

BATTLE OF THE BOTS



THE PROBLEM

We track technical support with ServiceNow tickets.

We want to map tickets to apps so GSA can understand the cost of support for its apps.

To get that useful data, we will want the data from service tickets to have a field that maps directly to the name of the app it applies to.

THE PROBLEM

For example, we would want this entry to be identified as belonging to ConcurGov

Description: “Concur - checking on status of claim”

Category: “ConcurGov”

Sub-category: “User Training”

Item: “Authorization”

THE PROBLEM

This is a classification problem, we want to know what app category should be assigned for each ServiceNow entry.

Humans can do this easily and accurately, but it's repetitive and takes time away from doing more valuable work, so this is a great task to automate.

THE CONTENDERS

Machine learning



VS.



Leveraging API data

THE CONTENDERS

Script leveraging GEAR API data

[app_identifier](#)

Machine learning script

[tmb_ml_discovery](#)

Code is available on GSA's GitHub org

github.com/GSA

WHAT IS MACHINE LEARNING?

You can think of machine learning as a broad group of algorithms that identify patterns



TMB_ml_discovery



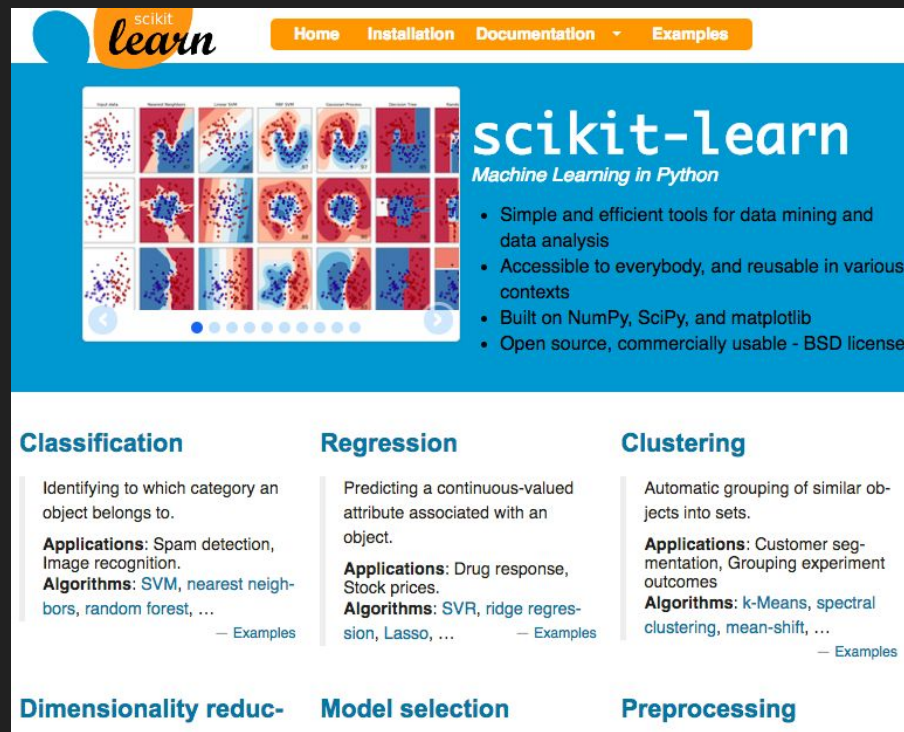
Uses Naive Bayes classification algorithm.

That means it calculates the chances that a word will be a part of a classification based on training data.

TMB_ML_Discovery

Prototyped using SciKit Learn in a Jupyter notebook.

SciKit Learn is an open source Python library for machine learning.



The image shows the homepage of the SciKit Learn website. At the top, there is a navigation bar with links for Home, Installation, Documentation, and Examples. The main header features the SciKit Learn logo and the text "Machine Learning in Python". Below the header, there is a grid of 12 small plots showing various machine learning results. To the right of the grid, there is a list of bullet points describing the library's features. Below the main content, there are three columns of text describing different machine learning tasks: Classification, Regression, and Clustering. Each column includes a brief definition, applications, and algorithms. At the bottom, there are three more columns: Dimensionality reduction, Model selection, and Preprocessing.

scikit-learn
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification
Identifying to which category an object belongs to.
Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...
— Examples

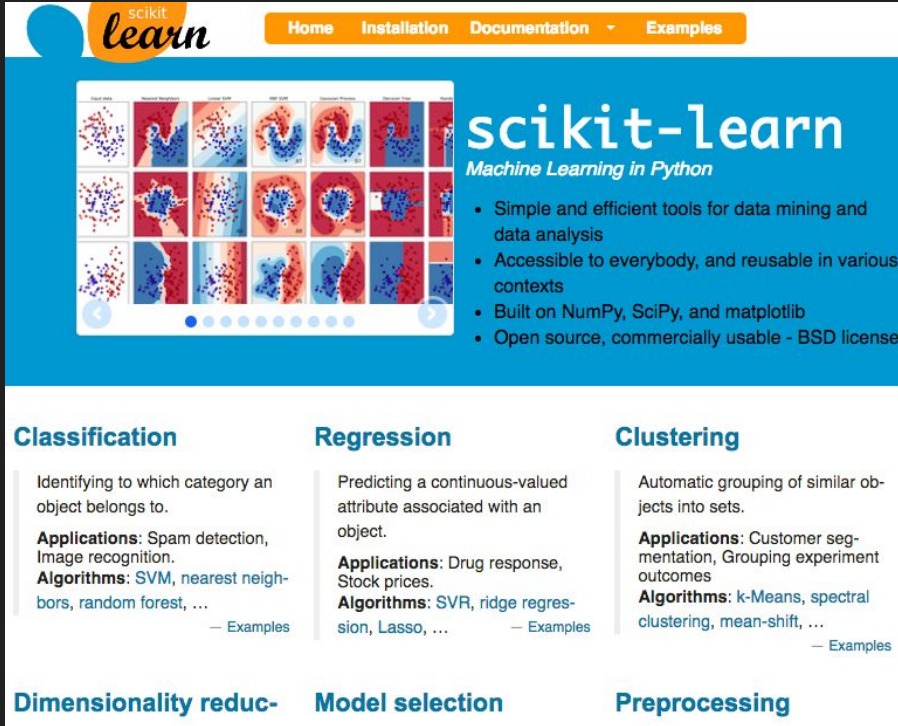
Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...
— Examples

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...
— Examples

Dimensionality reduction **Model selection** **Preprocessing**

TMB_ml_discovery

Hardest part is figuring out how to clean and format the data for processing.



scikit-learn

Home Installation Documentation Examples

scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ...

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ...

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ...

— Examples

Dimensionality reduction

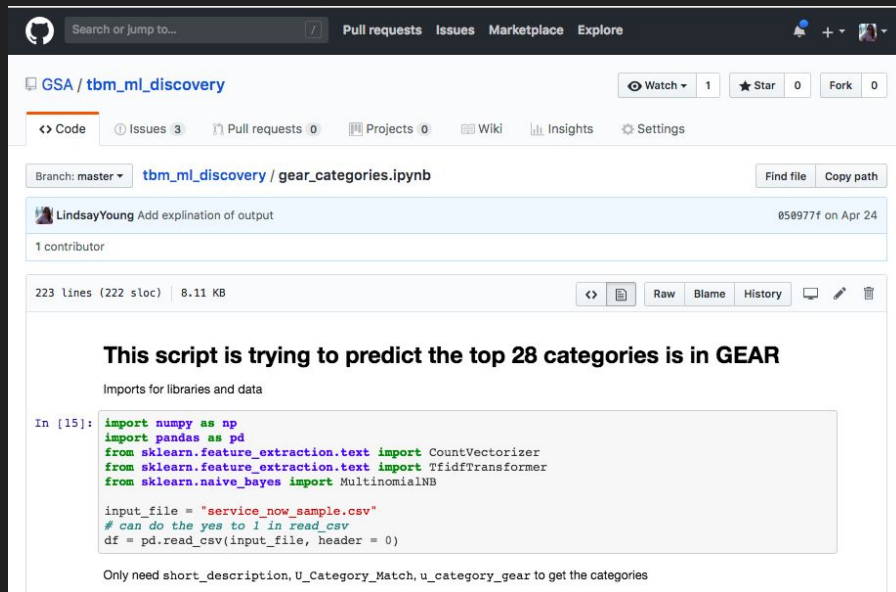
Model selection

Preprocessing

TMB_ML_DISCOVERY

Uses **Pandas** to easily
work with CSVs

Numpy and **SciKit Learn**
do the heavy lifting



The screenshot shows a GitHub repository page for 'GSA / tmb_ml_discovery'. The file 'gear_categories.ipynb' is open, showing a Jupyter Notebook cell. The cell contains Python code for text classification using Pandas, Numpy, and SciKit Learn. The code imports the necessary libraries and reads a CSV file named 'service_now_sample.csv'. The output of the cell is a text description of the script's purpose: 'This script is trying to predict the top 28 categories is in GEAR'.

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

input_file = "service_now_sample.csv"
# can do the yes to 1 in Read_csv
df = pd.read_csv(input_file, header = 0)
```

Only need short_description, U_Category_Match, u_category_gear to get the categories

TMB_mL_Discovery

After loading the sample data CSV into a dataframe, I eliminated the columns that I did not need and added a column for each app, 1 if it was that category 0 if it was not.

Only need short_description, U_Category_Match, u_category_gear to get the categories

```
In [16]: gear_df = df[['short_description', 'u_category_gear']]
# remove nulls
gear_df = gear_df.replace(np.nan, '', regex=True)

gear_df['eoffer_emod'] = np.where(gear_df['u_category_gear']=='eOffer/eMod - Electronic Offers/Ele
ctronic Modifications', 1, 0)
```

TMB_ML_DISCOVERY

Vectorizing means assigning numbers to words to keep track of the vocabulary.

Ignore stop words like “a”, “the”, etc.

```
In [17]: # vectorize discription
count_vect = CountVectorizer(stop_words='english')
X_train_counts = count_vect.fit_transform(gear_df['short_description'])
X_train_counts.shape
```

```
Out[17]: (83933, 21967)
```

TMB_ML_DISCOVERY

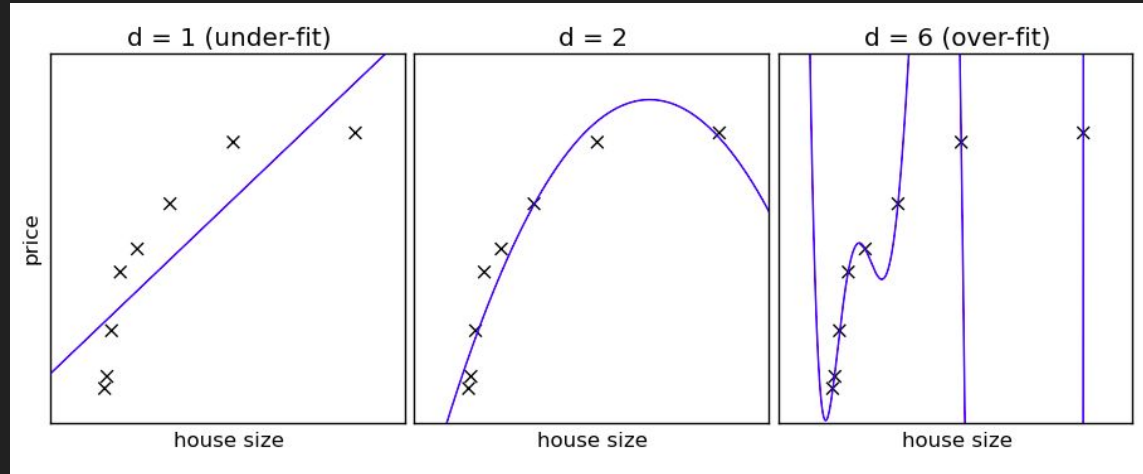
Shaping the data is a technique to normalize a data set.

```
In [18]: # shape data
         tfidf_transformer = TfidfTransformer()
         X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
         X_train_tfidf.shape
```

```
Out[18]: (83933, 21967)
```

SHAPING DATA

The library will normalize for us, Naive Bayes is high bias - low variance



TMB_ML_DISCOVERY

Loop through each app model to create a predictive model and apply it to the existing data as a test.

This will train and test the data for each category. It prints out each category and the accuracy of the model. So, my_app 0.99 means that the data model predicted the correct category for my app 99% of the time.

```
In [19]: # The most frequently appearing apps in the data
top_apps = ['eoffer_emod', 'vcss', 'any_connect_windows', 'easi', 'google_email', 'pegasys_admin',
            'fss_online', 'etams', 'pegasys_data', 'aloha', 'fmis', 'apm', 'google_docs', 'google_ch
            rome',
            'ears', 'bookit', 'rocis', 'eviewer', 'google_calendar', 'geco', 'ors', 'google_sites',
            'bi',
            'google_hangout', 'google_groups', 'vitap', 'ocms', 'pegasys_vrm']

for app in top_apps:
    formatted_category = gear_df[[app]]
    text_clf = MultinomialNB().fit(X_train_counts, formatted_category.values.ravel())
    predicted = text_clf.predict(X_train_counts)
    print(app, np.mean(predicted == formatted_category.values.ravel()))
    gear_df[app] = predicted
```


TMB_ML_Discovery

Results:

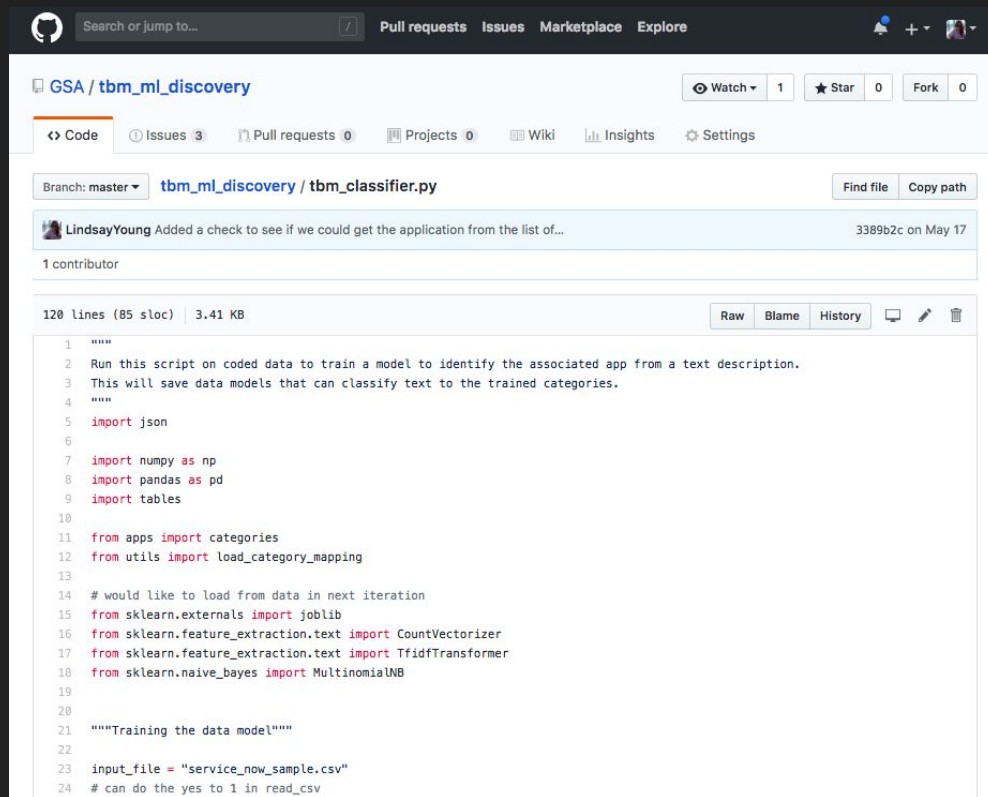
99% to 97% accuracy on each term.

Combined, the data was closer to 80% accurate.

TMB_ML_DISCOVERY

Turned this into
a script.

Trains the data
models and
predicts new
data.



The screenshot shows a GitHub repository page for 'GSA / tmb_ml_discovery'. The repository has 1 watch, 0 stars, and 0 forks. The current branch is 'master', and the selected file is 'tmb_ml_discovery / tmb_classifier.py'. A commit by LindsayYoung is shown, with the message 'Added a check to see if we could get the application from the list of...' and the commit hash '3389b2c' on May 17. The file 'tmb_classifier.py' is 120 lines long (85 sloc) and 3.41 KB. The code is as follows:

```
1 """
2 Run this script on coded data to train a model to identify the associated app from a text description.
3 This will save data models that can classify text to the trained categories.
4 """
5 import json
6
7 import numpy as np
8 import pandas as pd
9 import tables
10
11 from apps import categories
12 from utils import load_category_mapping
13
14 # would like to load from data in next iteration
15 from sklearn.externals import joblib
16 from sklearn.feature_extraction.text import CountVectorizer
17 from sklearn.feature_extraction.text import TfidfTransformer
18 from sklearn.naive_bayes import MultinomialNB
19
20
21 """Training the data model"""
22
23 input_file = "service_now_sample.csv"
24 # can do the yes to 1 in read_csv
```

TMB_mL_Discovery

Added some logic from the original excel lookup logic and accuracy improved to **92.5%** matches compared to the original data

Not bad, but wanted to test this against
another approach

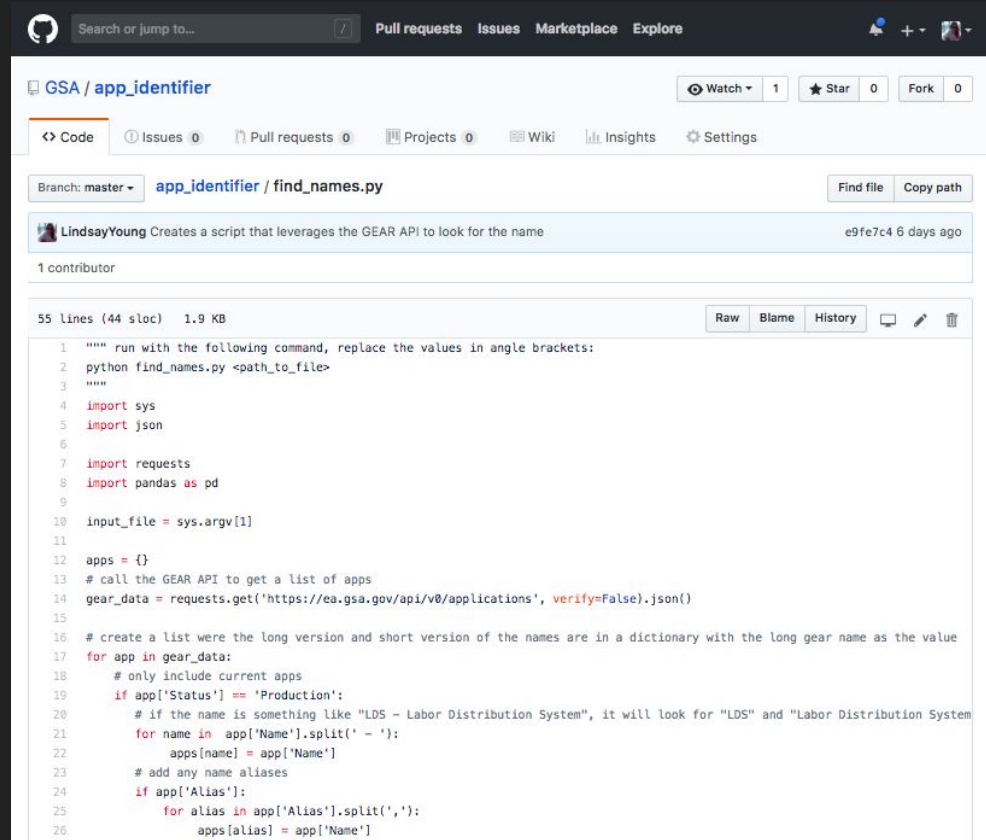
app_IDENTIFIER

Uses the **GEAR API** to pull a current list of applications. It then looks for the name, acronym or nickname of the app in the text of the ServiceNow ticket.



app_identifier

Calls the GEAR API
and creates a
dictionary of current
app names or
nicknames and the
app it belongs to



```
1 """ run with the following command, replace the values in angle brackets:
2 python find_names.py <path_to_file>
3 """
4 import sys
5 import json
6
7 import requests
8 import pandas as pd
9
10 input_file = sys.argv[1]
11
12 apps = {}
13 # call the GEAR API to get a list of apps
14 gear_data = requests.get('https://ea.gsa.gov/api/v0/applications', verify=False).json()
15
16 # create a list where the long version and short version of the names are in a dictionary with the long gear name as the value
17 for app in gear_data:
18     # only include current apps
19     if app['Status'] == 'Production':
20         # if the name is something like "LDS - Labor Distribution System", it will look for "LDS" and "Labor Distribution System"
21         for name in app['Name'].split(' - '):
22             apps[name] = app['Name']
23         # add any name aliases
24         if app['Alias']:
25             for alias in app['Alias'].split(','):
26                 apps[alias] = app['Name']
```

APP_IDENTIFIER

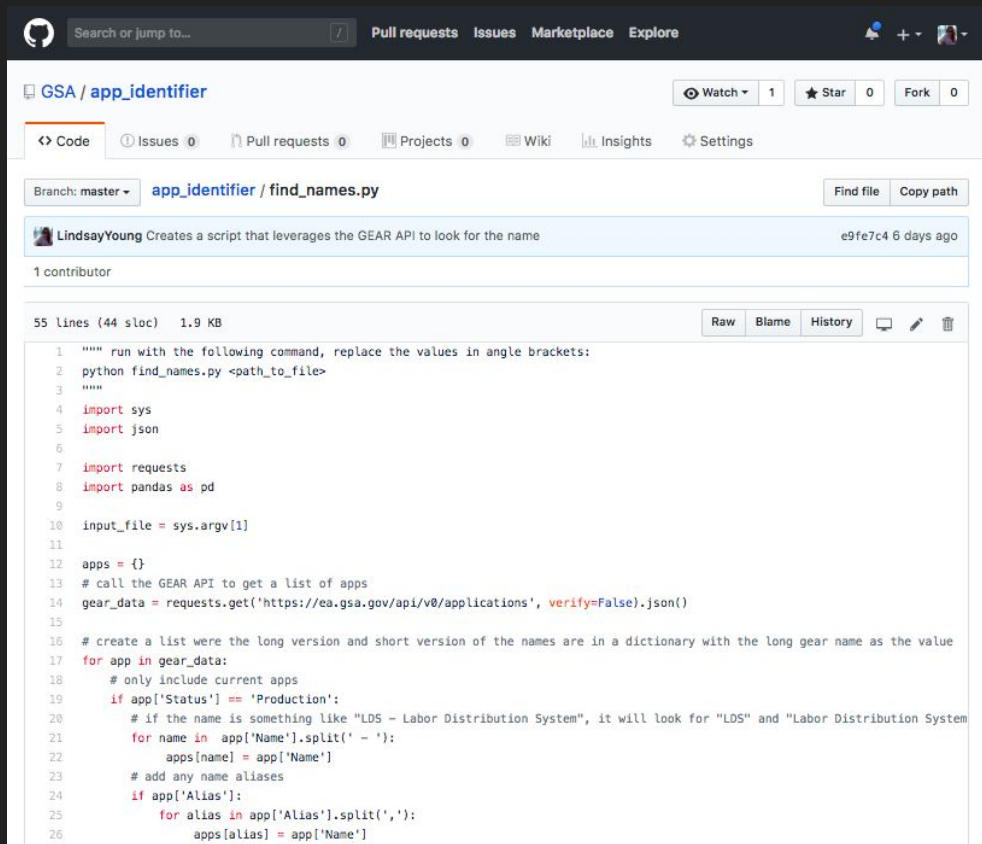
Also uses a csv of key phrases to get better results.

Looks for names, nicknames and phrases in the text.

```
28 # add some extra matches from the phrase_enhancements.csv file
29 extra_apps = {}
30 phrase_df = pd.read_csv('phrase_enhancements.csv', header = 0)
31 for row in phrase_df:
32     extra_apps[row[0]] = row[1]
33
34 # combine the GEAR dictionary with the phrase dictionary
35 apps.update(extra_apps)
36
37 # reads the file that needs to be categorized
38 df = pd.read_csv(input_file, header = 0)
39
40 # looks at each row in key fields where the app name or key phrase might appear
41 def look_4_name(row):
42     text = str(row['short_description']) + str(row['u_category']) + str(row['u_subcategory']) + str(row['u_item'])
43     # looks to match the longest words first
44     for key in sorted(apps, key=len, reverse=True):
45         if key in str(text):
46             return apps[key]
47     # no matches in GEAR
48     return 'Allocate according to Apptio "Incident Category Crosswalk"'
49
50 df['name_prediction'] = df.apply(lambda row: look_4_name(row), axis=1)
51
52 # write results to a file
53 new_file_name = input_file[:-4] + "_processed.csv"
54 df.to_csv(new_file_name)
```

APP_IDENTIFIER

This simple approach gives 87% matches compared to the original data.



```
1 """ run with the following command, replace the values in angle brackets:
2 python find_names.py <path_to_file>
3 """
4 import sys
5 import json
6
7 import requests
8 import pandas as pd
9
10 input_file = sys.argv[1]
11
12 apps = {}
13 # call the GEAR API to get a list of apps
14 gear_data = requests.get('https://ea.gsa.gov/api/v0/applications', verify=False).json()
15
16 # create a list where the long version and short version of the names are in a dictionary with the long gear name as the value
17 for app in gear_data:
18     # only include current apps
19     if app['Status'] == 'Production':
20         # if the name is something like "LDS - Labor Distribution System", it will look for "LDS" and "Labor Distribution System"
21         for name in app['Name'].split(' - '):
22             apps[name] = app['Name']
23         # add any name aliases
24         if app['Alias']:
25             for alias in app['Alias'].split(','):
26                 apps[alias] = app['Name']
```


STRENGTHS and Weaknesses



TMB_ML_DISCOVERY STRENGTHS



- Pretty good accuracy
- Can leverage existing SciKitLearn library
- Lots of examples of classification with machine learning
- Fun to build

TMB_ML_DISCOVERY weaknesses



- Not the most elegant implementation
 - Code creating code
 - More code and dependencies
- Hard to update
 - Needs numerous data samples when there is a new app

APP_IDENTIFIER STRENGTHS

- Adapts to the changing app list automatically
- Pretty good (not as good) accuracy
- Found additional apps hand coding missed.
- Very straightforward code base



app_IDENTIFIER weaknesses

- Not as accurate
- Some extra false positives



APP_IDENTIFIER wins!



CONCLUSIONS

The goal for this project was to save time.

Needing to add numerous training samples every time there was a new app, would negate the time savings from automation.

Choosing maintainability and simplicity was the best choice to save time.