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 mapps 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

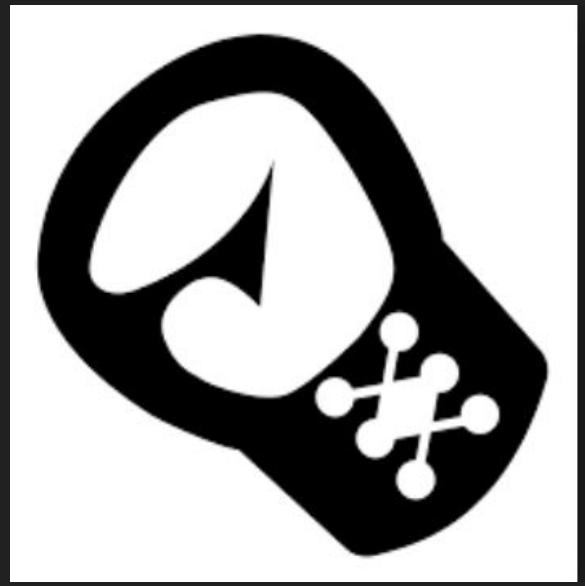
This is a classification problem, we want to know what app category should be assigned for each ServiceNow entry.

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

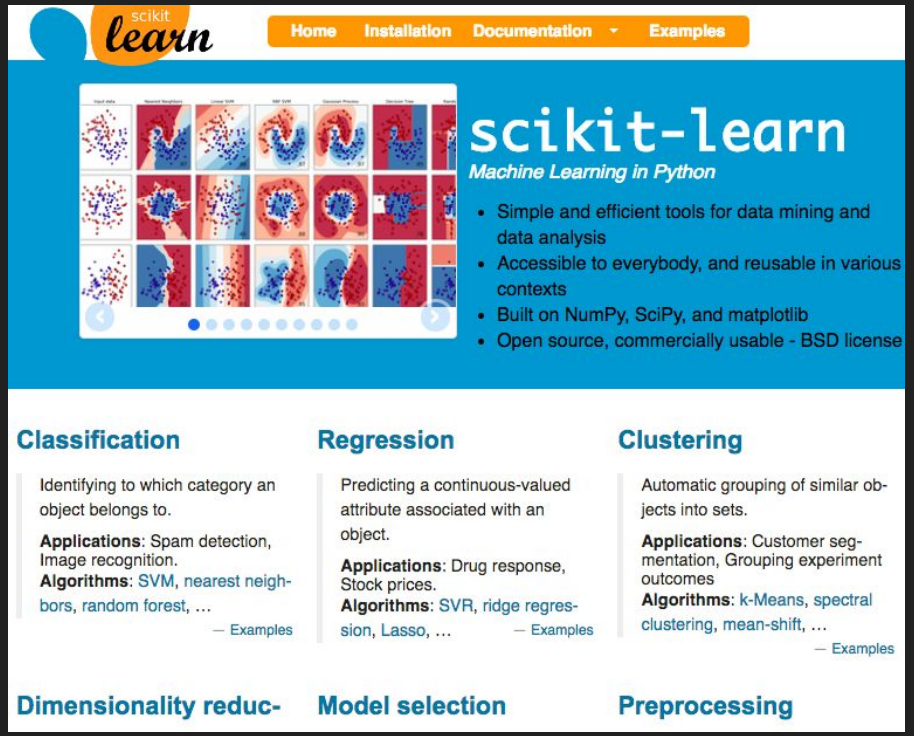
Uses Naive Bayes classification algorithm.

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

TMB_ML_Discovery

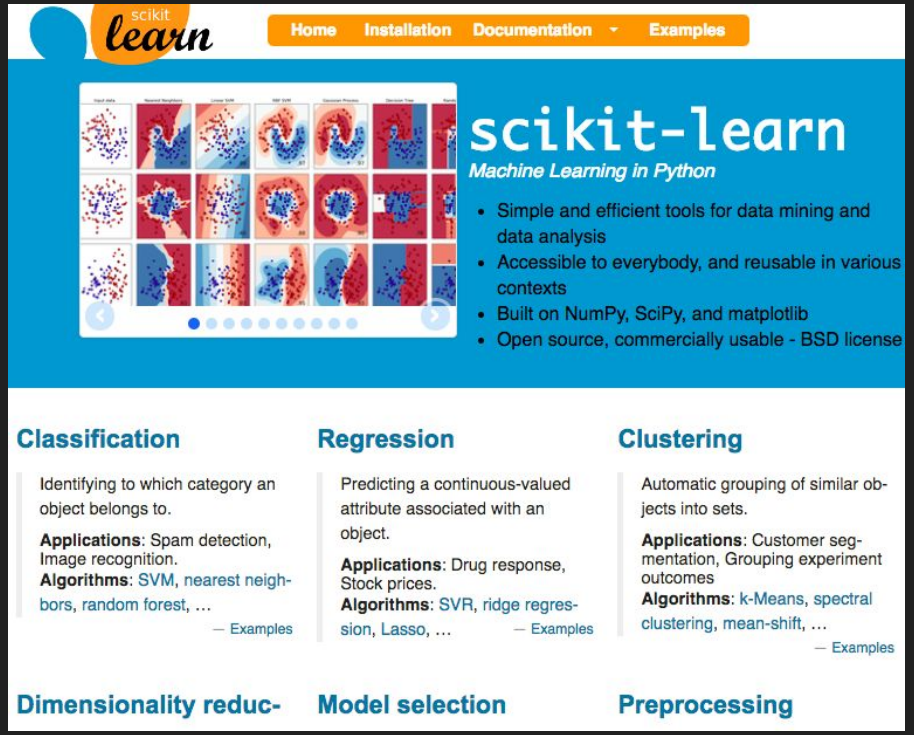
Prototyped using SciKit Learn in a Jupyter notebook.

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



TMB_ML_Discovery

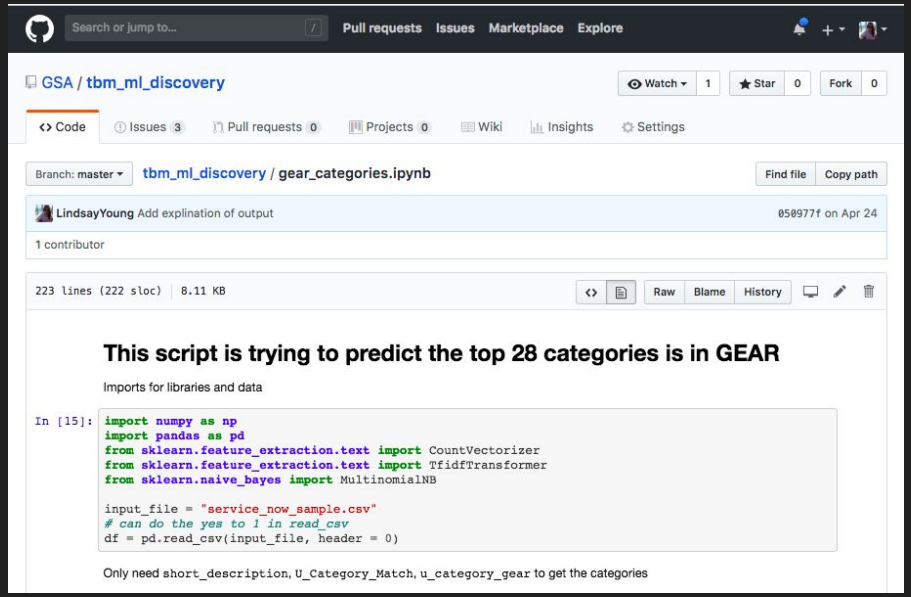
Hardest part is figuring out how to clean and format the data for processing.



TMB_ML_DISCOVERY

Uses Pandas to easily work with CSVs

Numpy and SciKit Learn do the heavy lifting



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/Electronic Modifications', 1, 0)
```

TMB_ML_Discovery

Vectorizing means assigning numbers to 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)
```

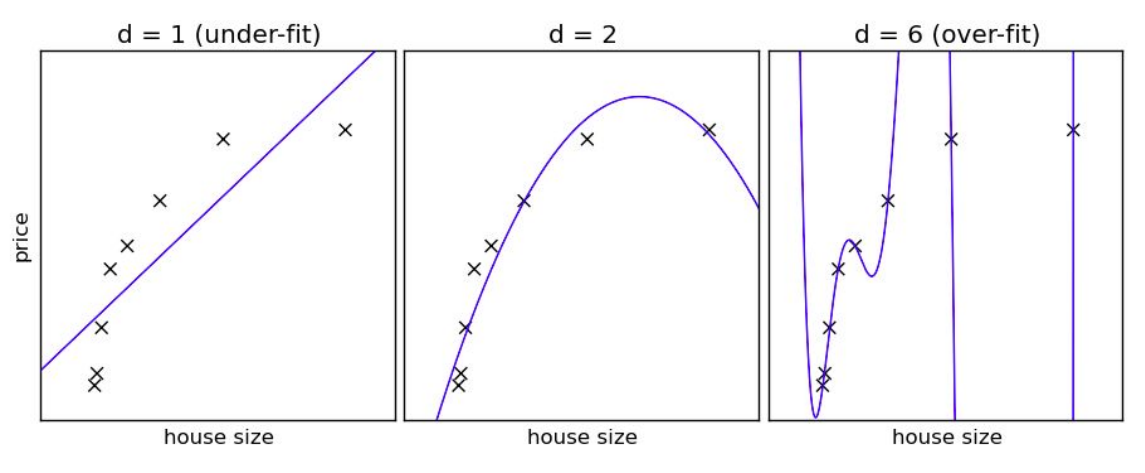
TMB_ML_Discovery

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



From: https://radimrehurek.com/data_science_python/

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

TMB_ML_Discovery

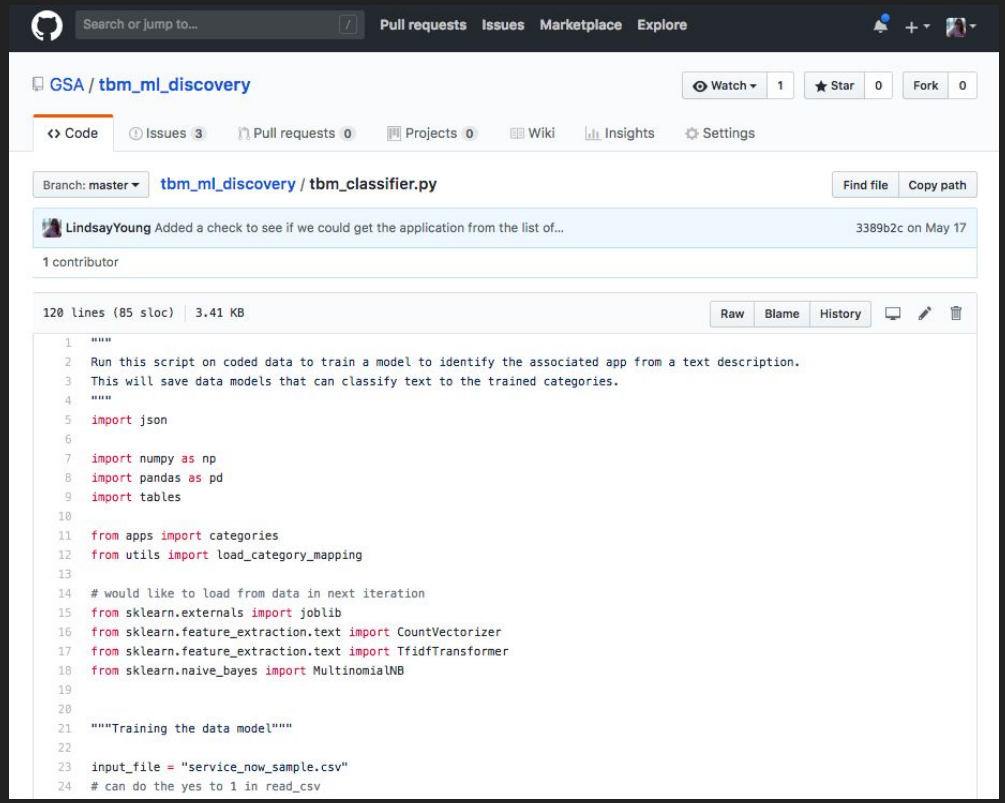
Results:

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.



TMB_ML_Discovery

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

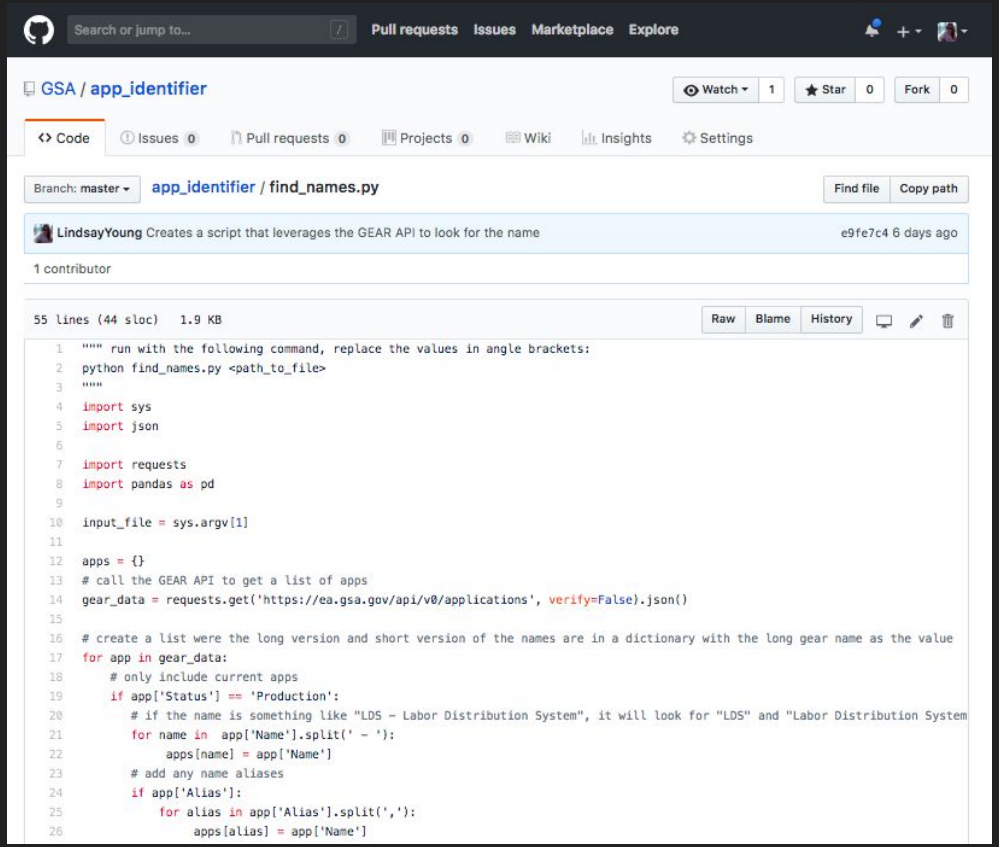
Not bad, but wanted to test this against another approach

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



Search or jump to... / Pull requests Issues Marketplace Explore

GSA / app_identifier

Watch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master app_identifier / find_names.py Find file Copy path

LindsayYoung Creates a script that leverages the GEAR API to look for the name e9fe7c4 6 days ago

1 contributor

55 lines (44 sloc) 1.9 KB Raw Blame History

```
""" run with the following command, replace the values in angle brackets:
python find_names.py <path_to_file>
"""
import sys
import json

import requests
import pandas as pd

input_file = sys.argv[1]

apps = {}
# call the GEAR API to get a list of apps
gear_data = requests.get('https://ea.gsa.gov/api/v0/applications', verify=False).json()

# create a list were the long version and short version of the names are in a dictionary with the long gear name as the value
for app in gear_data:
    # only include current apps
    if app['Status'] == 'Production':
        # if the name is something like "LDS - Labor Distribution System", it will look for "LDS" and "Labor Distribution System
        for name in  app['Name'].split(' - '):
            apps[name] = app['Name']
        # add any name aliases
        if app['Alias']:
            for alias in app['Alias'].split(','):
                apps[alias] = app['Name']
```

app_identifier

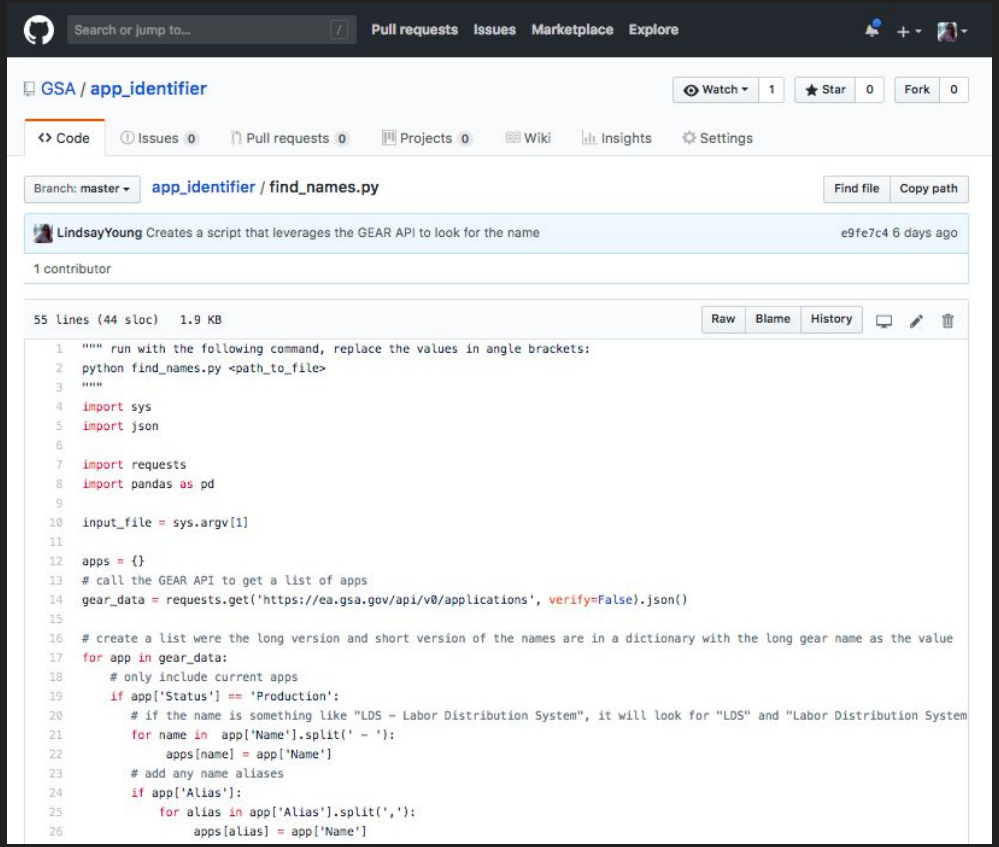
Also uses a csv of key phrases to get better results.

Looks for names, nicknames and phrases in the text.

```
# add some extra matches from the phrase_enhancements.csv file
extra_apps = {}
phrase_df = pd.read_csv('phrase_enhancements.csv', header = 0)
for row in phrase_df:
  extra_apps[row[0]] = row[1]

# combine the GEAR dictionary with the phrase dictionary
apps.update(extra_apps)

# reads the file that needs to be categorized
df = pd.read_csv(input_file, header = 0)

# looks at each row in key fields where the app name or key phrase might appear
def look_4_name(row):
    text = str(row['short_description']) + str(row['u_category']) + str(row['u_subcategory']) + str(row['u_item'])
    # looks to match the longest words first
    for key in sorted(apps, key=len, reverse=True):
        if key in str(text):
            return apps[key]
    # no matches in GEAR
    return 'Allocate according to Apptio "Incident Category Crosswalk"'

df['name_prediction'] = df.apply(lambda row: look_4_name(row), axis=1)

# write results to a file
new_file_name = input_file[:-4] + "_processed.csv"
df.to_csv(new_file_name)
```

app_identifier

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



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 matinability and simplicity was the best choice to save time.