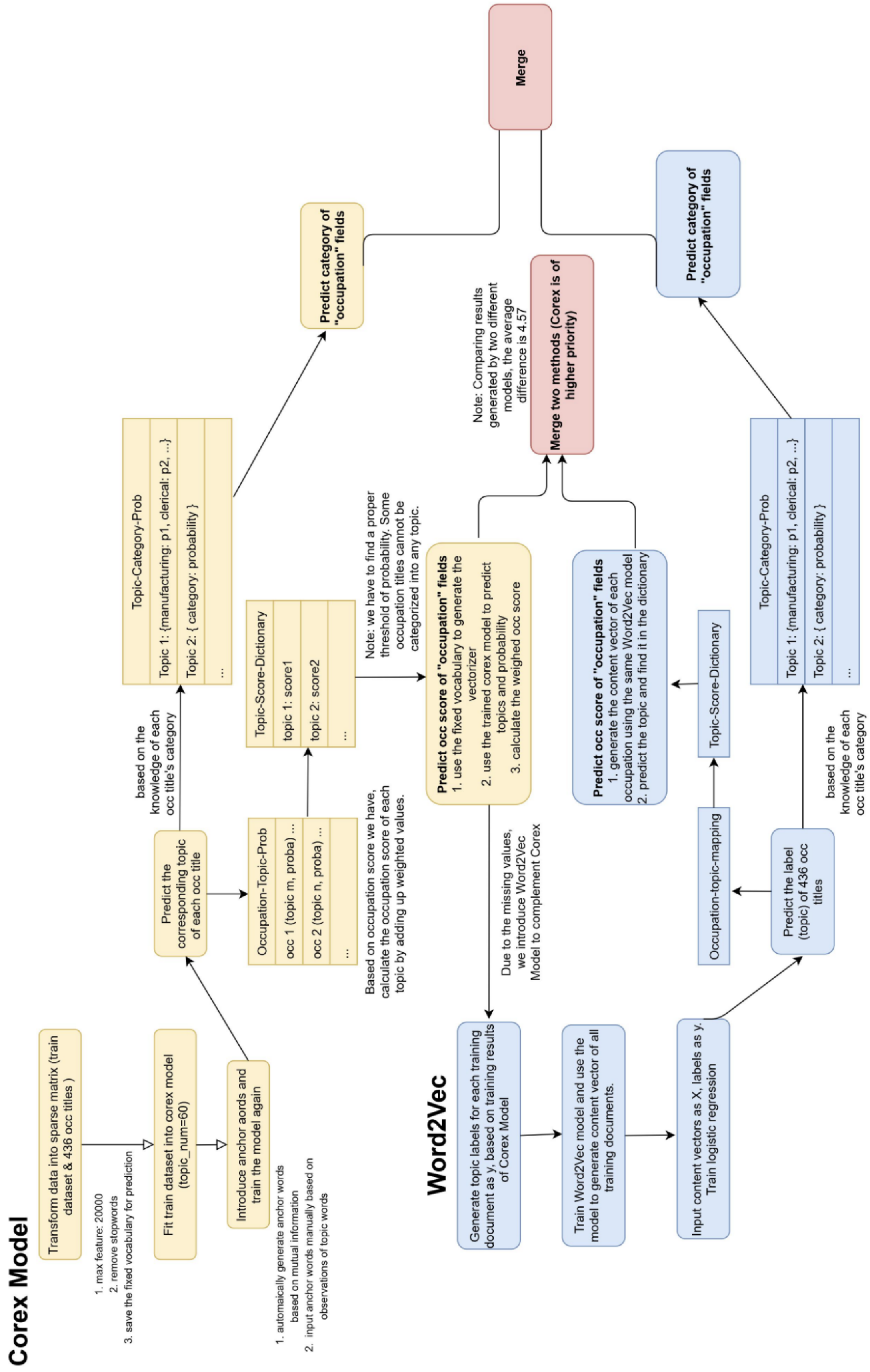
General Workflow


Corex Model
Transform data into sparse matrix (train dataset & 436 occ titles)
1. max feature: 20000
2. remove stopwords
3. save the fixed vocabulary for prediction
Fit train dataset into corex model (topic_num=60)
Introduce anchor aords and train the model again
1. automaically generate anchor words based on mutual information
2. input anchor words manually based on observations of topic words
Predict the corresponding topic of each occ title
based on the knowledge of each occ title's category
Topic-Category-Prob
Topic 1: {manufacturing: p1, clerical: p2, ...}
Topic 2: { category: probability }
...
Occupation-Topic-Prob
occ 1 (topic m, proba) ...
occ 2 (topic n, proba) ...
...
Based on occupation score we have, calculate the occupation score of each topic by adding up weighted values.
Topic-Score-Dictionary
topic 1: score1
topic 2: score2
...
Note: we have to find a proper threshold of probability. Some occupation titles cannot be categorized into any topic.
Predict occ score of "occupation" fields
1. use the fixed vocabulary to generate the vectorizer
2. use the trained corex model to predict topics and probability
3. calculate the weighed occ score
Predict category of "occupation" fields
Note: Comparing results generated by two different models, the average difference is 4.57
Merge two methods (Corex is of higher priority)
Merge
Due to the missing values, we introduce Word2Vec Model to complement Corex
Word2Vec
Generate topic labels for each training document as y, based on training results of Corex Model
Train Word2Vec model and use the model to generate content vector of all training documents.
Input content vectors as X, labels as y. Train logistic regression
Predict the label (topic) of 436 occ titles
Occupation-topic-mapping
based on the knowledge of each occ title's category
Topic-Category-Prob
Topic 1: {manufacturing: p1, clerical: p2, ...}
Topic 2: { category: probability }
...
Topic-Score-Dictionary
Predict occ score of "occupation" fields
1. generate the content vector of each occupation using the same Word2Vec model
2. predict the topic and find it in the dictionary
Predict category of "occupation" fields

Exploration of Corex Model

About Anchored CorEx

Github page:

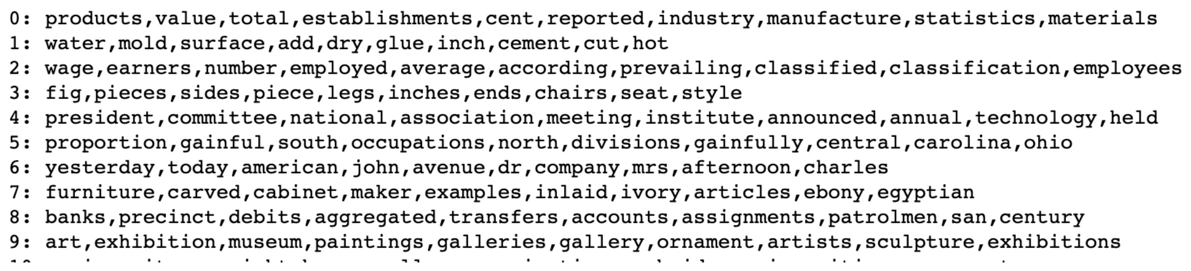
https://github.com/gregversteeg/corex_topic

- Correlation Explanation (CorEx) is a topic model that yields rich topics that are maximally informative about a set of documents. The advantage of using CorEx versus other topic models is that it can be easily run as an unsupervised, semi-supervised, or hierarchical topic model depending on a user's needs.
- For semi-supervision, CorEx allows a user to integrate their domain knowledge via "anchor words."

Comparison with LDA

Pros:

- Generate more coherent topics (by eyeballing)
- Semi-supervision can guide the training process, enhancing the model interpretability.



```
0: products,value,total,establishments,cent,reported,industry,manufacture,statistics,materials
1: water,mold,surface,add,dry,glue,inch,cement,cut,hot
2: wage,earners,number,employed,average,according,prevailing,classified,classification,employees
3: fig,pieces,sides,piece,legs,inches,ends,chairs,seat,style
4: president,committee,national,association,meeting,institute,announced,annual,technology,held
5: proportion,gainful,south,occupations,north,divisions,gainfully,central,carolina,ohio
6: yesterday,today,american,john,avenue,dr,company,mrs,afternoon,charles
7: furniture,carved,cabinet,maker,examples,inlaid,ivory,articles,ebony,egyptian
8: banks,precinct,debits,aggregated,transfers,accounts,assignments,patrolmen,san,century
9: art,exhibition,museum,paintings,galleries,gallery,ornament,artists,sculpture,exhibitions
```

Cons:

- Corex enforces single membership of words. If a user anchors a word to multiple topics, the single membership will be overridden.
- Corex does not provide a built-in function to calculate the coherence score, hence, it is hard for us to evaluate the performance.
- Fewer code sources of Corex
- Some texts cannot be categorized into any topic, which formulates missing values in future calculations.

Training Process

Data source:

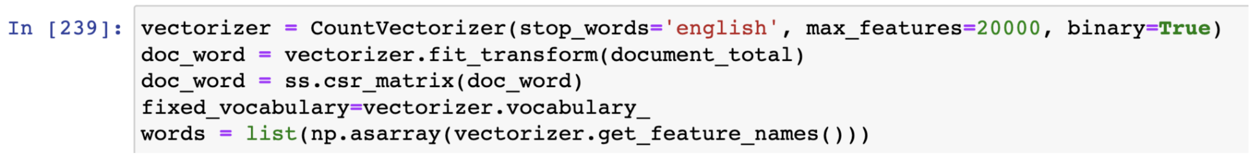
- train.npy # historical materials 4451 documents
- train2.npy # census bureau 4226 documents
- nyt_text_modified.txt #historical news fetched with New York Times API
- OCC pairs.csv #occupation titles

Data Preprocessing:

- Eliminate texts shorter than 15 words
- Chop long texts into smaller pieces of length 30
- Remove stopwords
- Remove common words (occupation, occupations, men, etc.)

Data transformation:

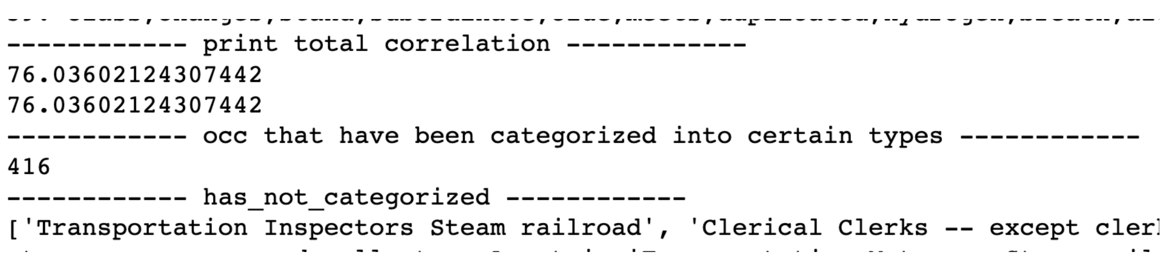
- Transform data into a sparse matrix with CountVectorizer (sklearn)
  - Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
  - Note: the attribute **fixed_vocabulary_** is a boolean value: True if a fixed vocabulary of term to indices mapping is provided by the user. We have to store the vocabulary in a fixed dictionary for future use.



```
In [239]: vectorizer = CountVectorizer(stop_words='english', max_features=20000, binary=True)
          doc_word = vectorizer.fit_transform(document_total)
          doc_word = ss.csr_matrix(doc_word)
          fixed_vocabulary=vectorizer.vocabulary_
          words = list(np.asarray(vectorizer.get_feature_names()))
```

Model training:

- Set topic_num to 60
- Based on observation of top 10 topic words, generate a list of anchor words. Choose the first 3 words (highest mutual information) as each topic's anchor words. Modify anchor words manually to make it suitable for occupation classification.
- Train the model again and predict the belonging topic of each occupation title.
- 416/436 can be categorized into certain topics.

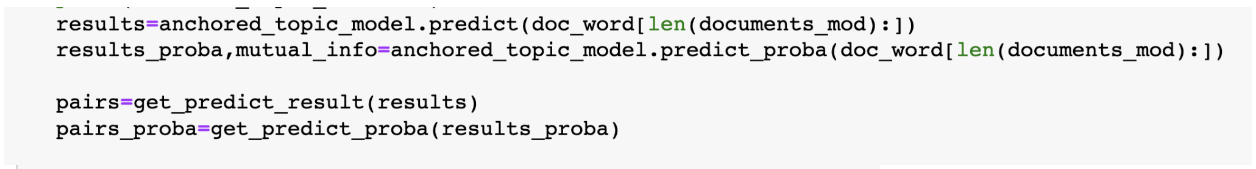


```
------------ print total correlation ------------
76.03602124307442
76.03602124307442
------------ occ that have been categorized into certain types ------------
416
------------ has_not_categorized ------------
['Transportation Inspectors Steam railroad', 'Clerical Clerks -- except cler
```

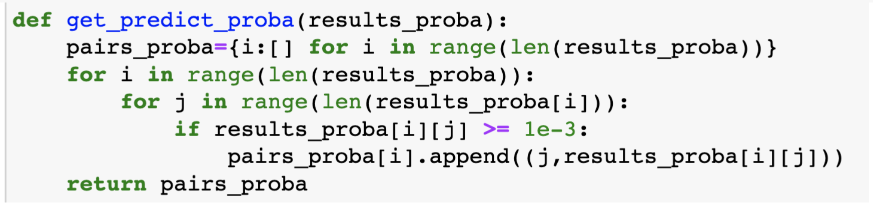
Occupation Score Calculation

Calculate occupation score for each topic:

- Corex has two predict functions: one is **model.predict( )**, the other one is **model.predict_proba( )** which returns a probability matrix and mutual information matrix.
- A problem of **get_predict_proba( )** function is that we have to discuss a proper threshold for the probability. For sure, 0.001 is an extremely low probability. If the probability of an occupation title belonging to a topic is very low, then we can say it cannot be fully trusted to add up the occupation score for that particular topic. Code as follows:

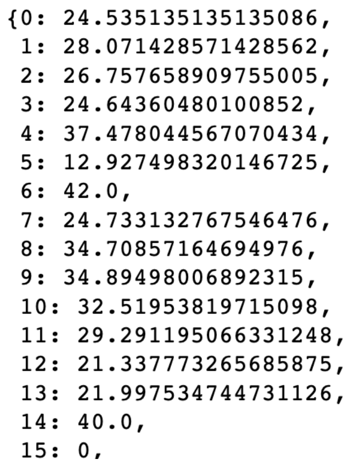


```
results=anchored_topic_model.predict(doc_word[len(documents_mod):])
results_proba,mutual_info=anchored_topic_model.predict_proba(doc_word[len(documents_mod):])

pairs=get_predict_result(results)
pairs_proba=get_predict_proba(results_proba)
```



```
def get_predict_proba(results_proba):
    pairs_proba={i:[] for i in range(len(results_proba))}
    for i in range(len(results_proba)):
        for j in range(len(results_proba[i])):
            if results_proba[i][j] >= 1e-3:
                pairs_proba[i].append((j,results_proba[i][j]))
    return pairs_proba
```

- Based on the categorization result, calculate the weighted occ value of each occupation title. Then, for each topic, add up weighted values and normalize the score. Some topics may contain only a few occupations, which might be a problem.

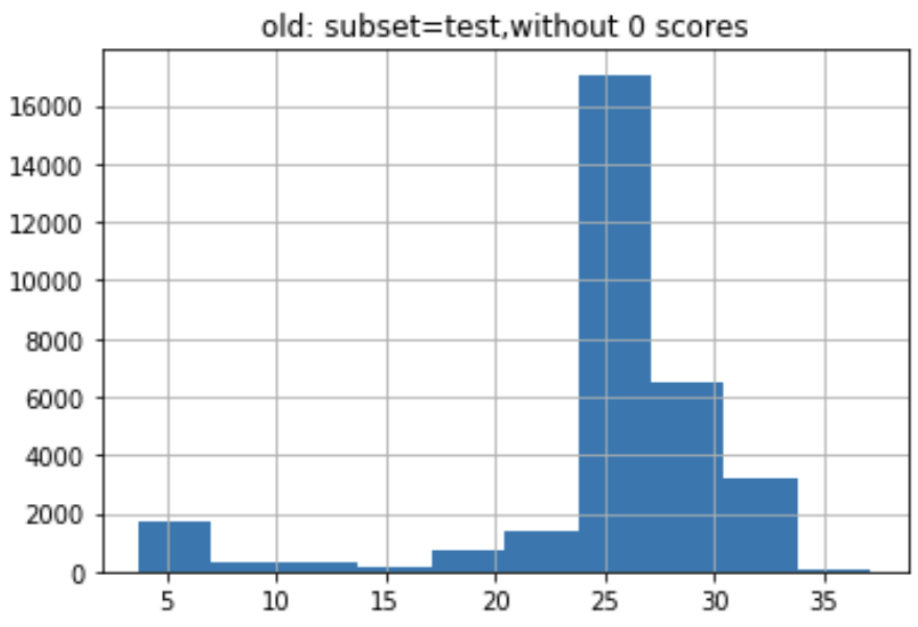


```
{0: 24.535135135135086,
 1: 28.071428571428562,
 2: 26.757658909755005,
 3: 24.64360480100852,
 4: 37.478044567070434,
 5: 12.927498320146725,
 6: 42.0,
 7: 24.733132767546476,
 8: 34.70857164694976,
 9: 34.89498006892315,
 10: 32.51953819715098,
 11: 29.291195066331248,
 12: 21.337773265685875,
 13: 21.997534744731126,
 14: 40.0,
 15: 0,
```

Calculate approximate occ score for census data:

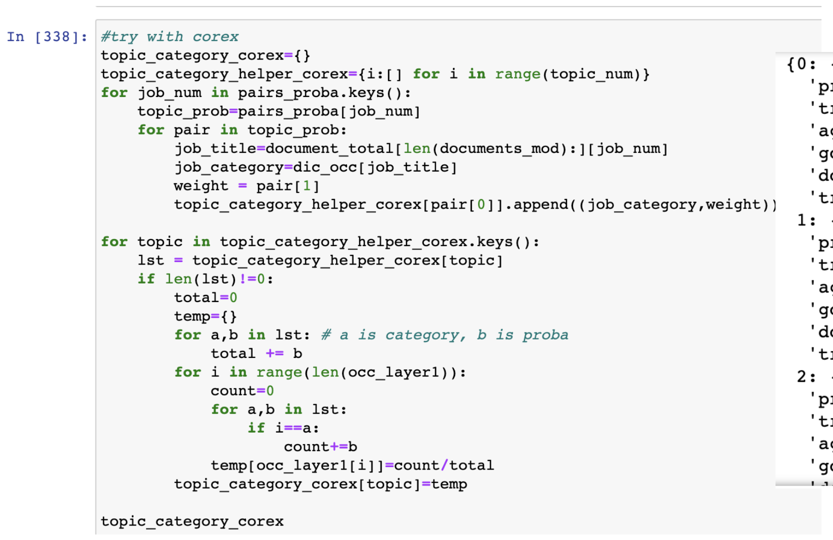
- Build a vectorizer of occupation entries:
  vectorizer = CountVectorizer(stop_words='english', max_features=20000, vocabulary=fixed_vocabulary,binary=True)
- Turn all words in occupation entries into lower cases.

- Based on the probability prediction, add up weighted values of each topic's occupation score. Normalize the result. See match_occ( ) function.
- Coverage: 0.985%. Among 32024 non-nan values, 495 occupations cannot be mapped.

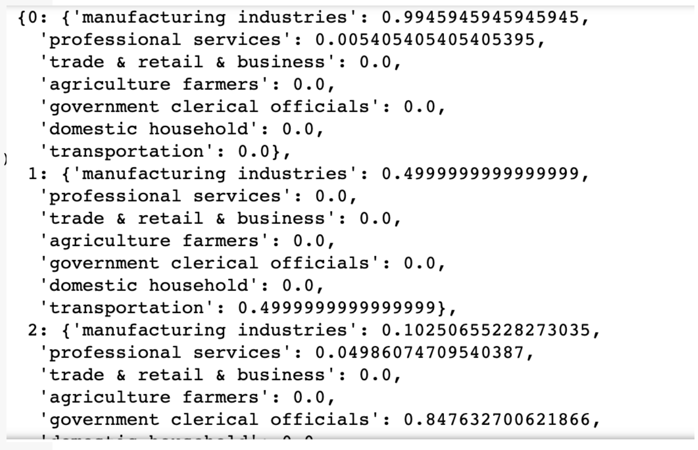


Predict Job Category

- Form a list of big job categories:
    - occ_layer1=["manufacturing industries", "professional services", "trade & retail & business", "agriculture farmers", "government clerical officials", 'domestic household', 'transportation']
- As we know the big category of each official occupation title and its classification result, we can build a dictionary recording each topic's big category. Occupation titles of different big categories may be classified into the same topic. We should calculate the proportion of each category for each topic. Within each topic, values add up to 1.



```
In [338]: #try with corex
topic_category_corex={}
topic_category_helper_corex={i:[] for i in range(topic_num)}
for job_num in pairs_proba.keys():
    topic_prob=pairs_proba[job_num]
    for pair in topic_prob:
        job_title=document_total[len(documents_mod):][job_num]
        job_category=dic_occ[job_title]
        weight = pair[1]
        topic_category_helper_corex[pair[0]].append((job_category,weight))

for topic in topic_category_helper_corex.keys():
    lst = topic_category_helper_corex[topic]
    if len(lst)!=0:
        total=0
        temp={}
        for a,b in lst: # a is category, b is proba
            total += b
        for i in range(len(occ_layer1)):
            count=0
            for a,b in lst:
                if i==a:
                    count+=b
            temp[occ_layer1[i]]=count/total
        topic_category_corex[topic]=temp

topic_category_corex
```

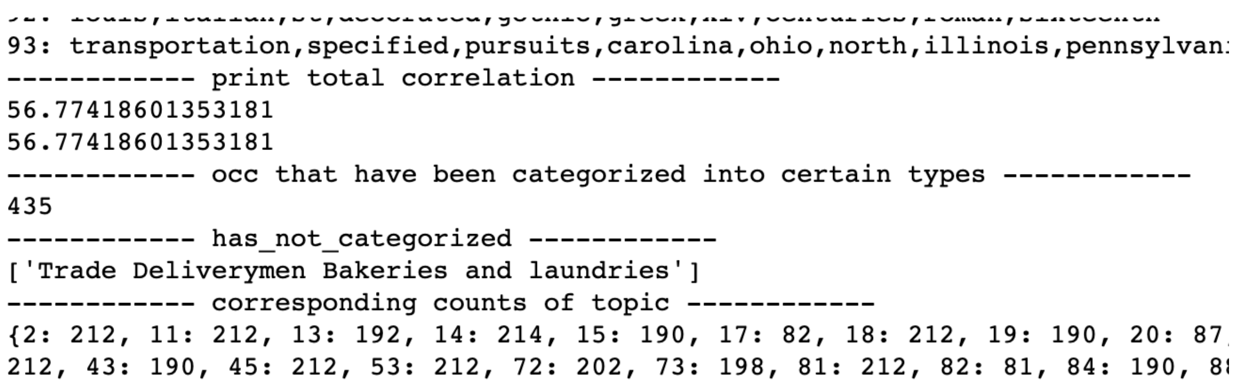


```
{0: {'manufacturing industries': 0.9945945945945945,
  'professional services': 0.005405405405405395,
  'trade & retail & business': 0.0,
  'agriculture farmers': 0.0,
  'government clerical officials': 0.0,
  'domestic household': 0.0,
  'transportation': 0.0},
 1: {'manufacturing industries': 0.49999999999999,
  'professional services': 0.0,
  'trade & retail & business': 0.0,
  'agriculture farmers': 0.0,
  'government clerical officials': 0.0,
  'domestic household': 0.0,
  'transportation': 0.49999999999999},
 2: {'manufacturing industries': 0.10250655228273035,
  'professional services': 0.04986074709540387,
  'trade & retail & business': 0.0,
  'agriculture farmers': 0.0,
  'government clerical officials': 0.847632700621866,
```

- **corex_category(occ)** maps the big category of each occupation entry in the census data. First, we get the classification probability result of each occupation. For example, for occupation o1, the result is {t1:p1, t2:p2}. For topic 1, its category proportion is {t1c1:a, t1c2:b; for topic 2, its category proportion is {t2c1:c, t2c2:d}. The category probability of that occupation would be {c1: p1*a + p2*c, c2:p1*b + p2*d}. We select the category with the highest probability.
- Coverage: 100%

Small category prediction

- Load data: three_level_occupation_scores.xlsx
- Fetch all some categories:
    - small_category=list(set(list(three_level['Small Categories'])))
    - 96 different categories
- Set 96 as the new topic number. Train a new model with the same training dataset. Set all small categories as the anchor words.
- **Surprisingly, almost every single occupation title among those 436 entries can be categorized into a topic.**



```
92: louis,italian,st,decorated,gothic,greek,xiv,centuries,roman,sixteenth
93: transportation,specified,pursuits,carolina,ohio,north,illinois,pennsylvan
------------ print total correlation ------------
56.77418601353181
56.77418601353181
------------ occ that have been categorized into certain types ------------
435
------------ has_not_categorized ------------
['Trade Deliverymen Bakeries and laundries']
------------ corresponding counts of topic ------------
{2: 212, 11: 212, 13: 192, 14: 214, 15: 190, 17: 82, 18: 212, 19: 190, 20: 87
212, 43: 190, 45: 212, 53: 212, 72: 202, 73: 198, 81: 212, 82: 81, 84: 190, 8
```

- Problems with this method:
    - Extremely detailed topics with redundant information
    - Too many topics, leading to large intersections, inconsistency, nonexclusiveness, etc.
- Predict small category of census data
    - Very poor coverage: 20822 entries cannot be classified
    - Coverage: 35%
    - **A good thing: as long as the job can be classified, the prediction result is quite accurate (by eyeballing).**
    - A thought: for occupations that can be refined into small categories, we can use the small categories to reversely deduct the large categories, making the results more accurate. The two layers can also be mutually confirmed somehow.
    - Examples:
        - City Waiter Worker Water Inspector:

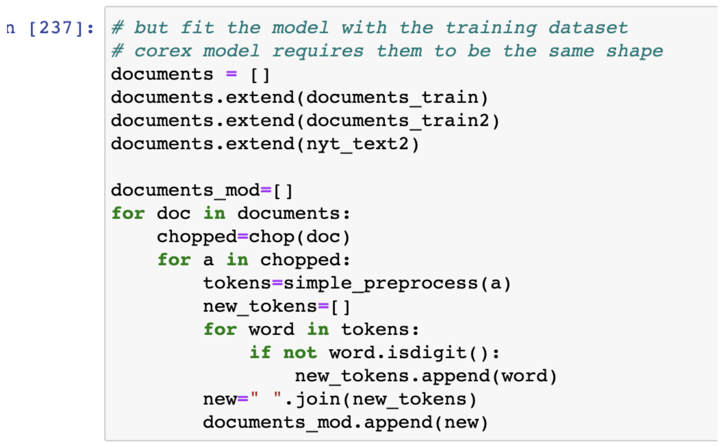
            - Water transportation, selected occupations
            - Officials and inspectors -- city and county
            - Bookkeepers, cashiers, and accountants
        - Oil Worker Stenographer
            - Oil, gas and salt well operatives
        - Housework Servant
            - Servants
        - Roofing Co Tinsmith
            - Mechanics, not otherwise specified
        - Public Dept Policeman
            - Laborers -- public service
        - Office Business Salesman
            - Messenger, bundle, and office boys -- except telegraph messengers

APIs that you can use

- If you use the same training dataset as I am using and keep the parameters (topic number, max_features of vectorizer, etc.) and preprocessing procedures the same as it displays in the code, you don't need to modify the anchor words. You can directly use the **predict_occ(file) and predict_big_job_category(file) to** process a new census dataset. Both functions return the dataframe, and you have to store it in a new csv file.
    - **predict_occ(file) will generate 3 additional columns**
        - Occupation Score Corex
        - Occupation Score Word2Vec
        - Occupation Score Merged
    - **predict_big_job_category(file) will generate 3 additional columns**
        - Occupation Category Word2Vec
        - Occupation Category Corex
        - Occupation Small Category

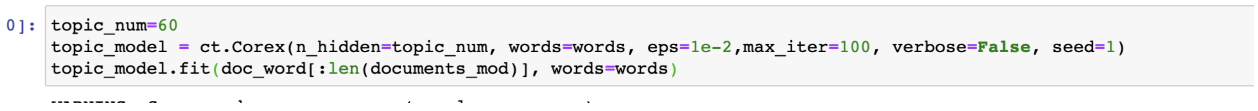
Parts that you can modify on:

- If you try to enrich the training dataset or to delete a part of it, there will be several things you should modify on:
    - Modify this part of the code. Extend your new dataset into documents and then preprocess them (chop and remove digit). Later, concatanate occ titles with other training datasets.



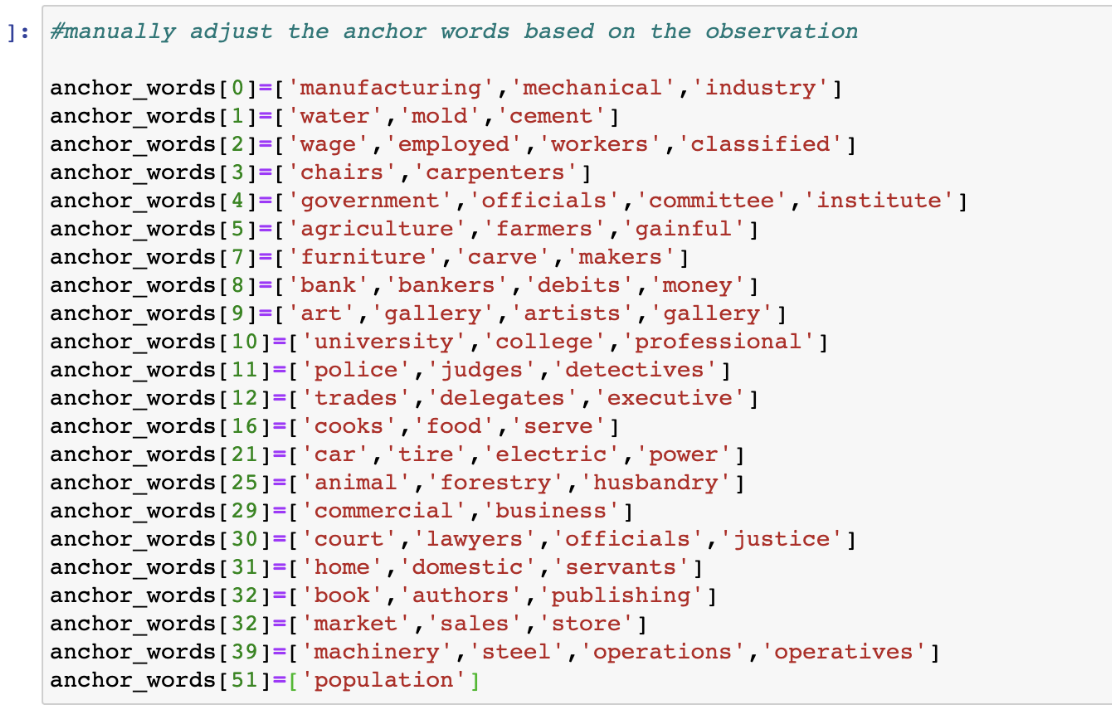
```
n [237]: # but fit the model with the training dataset
         # corex model requires them to be the same shape
         documents = []
         documents.extend(documents_train)
         documents.extend(documents_train2)
         documents.extend(nyt_text2)

         documents_mod=[]
         for doc in documents:
             chopped=chop(doc)
             for a in chopped:
                 tokens=simple_preprocess(a)
                 new_tokens=[]
                 for word in tokens:
                     if not word.isdigit():
                         new_tokens.append(word)
                 new=" ".join(new_tokens)
                 documents_mod.append(new)
```

- ○ Different training dataset will result in different topic numbers. After my exploration, I found grid search fine tuning based on the total correlation metric is not very helpful in this project. I believe 60 is a proper number which can preserve enough information and make topics exclusive from each other.



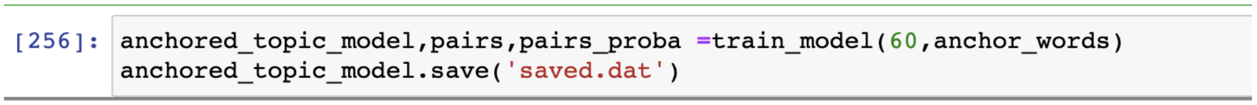
```
0]: topic_num=60
    topic_model = ct.Corex(n_hidden=topic_num, words=words, eps=1e-2,max_iter=100, verbose=False, seed=1)
    topic_model.fit(doc_word[:len(documents_mod)], words=words)
```

- ○ Semi-supervision: based on the observation of topic words, you have to modify the anchor words by manually building up word lists. After numerous attempts, I think the key point here is to identify and emphasize occupation-related words while preserving the training result of model. Try to set nouns as anchor words instead of adjectives or verbs. ‘Domestic’, ‘personal’, ‘mechanical’, etc., for sure, are good choices.



```
]: #manually adjust the anchor words based on the observation

   anchor_words[0]=['manufacturing','mechanical','industry']
   anchor_words[1]=['water','mold','cement']
   anchor_words[2]=['wage','employed','workers','classified']
   anchor_words[3]=['chairs','carpenters']
   anchor_words[4]=['government','officials','committee','institute']
   anchor_words[5]=['agriculture','farmers','gainful']
   anchor_words[7]=['furniture','carve','makers']
   anchor_words[8]=['bank','bankers','debits','money']
   anchor_words[9]=['art','gallery','artists','gallery']
   anchor_words[10]=['university','college','professional']
   anchor_words[11]=['police','judges','detectives']
   anchor_words[12]=['trades','delegates','executive']
   anchor_words[16]=['cooks','food','serve']
   anchor_words[21]=['car','tire','electric','power']
   anchor_words[25]=['animal','forestry','husbandry']
   anchor_words[29]=['commercial','business']
   anchor_words[30]=['court','lawyers','officials','justice']
   anchor_words[31]=['home','domestic','servants']
   anchor_words[32]=['book','authors','publishing']
   anchor_words[32]=['market','sales','store']
   anchor_words[39]=['machinery','steel','operations','operatives']
   anchor_words[51]=['population']
```

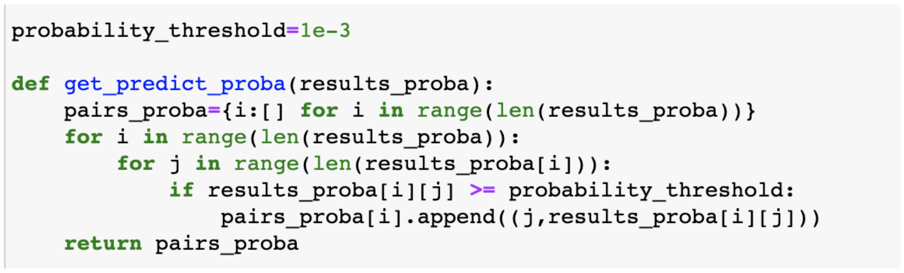
○ Train your model and save it by running the following code.



```
[256]: anchored_topic_model,pairs,pairs_proba =train_model(60,anchor_words)
       anchored_topic_model.save('saved.dat')
```

   ■ **train_model(topic_num,anchor_w=anchor_words)** function will return 3 things: anchored topic model itself, a dictionary of prediction results, and a dictionary of prediction probability. You can load an existing model by running **model=load(filename) function.**

○ As mentioned above, different probability threshold will lead to different coverage of our prediction. Corex is different from LDA: it assigns texts to certain topics with very high probability. Some topics may never appear in the training result if we use the binary prediction result to form the occ_score dictionary. I choose to use the probability dictionary for the calculation of occupation score. As a result, the choice of probability of threshold becomes a problem. A very high threshold will cause a radical concentration of classification. A very low threshold makes the calculation less convincing. 1e-3, as a low threshold, results in a very good coverage in my code.

○ 

```
probability_threshold=1e-3

def get_predict_proba(results_proba):
    pairs_proba={i:[] for i in range(len(results_proba))}
    for i in range(len(results_proba)):
        for j in range(len(results_proba[i])):
            if results_proba[i][j] >= probability_threshold:
                pairs_proba[i].append((j,results_proba[i][j]))
    return pairs_proba
```

Word2Vec as a complement

Training process

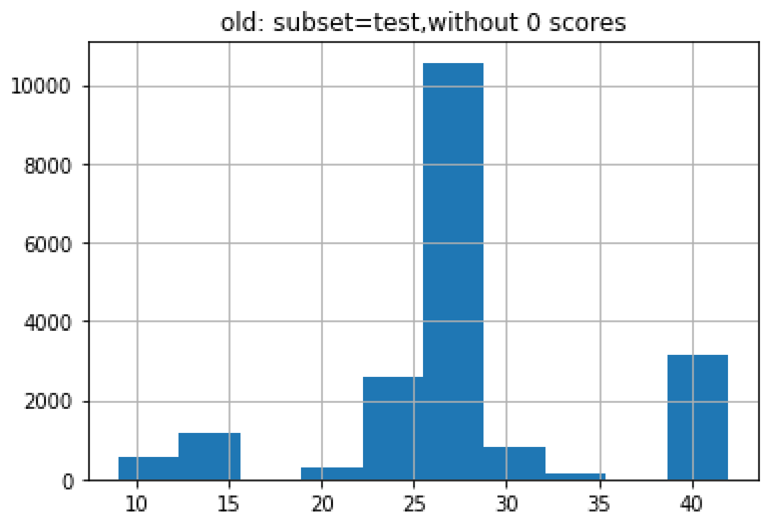
- Generate labels for each training document
- Train the word2vec model. Set size to 200.
- Create content vector matrix
    - A problem has occured while forming this matrix: one row has a different shape, which probably because its component does not appear in the training data. I just pop it to make the shape coherent.
    - Fit the matrix, X, and labels into a logistic regression model.
    - For the prediction part, generate the content vector of each x_predict by following the same logic, and get its label.

Calculate occupation score dictionary

- Based on the classification result, for each topic, add up the occ_score of occupation titles clustered into it.
- Results in a much shorter dictionary. The Word2Vec classification result is more concentrated, while corex is more scattered.

Calculate approximate occ score for census data

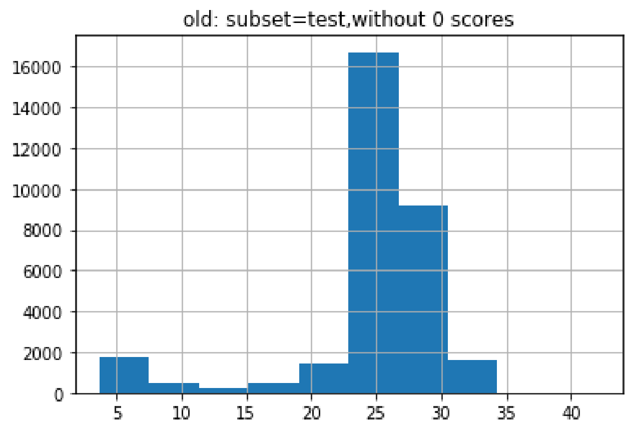
- Coverage: 0.612
    - 12727 occupations cannot be classified



- Difference between Corex_occ_score and Word2Vec_occ_score:

Full Occupation	Occupation Score	Occupation Score_Word2Vec	Occupation Score Merged
Police Dept Policeman	27.85376	28.11628	27.85376
Lof Dept Clerk	23.95357	0.00000	23.95357
Worker House Clerk	26.62601	42.00000	26.62601
Cast Co Clerk	24.29866	0.00000	24.29866
Roofing Co Tinsmith	24.09122	0.00000	24.09122
Public School Teacher	26.40179	30.92000	26.40179

-
    - Average difference between two columns: 6.91
- Merge two scores (keep corex score if both exist)
    - Coverage: 0.9952, 158 occupations among 32771 cannot be classified



- What about calculating the average of these two values?

Predict Job Category

- Same as Corex. Logic is much simpler. Eliminate the probability part.
- Coverage: 0.6116
- Compare result

Full Occupation	Occupation Score	Occupation Score_Word2Vec	Occupation Score Merged	Occupation Category Word2Vec	Occupation Category Corex
Steam Shop Housekeeper	6.45076	15.00000	6.45076	professional services	manufacturing industries
Selow Small Book Keeper	22.55748	28.33333	22.55748	manufacturing industries	manufacturing industries
Express Driver	0.00000	28.11628	28.11628	transportation	manufacturing industries
Hotel Foreman	27.85340	28.11628	27.85340	transportation	transportation
Saloon Bar Tender	27.85344	30.92000	27.85344	professional services	transportation
Unemployed Student	23.95357	30.92000	23.95357	professional services	manufacturing industries
Contractor Agent	23.95357	28.11628	23.95357	transportation	manufacturing industries
Baber Wagon Driver	23.95357	42.00000	23.95357	trade & retail & business	manufacturing industries
Factory Embroiderer	23.95357	25.00000	23.95357	government clerical officials	manufacturing industries
House Builder	26.62600	28.11628	26.62600	transportation	manufacturing industries

Concerns

- As we combine the industry field with occupation field to form a new column called full occupation, we should consider the question whether this combination will affect the classification result. Do we need to assign different weights to occupation and industry when calculating occupation score and predict its big job category?
    - I applied both functions only on the occupation field and found the results are not satisfying. Especially for word2vec, many occupations cannot be categorized due to the usage of one-hot encoding method in word2vec training process.
- If we apply the same logic to the sub-category prediction, it is theoretically possible. As we know the sub-category of each occupation title, we can add up the weighted value for each one of topics to calculate the sub-category proportion. However, there are two main problems in this method:
    - The dataset is not large enough to gain complete information about the sub-category of occupations.
    - Too many categories will be contained in a single topic, and each one of them has a small proportion, which will lead to messy results.

  A better way is to set all these small job categories as anchor words directly.