Week 11 - Text Topic Models using Python

This is an example of how to identify topics in text material and then integrate with other data to develop models for structured data. The basic approach is to first develop topic groups and identify each observation with one of these groups. The final step is to integrate these topic assignments with other data to help develop models for forecasting or classification. In this process, we can develop a deeper understanding of the content of the documents or reviews and use that information to make better forecasts.

Data

The data for this example consists of N=11,717 reviews of California Chardonnay. It contains not only the reviews but also other information about the wine, e.g. price, year, points, etc. However, in this example, only the review, found in the **description** attribute will be used to prepare a cluster analysis of the reviews.

Import Packages

In [1]:

```
#  classes provided for the course
from Class_replace_impute_encode import ReplaceImputeEncode
from Class_regression import linreg
from sklearn.linear_model import LinearRegression

import pandas as pd
import numpy  as np
import string
import nltk
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF
```

Download NLTK Supporting Files

The **NLTK** package uses several supporting files. These need to be downloaded, but only once. Download them initially using the following statements. After these execute successfully, comment them out of your code.

In [2]:

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /Users/Home/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/Home/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package stopwords to /Users/Home/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/Home/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[2]:

True

The User Functions

The following functions will be used to customize the parse, pos, stop, stem process necessary for text analysis. These are done using the **NLTK** package, customized to remove certain words and symbols, and handle synonyms.

In [3]:

```
# my_analyzer replaces both the preprocessor and tokenizer
# it also replaces stop word removal and ngram constructions

def my_analyzer(s):
    # Synonym List
    syns = {'cab': 'cabernet', \
              'veh': 'vehicle', 'car': 'vehicle', 'chev':'cheverolet', \
              'chevy':'cheverolet', 'air bag': 'airbag', \
              'seat belt':'seatbelt', "n't":'not', 'to30':'to 30', \
              'wont':'would not', 'cant':'can not', 'cannot':'can not', \
              'couldnt':'could not', 'shouldnt':'should not', \
              'wouldnt':'would not', 'straightforward': 'straight forward' }

    # Preprocess String s
    s = s.lower()
    # Replace special characters with spaces
    s = s.replace('-', ' ')
    s = s.replace('_', ' ')
    s = s.replace(',', '. ')
    # Replace not contraction with not
    s = s.replace("'nt", " not")
    s = s.replace("n't", " not")
    # Tokenize
    tokens = word_tokenize(s)
    #tokens = [word.replace(',','') for word in tokens ]
    tokens = [word for word in tokens if ('*' not in word) and \
              ("''" != word) and ("``" != word) and \
              (word!='description') and (word !='dtype') \
```

```
            and (word != 'object') and (word!="'s")]

    # Map synonyms
    for i in range(len(tokens)):
        if tokens[i] in syns:
            tokens[i] = syns[tokens[i]]

    # Remove stop words
    punctuation = list(string.punctuation)+['..', '...']
    pronouns = ['i', 'he', 'she', 'it', 'him', 'they', 'we', 'us', 'them']
    others   = ["'d", "co", "ed", "put", "say", "get", "can", "become",\
                "los", "sta", "la", "use", "iii", "else"]
    stop = stopwords.words('english') + punctuation + pronouns + others
    filtered_terms = [word for word in tokens if (word not in stop) and \
                  (len(word)>1) and (not word.replace('.','',1).isnumeric()) \
                  and (not word.replace("'",'',2).isnumeric())]

    # Lemmatization & Stemming - Stemming with WordNet POS
    # Since lemmatization requires POS need to set POS
    tagged_words = pos_tag(filtered_terms, lang='eng')
    # Stemming with for terms without WordNet POS
    stemmer = SnowballStemmer("english")
    wn_tags = {'N':wn.NOUN, 'J':wn.ADJ, 'V':wn.VERB, 'R':wn.ADV}
    wnl = WordNetLemmatizer()
    stemmed_tokens = []
    for tagged_token in tagged_words:
        term = tagged_token[0]
        pos  = tagged_token[1]
        pos  = pos[0]
        try:
            pos   = wn_tags[pos]
            stemmed_tokens.append(wnl.lemmatize(term, pos=pos))
        except:
            stemmed_tokens.append(stemmer.stem(term))
    return stemmed_tokens

def display_topics(lda, terms, n_terms=15):
    for topic_idx, topic in enumerate(lda):
        if topic_idx > 8:
            break
        message  = "Topic #%d: " %(topic_idx+1)
        print(message)
        abs_topic = abs(topic)
        topic_terms_sorted = \
                [[terms[i], topic[i]] \
                     for i in abs_topic.argsort()[:-n_terms - 1:-1]]
        k = 5
        n = int(n_terms/k)
        m = n_terms - k*n
```

```
        for j in range(n):
            l = k*j
            message = ''
            for i in range(k):
                if topic_terms_sorted[i+l][1]>0:
                    word = "+"+topic_terms_sorted[i+l][0]
                else:
                    word = "-"+topic_terms_sorted[i+l][0]
                message += '{:<15s}'.format(word)
            print(message)
        if m> 0:
            l = k*n
            message = ''
            for i in range(m):
                if topic_terms_sorted[i+l][1]>0:
                    word = "+"+topic_terms_sorted[i+l][0]
                else:
                    word = "-"+topic_terms_sorted[i+l][0]
                message += '{:<15s}'.format(word)
            print(message)
        print("")
    return
```

Read Document

The following code reads the document and places its contents into a Pandas dataframe **df**. It also sets up some constants used later in the code.

In [12]:

```
# Increase column width to let pandy read large text columns
pd.set_option('max_colwidth', 32000)
# Read N=13,575 California Cabernet Savignon Reviews
file_path = '/Users/Home/Desktop/python/Excel/'
df = pd.read_excel(file_path+"CaliforniaCabernet.xlsx")

# Setup program constants and reviews
n_reviews  = len(df['description'])
s_words    = 'english'
ngram = (1,2)
reviews = df['description']

# Constants
m_features      = None # default is None
n_topics        =  9   # number of topics
max_iter        = 10   # maximum number of iterations
max_df          = 0.5  # max proportion of docs/reviews allowed for a term
learning_offset = 10.      # default is 10
learning_method = 'online' # alternative is 'batch' for large files
tf_matrix='tfidf'
```

Create Term/Doc Matrix using Custom Analyzer

Notice the use of **CountVectorizer**, but it's calling the custom text analysis function **my_analyzer**. This method creates the term/doc matrix of term frequencies. In this example it is score in **tf**. This will be a *CSR*, Compressed Sparce Row matrix, since so many of it's elements will be zero. **tf** will only contain the nonzero entries.

If you want to see those entries, use the following function **tf[0].nonzero()** to examine the nonzero values for the first row. It is unusual, but *sklearn* does not create the traditional term/doc matrix where the rows are the terms and the columns are the documents. Instead *sklearn* returns the doc/term matrix where the rows and columns are transposed.

As a result *tf[i].nonzero()* displays the nonzero terms for the ith document.

In [5]:

```
# Create the Review by Term Frequency Matrix using Custom Analyzer
# max_df is a limit for terms.  If a term has more than this
# proportion of documents then that term is dropped.  Use max_df=1.0
# to eliminate this behavior.  Typical values are max_df between 0.5 and 0.95
cv = CountVectorizer(max_df=max_df, min_df=2, max_features=m_features,\
                     analyzer=my_analyzer)
tf    = cv.fit_transform(reviews)
terms = cv.get_feature_names()
print('{:.<22s}{:>6d}'.format("Number of Reviews", len(reviews)))
print('{:.<22s}{:>6d}'.format("Number of Terms",   len(terms)))

term_sums = tf.sum(axis=0)
term_counts = []
for i in range(len(terms)):
    term_counts.append([terms[i], term_sums[0,i]])
def sortSecond(e):
    return e[1]
term_counts.sort(key=sortSecond, reverse=True)
print("\nTerms with Highest Frequency:")
for i in range(10):
    print('{:<15s}{:>5d}'.format(term_counts[i][0], term_counts[i][1]))
```

```
Number of Reviews..... 13135
Number of Terms.......  5611

Terms with Highest Frequency:
wine           7439
tannin         5134
cherry         5123
oak            4670
black          4596
currant        4404
dry            4143
fruit          3543
rich           2947
drink          2929
```

TF-IDF

Next, transform the TF matrix from term counts to term counts weighted by the IDF, Inverse Document Frequency: **IDF(i) = log(d/d(i))** where i is the ith term and d is the total number of documents and d(i) is the number for the ith term.

In [6]:

```
# Construct the TF/IDF matrix from the Term Frequency matrix
print("\nConstructing Term/Frequency Matrix using TF-IDF")
# Default for norm is 'l2', use norm=None to supress
tfidf_vect = TfidfTransformer(norm=None, use_idf=True) #set norm=None
# tf matrix is (n_reviews)x(m_terms)
tf = tfidf_vect.fit_transform(tf)

# Display the terms with the largest TFIDF value
term_idf_sums = tf.sum(axis=0)
term_idf_scores = []
for i in range(len(terms)):
    term_idf_scores.append([terms[i], term_idf_sums[0,i]])
print("The Term/Frequency matrix has", tf.shape[0], " rows, and",\
      tf.shape[1], " columns.")
print("The Term list has", len(terms), " terms.")
term_idf_scores.sort(key=sortSecond, reverse=True)
print("\nTerms with Highest TF-IDF Scores:")
for i in range(10):
    j = i
    print('{:<15s}{:>8.2f}'.format(term_idf_scores[j][0], \
          term_idf_scores[j][1]))
```

```
Constructing Term/Frequency Matrix using TF-IDF
The Term/Frequency matrix has 13135  rows, and 5611  columns.
The Term list has 5611  terms.

Terms with Highest TF-IDF Scores:
wine           12619.14
tannin         10080.72
cherry         10070.38
black           9932.89
oak             9651.13
currant         9241.95
dry             9119.30
fruit           8436.35
rich            7429.52
drink           7390.27
```

SVD - Singular Value Decomposition

Now that the tf matrix is constructed, perform a singular value decomposition of the matrix to identify the topic groups. SVD on the TFIDF matrix is referred to as LSA, *Latent Semantic Analysis.*

In [7]:

```
# In sklearn, SVD is synonymous with LSA (Latent Semantic Analysis)
uv = TruncatedSVD(n_components=n_topics, algorithm='arpack',\
                                    tol=0, random_state=12345)

U = uv.fit_transform(tf)
# Display the topic selections
print("\n********** GENERATED TOPICS **********")
display_topics(uv.components_, terms, n_terms=15)

# Store topic selection for each doc in topics[]
topics = [0] * n_reviews
for i in range(n_reviews):
    max       = abs(U[i][0])
    topics[i] = 0
    for j in range(n_topics):
        x = abs(U[i][j])
        if x > max:
            max = x
            topics[i] = j
```

```
********** GENERATED TOPICS **********
Topic #1:
+wine          +black         +tannin        +oak           +currant
+cherry        +dry           +year          +fruit         +rich
+ripe          +show          +drink         +chocolate     +good

Topic #2:
-year          +palate        +body          +nose          +full
+black         +pepper        +plum          +finish        +red
+tobacco       -next          +aroma         -good          -develop

Topic #3:
-sweet         -soft          -cherry        +year          -drink
+cellar        -ripe          -like          +vineyard      +mountain
+black         +time          -oak           +age           +verdot

Topic #4:
-dry           -black         -good          +verdot        +new
+petit         -currant       +oak           +soft          +merlot
-show          +blend         -tannic        -cedar         +sweet

Topic #5:
-fruit         +dry           +napa          -palate        -nose
+verdot        +petit         +valley        +blend         +merlot
-dark          +cedar         +currant       +price         +good

Topic #6:
-chocolate     +body          -show          +wine          +full
-nose          -bottle        +tannic        +taste         -black
+like          +good          -dark          -blueberry     +dry
```

```
Topic #7:
-full         -body         +verdot        +petit         +cherry
+merlot       +blend        +franc         +nose          -dark
-cassis       +malbec       +little        +bottle        -chocolate

Topic #8:
+black        +dark         +chocolate     +currant       -valley
-wine         +full         +verdot        +petit         -fruit
-napa         -well         +smoky         +year          +body

Topic #9:
-year         -next         -develop       +black         -body
-bottle       -full         +chocolate     +currant       -complexity
-six          +time         +dark          -red           +like
```

Save Topic Scores

In [20]:

```
#*********** Calculate Topic Scores for Each Document *************
rev_scores = []
for i in range(n_reviews):
    u = [0] * (n_topics+1)
    u[0] = topics[i]
    for j in range(n_topics):
        u[j+1] = U[i][j]
    rev_scores.append(u)

# Setup review topics in new pandas dataframe 'df_rev'
cols = ["topic"]
for i in range(n_topics):
    s = "T"+str(i+1)
    cols.append(s)
df_rev = pd.DataFrame.from_records(rev_scores, columns=cols)
df     = df.join(df_rev)
```

Display Associations of Topics with Average Price and Points

In [22]:

```
# Data Map for these Reviews
attribute_map = {
    'Review':[3,(1, 14000),[0,0]],
    'description':[3,(''),[0,0]],
    'year':[3,(1900,2020),[0,0]],
    'points':[0,(1, 100),[0,0]],
    'price':[0,(1, 3000),[0,0]],
    'winery':[3,(''),[0,0]],
    'Region':[2,('South Coast', 'Sonoma', 'Sierra Foothills',    \
                 'Redwood Valley', 'Red Hills Lake County',      \
```

```
                 'North Coast', 'Napa-Sonoma', 'Napa',            \
                 'Mendocino/Lake Counties', 'Mendocino Ridge',   \
                 'Mendocino County', 'Mendocino', 'Lake County', \
                 'High Valley', 'Clear Lake', 'Central Valley',  \
                 'Central Coast', 'California Other'),[0,0]],
    'topic':[2,(0,1,2,3,4,5,6,7,8),[0,0]],
    'T1':[0,(-1e+8,1e+8),[0,0]],
    'T2':[0,(-1e+8,1e+8),[0,0]],
    'T3':[0,(-1e+8,1e+8),[0,0]],
    'T4':[0,(-1e+8,1e+8),[0,0]],
    'T5':[0,(-1e+8,1e+8),[0,0]],
    'T6':[0,(-1e+8,1e+8),[0,0]],
    'T7':[0,(-1e+8,1e+8),[0,0]],
    'T8':[0,(-1e+8,1e+8),[0,0]],
    'T9':[0,(-1e+8,1e+8),[0,0]]}

print("****** Examining Topic Assignment Versus Points & Price *****")
n_regions    = 18
avg_points   = [0] * n_topics
avg_price    = [0] * n_topics
t_counts     = [0] * n_topics
# region is a dictionary of lists by region
# Each list has 4 values: sum_points, npoints, sum_price, nprice
region = {}
for r in attribute_map['Region'][1]:
    region[r] = [0, 0, 0, 0]
for i in range(n_reviews):
    j = int(df['topic'].iloc[i])
    t_counts[j] += 1
    avg_points[j] += df['points'].iloc[i]
    region[df['Region'].iloc[i]][0] += df['points'].iloc[i]
    region[df['Region'].iloc[i]][1] += 1
    if pd.isnull(df['price'].iloc[i])==True:
        continue
    avg_price   [j] += df['price' ].iloc[i]
    region[df['Region'].iloc[i]][2] += df['price'].iloc[i]
    region[df['Region'].iloc[i]][3] += 1
print('{:<6s}{:>7s}{:>8s}{:>8s}'.format("TOPIC", "N", "POINTS", "PRICE"))
for i in range(n_topics):
    if t_counts[i]>0:
        avg_points[i] = avg_points[i]/t_counts[i]
        avg_price [i] = avg_price [i]/t_counts[i]
    print('{:>3d}{:>10d}{:>8.2f}{:>8.2f}'.format((i+1), t_counts[i], \
          avg_points[i], avg_price[i]))
print("")
print('{:<24s}{:>5s}{:>9s}{:>8s}'.format("REGION", "N", "POINTS", "PRICE"))
for r in attribute_map['Region'][1]:
    region[r][0] = region[r][0]/region[r][1] # Avg points
    region[r][2] = region[r][2]/region[r][3] # Avg price
```

```
    print('{:<24s}{:>6d}{:>8.2f}{:>8.2f}'.format(r, region[r][1], region[r][0], \
          region[r][2]))
```

```
****** Examining Topic Assignment Versus Points & Price *****
TOPIC       N  POINTS   PRICE
  1     11276   89.14   57.61
  2       512   88.71   50.91
  3       612   84.20   29.37
  4       234   86.69   42.85
  5        94   86.93   40.66
  6       158   84.52   31.45
  7       135   85.13   36.32
  8        56   87.02   57.16
  9        58   89.62   58.34

REGION                       N  POINTS   PRICE
South Coast                 52   87.04   61.37
Sonoma                    2277   88.09   41.81
Sierra Foothills           126   87.20   28.77
Redwood Valley               3   87.67   23.00
Red Hills Lake County       37   88.78   35.30
North Coast                183   86.07   21.11
Napa-Sonoma                 84   90.08   60.30
Napa                      7348   89.97   72.23
Mendocino/Lake Counties    196   86.22   27.63
Mendocino Ridge              3   86.00   40.00
Mendocino County            29   87.62   22.97
Mendocino                   30   87.27   24.80
Lake County                 34   87.74   30.50
High Valley                  3   88.67   30.00
Clear Lake                   1   84.00   32.00
Central Valley             203   85.34   18.21
Central Coast             1779   87.19   34.66
California Other           747   84.78   13.62
```

Drop Data with Missing Target

Drop the few cases with missing values for the wine price. This is necessary since linear regression cannot accept values with missing values, and it is not good practice to impute missing values for the target attribute.

In [23]:

```
target   = 'price'
# Drop data with missing values for target (price)
drops= []
for i in range(df.shape[0]):
    if pd.isnull(df['price'][i]):
        drops.append(i)
df = df.drop(drops)
df = df.reset_index()
```

Linear Regression

In [25]:

```
encoding = 'one-hot'
scale    = None  # Interval scaling:  Use 'std', 'robust' or None
# drop=False - do not drop last category - used for Decision Trees
rie = ReplaceImputeEncode(data_map=attribute_map, nominal_encoding=encoding, \
                          interval_scale = scale, drop=True, display=True)
encoded_df = rie.fit_transform(df)
varlist = [target, 'T1', 'T2', 'T3', 'T4', 'T5', 'T6', 'T7', 'T8', 'T9', \
           'points']
X = encoded_df.drop(varlist, axis=1)
y = encoded_df[target]
np_y = np.ravel(y) #convert dataframe column to flat array
col  = rie.col
for i in range(len(varlist)):
    col.remove(varlist[i])

lr = LinearRegression()
lr = lr.fit(X,np_y)

linreg.display_coef(lr, X, y, col)
linreg.display_metrics(lr, X, y)
```

```
********** Data Preprocessing ***********
Features Dictionary Contains:
11 Interval,
0 Binary, and
2 Nominal Attribute(s).

Data contains 13085 observations & 18 columns.

year:
        7436 missing: Drop this attribute.

Attribute Counts
.................. Missing  Outliers
Review.......         0         0
description..         0         0
year.........      7436         0
points.......         0         0
price........         0         0
winery.......         0         0
Region.......         0         0
topic........         0         0
T1...........         0         0
T2...........         0         0
T3...........         0         0
T4...........         0         0
T5...........         0         0
T6...........         0         0
```

```
T7..........        0         0
T8..........        0         0
T9..........        0         0

Coefficients
Intercept..        65.8740
Region0....       -47.4664
Region1....       -27.8104
Region2....       -45.2276
Region3....       -32.3092
Region4....       -34.3092
Region5....       -33.2484
Region6....       -38.6528
Region7....       -41.0102
Region8....       -19.8898
Region9....       -34.7456
Region10...         8.4922
Region11...        -3.9322
Region12...       -41.9438
Region13...       -28.4671
Region14...       -41.3092
Region15...       -35.2602
Region16...       -21.2841
topic0.....        -1.5648
topic1.....        -1.0193
topic2.....       -17.3042
topic3.....        -7.6420
topic4.....       -11.2345
topic5.....       -13.8625
topic6.....        -5.9842
topic7.....         6.3289

Model Metrics
Observations..........          13085
Coefficients..........             26
DF Error..............          13059
R-Squared.............         0.2462
Mean Absolute Error....       23.4908
Median Absolute Error..       16.9749
Avg Squared Error......     1304.2944
Square Root ASE........       36.1150
```