately predict categorization of new documents, and integrate new topics.

The final implemtation is intended to run in a containerized service, with queues utilized to add new categories.

Identify: Outline proposed methods & models

Methods:

- Utilize wikipedia api to acquire baseline of categories to be matched compared.
- Use LSA to model categorizations.
- Test calssification methods to quickly cateogrize new data.
- Utilize Docker or other containers to make this Data aquisition, model generation and data storage.
- Automate category addition when there is poor category fit.

Parse: Identify risks & assumptions

Risks:

- Replicating the work of half class/Project will not be distinct from others.

Assumption:

- There is value in knowin LSA. Is it a commidity level skill?

Parse: Your Database solution

Presently, I'm using postgres based on ease of use. The main advantage is maintaining references for a many-to-many relationship and using that relationship to predict matches for multiple categories.

Moving to a NoSQL soltution, may be very helpful when datasources other than wikipedia are included. Also, the current index of categories is based on the the wikipedia catgory_id/page_id of the category. This should be made independent of the datasource and another field can contain the metadata about sources. Maybe as a JSON object. Maybe this should be in Mongo.

Parse: Your data source and some data challenges

Data challenges: Cleaning the text to deal with unicode elements: the especially the punctuation elements, (em-dash, double dash).

Parse: Interesting data structures

The data structures are very simple. Just lists, dicts, and tuples. The utilization could better and standardized. All the code will need to be refactored to utilze the data better.

Also, much of the code could be abstracted further. to minimize redunancies.

Mine: Perform & summarize EDA (optional)

EDA: This is much more focused on the process than the results of the data. The data is well formed, but it does contain some pesky artifacts. There maybe utiliity/oportunities to include various markup langauges, like latex and markdown.

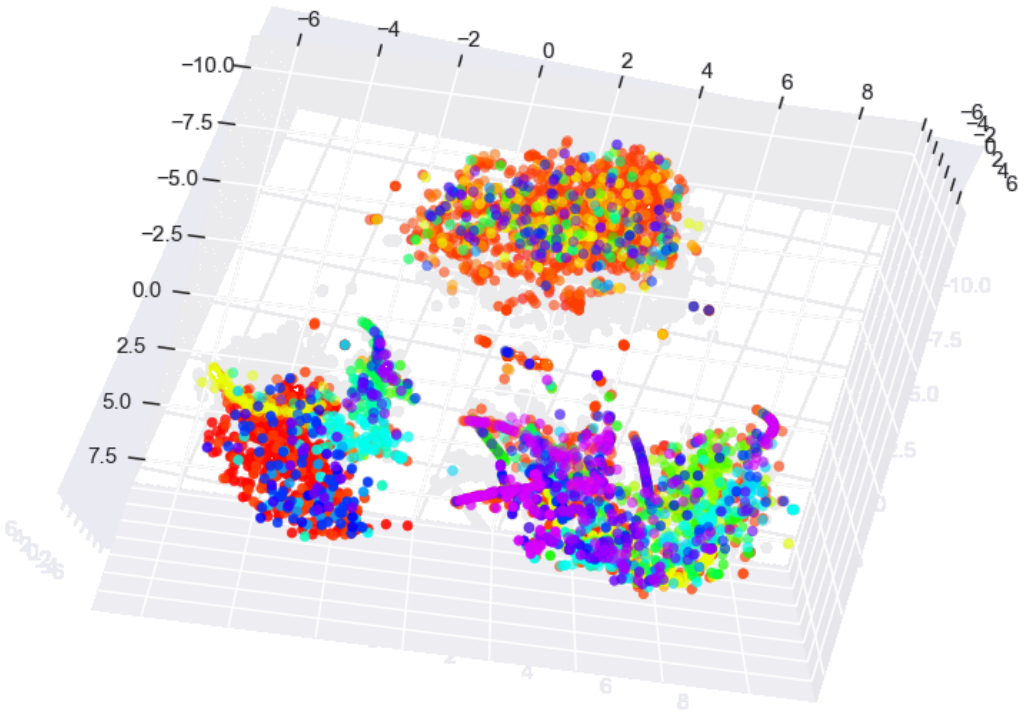
The main group of 23 categories from Wikipedia shows a pretty broad distribution of content.

category_name	**count**
sports cars	593
Arcade games	1644

category_name	count
desserts	161
chemistry	120
physics	45
psychology	288
cat breeds	94
Earth sciences	105
Sandwiches	122
Breads	127
Automotive technologies	163
belief	72
hygiene	95
sports terminology	178
shoes	92
influenza	40
Children	6
Physical_quantities	232
Submarine sandwich restaurants	42
Association football trophies and awards	58
Physical exercise	190
health care	85
machine learning	189

There is a single category, Arcade Games, with much larger representation in the corpus than the average category(206 articles, on average). This category does appear in the more frequently in the search function than other categories, in part due to it's size, and in part due to the breadth of topics in that category. If this trend continutes with the addtion of more categories, the construction of the model will need to be addressed.

After reducing the features of the SVD and a tSNE transformation, there do appear to be some clusters. Each color is a different category.



Current todo list. It is evolving and prone to being fanciful. Some of it is completed.

Problems with the solution provide by the teams.

1. Values passed from module to module were not perfectly aligned. The modules can be brought refactored to be more efficient.
2. The downloads could not effectively handle categories that reached the maximum number of return from the api.
3. The model is seems very large for the amount of information included. For example, when the raw text was 29MB, the model is 410 MB.
4. Using a Postgres database, single or double quotes (‘ or ` or “) can cause problems. Moving to a noSQL database may improve this situation.
5. A progress bar would be nice to have for some of the longer processes.
6. Lemmatize!!
7. The stop words being used need to be examined, and how the model applies the cut off for features. Is the dropped features relevant.
8. Category prediction is really poor. Matching on the corpus of a category is not being effective. t-SNE might be an option or average over the kNN for a many neighbors.
9. Rewrite Search.py at work from Command line
10. Recurse over subcategories. Make pages members of parent and sub-categories.
11. Document the words for vectorization.
12. API
13. Web interface.
14. Generalize to unlabeled categories?
15. How to load the vectorizer from redis?

16. Update the vectorizer with diffs/rsync in redis? Or just overwrite? Or memcache?
17. Check times, especially downloading, encoding and database reads.
18. Suppress outputs/add verbose modes.
19. Refactor database module. abstract or remove the error handling.
20. Build docker then docker swarm.
21. Add rq-montioring.and rq-dashboard.
22. Docker compose = make the system
23. Docker swarm, many machines feeding a main docker client
24. Check if search model works better with string or list.
25. Grid search models, SVD components plot vs accuracy of test set to be categorized. Maybe use subcategories as test data.
26. Plot model creation time vs.
27. SGD for categorization, or Xgboost.
28. Test 2-grams or 3-grams.
29. Put progress bar on build_model.py
30. Add hash