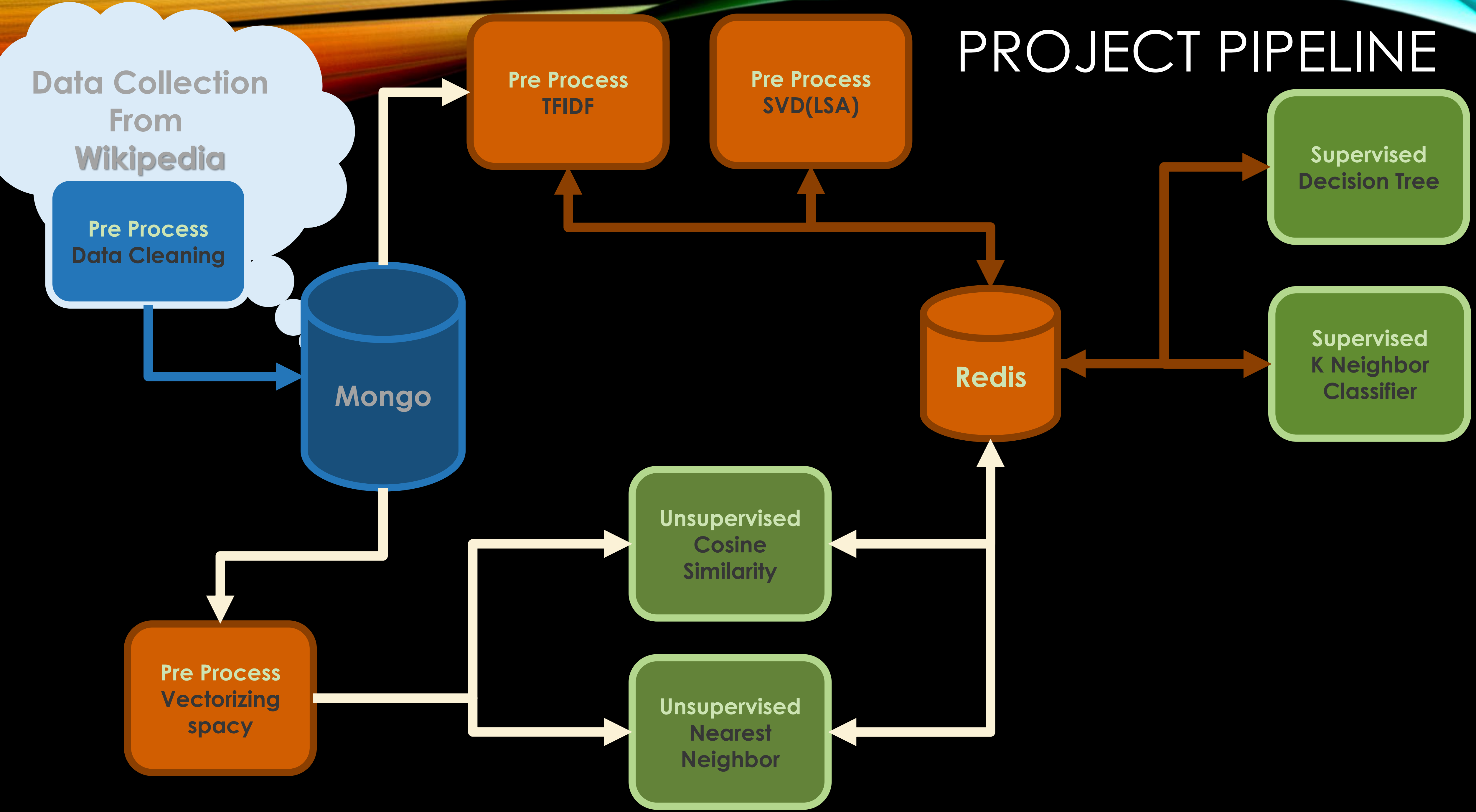
PROJECT 4
NATURAL LANGUAGE PROCESSING USING WIKIPEDIA API

Joyce Lin, General assembly, data science immersive

PROJECT PIPELINE



DATA COLLECTION

```python
def qry_category(category):
    new_str = re.sub('\s','+', category)
    r = requests.get('http://en.wikipedia.org/w/api.php?\
                     action=query&\
                     format=json&\
                     list=categorymembers&\
                     cmtitle=Category%3A+{}&\
                     cmlimit=max'.format(new_str))
    re.sub('\s','+', category)
    return pd.DataFrame(r.json()['query']['categorymembers'])
```

1. Used **Wikimedia API** search query to request page id and title

2. Collect page title from sub-categories using **recursive** method

```python
def get_page_df(category):
    ml_df = qry_category(category)
    pages_list=[]
    page_df = ml_df[~ml_df['title'].str.contains('Category:')]
    page_df['category'] = category
    pages_list.append(page_df)
    category_df = ml_df[ml_df['title'].str.contains('Category:')]
    categories = category_df[category_df['title'].\
                             str.contains('Category:')]['title'].str.replace('Category:',"")
    if (category_df.shape[0]>0):
    #if (category_df.shape[0]>0 and category_df.shape[0]<30):
        try:
            for category in categories:
                pages_list.append(get_page_df(category))
                if VERBOSE: print ('current_category: {}'.format(category))
        except KeyError:
            pass
    pages_df = pd.concat(pages_list)
    pages_df = pages_df.sort_values(by='title', axis=0, ascending=True)
    return pages_df
```

DATA COLLECTION

```
! pip install wikipedia
import wikipedia
```

```
def insert_to_wiki_mongo(Category='main page'):
    df = get_page_df(Category)
    dict_list = []
    for i in df['pageid']:
        try:
            page= wikipedia.page(pageid=i)
            if VERBOSE: print(page.pageid,page.title)
            dict_ = {
                "pageid":page.pageid,
                "title":page.title,
                "content":page.content,
                "category":Category,
                   }

            wiki_mongo_collection.update_one(dict_,{'$set': dict_}, upsert=True)
        except AttributeError:
            pass
        except ValueError:
            print(dict_)
            raise
turn cli.database_names(), wiki_mongo_collection.count()
```

1. Use **wikipedia library** to pull page content based on page title

2. Insert each **document** into my **wiki Mongo Collection**

```
wiki_df[['pageid']].\
    groupby(wiki_df['category']).count()
```

category	pageid
Business intelligence	1174
Geographic information systems	470
Microsoft Office	109
Petroleum	1414
Statistical_methods	345
Taiwan	582
Yoga	494
machine learning	1077

PREPROCESSING - DATA CLEANING

1. Clean article content with **regex**
2. Tokenize content into text using **lemmatization** method for removing **stop words**
3. Save cleaned content back to my **wiki mongo collection**

```
import re
from spacy.en import STOP_WORDS
from spacy.en import English
nlp = English()
```

```
for i in range(wiki_mongo_collection.count()):
    cur = wiki_mongo_collection.find()[i]
    content = cur['content']
    cleaned_content = cleaner(content)
    wiki_mongo_collection.update_one(
        {'content':content},
        {'$set': {'clean_content':cleaner(content)}}
        )
```

```
_mongo_collection.delete_many({'title':{'$regex':"File:.*"}})
ki_mongo_collection.delete_many({"clean_content":''})
wiki_mongo_collection.delete_many( { 'clean_content': None })
```

```
wiki_mongo_collection.count()
```

```
5665
```

```
def cleaner(text):
    text = text.lower()
    text = re.sub('<.{0,3}>',' ',text)
    text = re.sub('{.*\.*}',' ',text)
    text = re.sub('{.*\+.*}',' ',text)
    #text = re.sub('[\W]',' ',text)
    text = re.sub('[^a-z]',' ',text)
    text = ' '.join( [w for w in text.split() if len(w)>1] )
    #text = re.sub('/(?<!\S).(?!\S)\s*/',' ',text)
    text = re.sub('aa','a',text)
    text = re.sub('[0-9]','',text)
    text = re.sub('\s+',' ',text)
    text = ' '.join([i.lemma_ for i in nlp(text)
                    if i.orth_ not in STOP_WORDS])
eturn text
```

PREPROCESSING - TIFIDF

	title	category	content	clean_content
0	(1+ε)-approximate nearest neighbor search	machine learning	(1+ε)-approximate nearest neighbor search is a...	approximate near neighbor search special case ...

```
tfidf_vectorizer = TfidfVectorizer(min_df = 3, stop_words = 'english')
```

```
wiki_tfidf = tfidf_vectorizer.\
    fit_transform(wiki_df['clean_content'].values.astype('U'))
```

```
wiki_tfidf_df = pd.DataFrame(wiki_tfidf.toarray(),
                index = wiki_df.index,
                columns = tfidf_vectorizer.get_feature_names())
```

```
len(wiki_tfidf_df.columns)
```

```
30197
```

1. Get bag of words using TIFIDF
2. Dump table into redis

	aa	aal	aas	ab	aba	abacus	abadan	abandon	abandonment	abba	...	zuben	zuckerberg
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

```
r = redis.StrictRedis(redis_ip)
tfidf = pickle.dumps(wiki_tfidf_df)
r.set('wiki_tfidf', tfidf)
r.keys()
```

Text Frequency * Inverse Document Frequency

PREPROCESSING - LSA

1. Fit and transform TIFIDF dataframe to truncated SVD model
2. N_components = 800

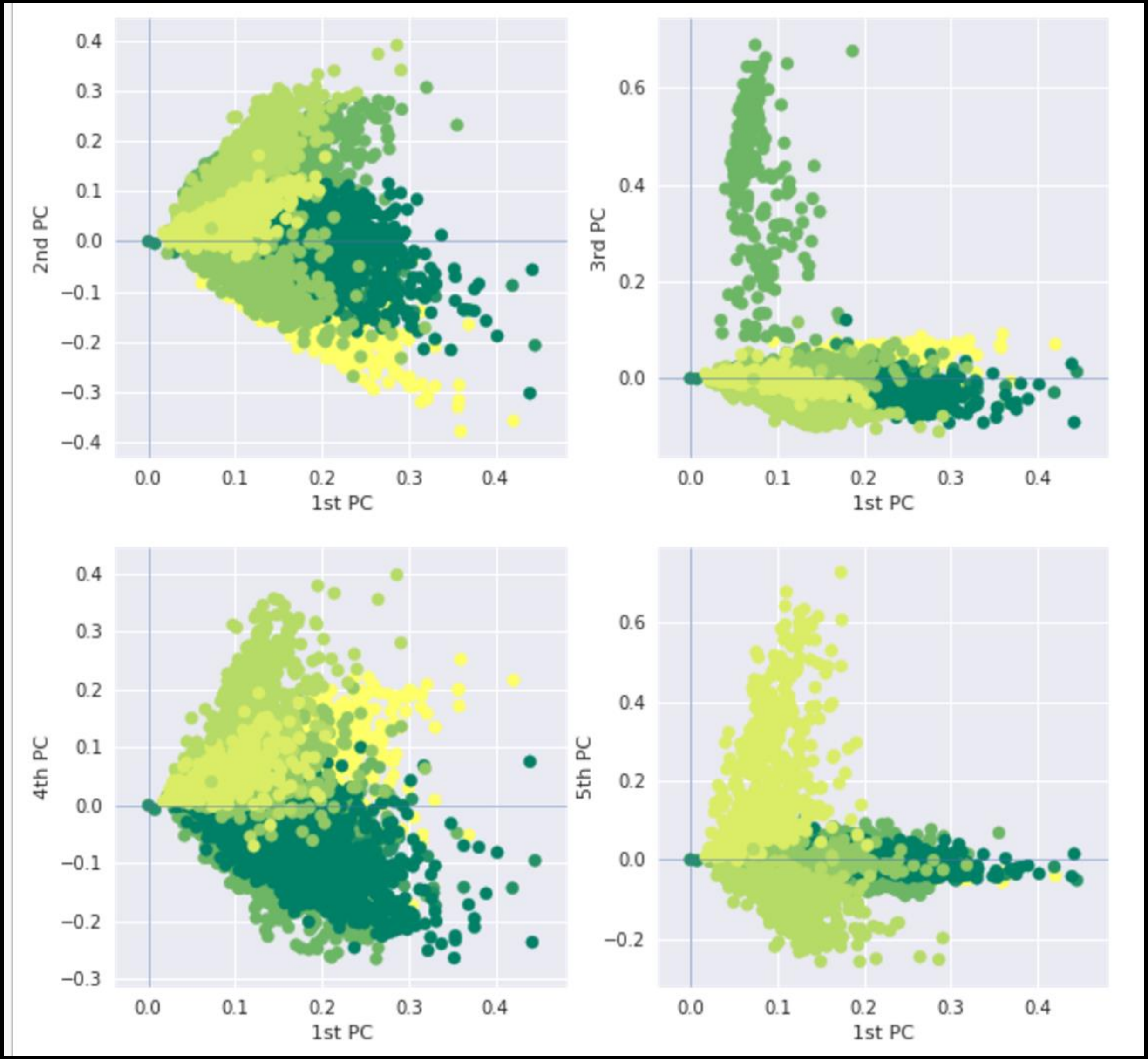
SVD (Latent Semantic Analysis)

```
def perform_LSA(n_components, vectorizer, df):
    SVD = TruncatedSVD(n_components)
    component_names = ["component_"+str(i+1) for i in range(n_components)]
    latent_semantic_analysis = pd.DataFrame(SVD.fit_transform(df),
                                             index = df.index,
                                             columns = component_names)
    vocabulary_expression = pd.DataFrame(SVD.components_,
                                          index = component_names,
                                          columns = vectorizer.get_feature_names())
    return latent_semantic_analysis, vocabulary_expression
```

```
latent_semantic_analysis, \
vocabulary_expression \
        = perform_LSA(800, tfidf_vectorizer,wiki_tfidf_df)
```

```
latent_semantic_analysis[['category','title','component_1','component_2']].sample(10,
```

	category	title	component_1	component_2
5505	Yoga	Anti-gravity yoga	0.092886	0.044423
739	machine learning	OpenNN	0.215921	-0.126171
3820	Business intelligence	Mokken scale	0.093972	-0.072468
360	machine learning	Gaussian process	0.185722	-0.190889



PREDICTION - SUPERVISED

```
#  x train
LSA_components = LSA.drop(['category','title','content'], axis = 1)
```

```
# y train
cat = LSA['category']
cat_dummies = pd.get_dummies(cat)
```

```
dtc = DecisionTreeClassifier()
knc = KNeighborsClassifier()
```

```
gs_dtc = GridSearchCV(dtc, param_grid=param_dtc,cv=5,n_jobs = -1)
gs_dtc.fit(LSA_components, cat_dummies)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'max_depth': [3, 6, 9, 12], 'min_samples_leaf': [2, 5, 10, 15]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

```
gs_dtc_res_df = pd.DataFrame(gs_dtc.cv_results_)
gs_dtc_res_df.sort('rank_test_score',ascending=True).head(5).T[1:6]
```

	9	13	10	14	8
mean_test_score	0.526037	0.52286	0.505384	0.505031	0.503795
mean_train_score	0.928861	0.938658	0.909003	0.911695	0.9406
param_max_depth	9	12	9	12	9
min_samples_leaf	5	5	10	10	[illegible]

```
gs_knc = GridSearchCV(knc, param_grid=param_knc,cv=5,n_jobs = -1)
gs_knc.fit(LSA_components, cat_dummies)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'n_neighbors': [5, 10, 15], 'leaf_size': [20, 30]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

```
gs_knc_res_df = pd.DataFrame(gs_knc.cv_results_)
gs_knc_res_df.sort('rank_test_score',ascending=True).head(5).T[2:6]
```

	0	3	1	4	
mean_test_score	0.302383	0.302383	0.230362	0.230362	0.2225[illegible]
mean_train_score	0.875375	0.875375	0.775861	0.775861	0.76955
param_leaf_size	20	30	20	30	20
neighbors	5	5	10	10	15

PREDICTION – UNSUPERVISED (NEAREST NEIGHBOR)

NLP - Nearest Neighbor

```
from spacy.en import English
nlp = English()
content_vec = nlp(df['clean_content'][0]).vector
content_vec.shape
```

```
(300,)
```

```
content_vecs = np.array([nlp(i).vector for i in df['clean_content']])
content_vecs.shape
```

```
(5665, 300)
```

```
from sklearn.neighbors import NearestNeighbors
nn = NearestNeighbors(n_neighbors=5)
n.fit(content_vecs)
```

```
stNeighbors(algorithm='auto', leaf_size=30, metric='minkowski',
    metric_params=None, n_jobs=1, n_neighbors=5, p=2, radius=1.0)
```

```
def most_similar(search):
    distance, indices = nn.kneighbors(nlp(search).vector.reshape(1,-1))

    return df.ix[indices[0]][['category','title','clean_content']]
```

```
most_similar('map')
```

	category	title	clean_content
1426	Geographic information systems	Internet Map Server	internet map server ims provide map internet u...
1290	Geographic information systems	Earth Point	earthpoint map southwest idaho real estate lis...
1460	Geographic information systems	Map regression	map regression process work backwards later ma...
?71	Geographic information systems	David Rumsey Historical Map Collection	david rumsey historical map collection world l...
	Geographic information systems	Google Maps	google map web mapping service develop google ...

PREDICTION – UNSUPERVISED COSINE SIMILARITY

1. Combine Wikipedia page content from a single category into one string

```
wiki_cat_df[['category','text']].head()
```

	category	text
0	machine learning	approximate near neighbor search special case...
1	Microsoft Office	analyse statistical analysis add microsoft ex...
2	Geographic information systems	world atlas virtual globe program develop cos...
3	Petroleum	avenue americas know news corp building inter...
4	Yoga	adho mukha sana adho mukha shvanasana ipa muk...

2. Vectorize the "category string" as well as the " page string"

```
cat_vecs = np.array([nlp(i).vector for i in wiki_cat_df['text']])
```

```
content_vecs = np.array([nlp(i).vector for i in wiki_df['clean_content']])
```

PREDICTION – UNSUPERVISED COSINE SIMILARITY

Get COSINE SIMILARITY SCORE by "strings"

```
import redis
redis_ip = '34.210.97.79'
content_vecs = pickle.loads(r.get('nlp_content_vecs'))
cat_vecs = pickle.loads(r.get('nlp_cat_vecs'))
def Similarity_score(text):
    wiki_cat_df = pd.DataFrame(list(cli.wiki_mongo_database.wiki_cat_collection.find({})))
    #cat_vecs = np.array([nlp(i).vector for i in wiki_cat_df['text']])
    r = redis.StrictRedis(redis_ip)
    cat_vecs = pickle.loads(r.get('nlp_cat_vecs'))
    similarity_score={}
    for j in range(len(cat_vecs)):
        similarity = cosine_similarity(cat_vecs[j].reshape(1,-1),nlp(text).vector.reshape(1,-1))
        similarity_score[(wiki_cat_df['category'][j])] = round(similarity[0][0],3)
        cs_s_df = pd.DataFrame.from_dict(similarity_score,orient='index')
        cs_s_df.columns = ['score']
        cs_s_df = cs_s_df.sort_values('score',ascending=False)
    return cs_s_df.head(3)
```

```
Similarity_score('lake')
```

	score
Petroleum	0.418
Taiwan	0.402
Yoga	0.379

PREDICTION – UNSUPERVISED COSINE SIMILARITY

```python
wiki_cos_sim_collection.drop()
dict_cos_sim={}
import redis
redis_ip = '34.210.97.79'
content_vecs = pickle.loads(r.get('nlp_content_vecs'))
cat_vecs = pickle.loads(r.get('nlp_cat_vecs'))

for i in range(len(content_vecs)):
    dict_cos_sim['title']=  wiki_df['title'][i]
    dict_cos_sim['category']= wiki_df['category'][i]
    for j in range(len(cat_vecs)):
        similarity = cosine_similarity(cat_vecs[j].reshape(1,-1),content_vecs[i].reshape(1,-1))
        dict_cos_sim['{}'.format(wiki_cat_df['category'][j])]= round(similarity[0][0],3)
    wiki_cos_sim_collection.update_one(dict_cos_sim,{'$set': dict_cos_sim}, upsert=True)
```

```python
cos_df['predicted_cat'] = 0
for i in range(len(cos_df)):
    cos_df['predicted_cat'][i] = cos_df.iloc[i, 2:-1].sort_values(ascending=False).index[0]
```

	category	title	machine learning	Microsoft Office	Geographic information systems	Petroleum	Yoga	Business intelligence	Taiwan	Statistical_methods	predicted_cat
0	machine learning	(1+ε)-approximate nearest neighbor search	0.862	0.821	0.862	0.849	0.839	0.852	0.840	0.858	Geographic information systems
1	machine learning	ADALINE	0.964	0.910	0.932	0.845	0.841	0.910	0.822	0.952	machine learning
2	machine learning	AI@50	0.928	0.885	0.924	0.921	0.920	0.937	0.905	0.916	Business intelligence
3	machine learning	AIVA	0.875	0.845	0.875	0.856	0.879	0.869	0.835	0.855	Yoga
4	machine learning	AIXI	0.967	0.892	0.934	0.912	0.921	0.950	0.894	0.964	machine learning

1. Run Cosine Similarity score on Wikipedia page content again each category
2. Assign the category that gets the highest cosine similarity score as predicted category by each page
3. Cosine Similarity Accuracy = (right prediction)/ (total predicted)

```python
print('pridiction score with Cosine Similarity:',
      sum(cos_df['category'] == cos_df['predicted_cat'])/len(cos_df))
```

```
pridiction score with Cosine Similarity: 0.704148278906
```

PROJECT LEARNED

- Be mindful on using recursive method
  - Better to pull one article and insert into database at a time
- Memory issue
- There is no true test/ train split on this prediction model

FURTHER STUDY

- Build python script for data collection
- Error handing on memory issue
- Further tuning on LSA n_component