Stat656 Solution to Week 10 Assignment

April 11, 2018

1 Week 10 Assignment

This is a solution to the week 10. The main purpose of this assignment is to investigate topic analysis using Latient Dierchlet Analysis. This is an example conducts text classification using Python together with the NLTK and sci-learn packages.

1.0.1 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.

1.0.2 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

        from Class_tree import DecisionTree
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.tree import export_graphviz
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import train_test_split
        from pydotplus.graphviz import graph_from_dot_data
        import graphviz

        import pandas as pd
        import numpy  as np
        import string
        from time import time
        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.decomposition import LatentDirichletAllocation
```

1.0.3 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
```

1.0.4 The User Functions

my_analyzer(s) - Called by the sklearn Count & TFIDF Vectorizer's 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 = {'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
```

display_topics(lda, terms, n_terms=15) - Called to Display Topics

```
In [4]: def display_topics(lda, terms, n_terms=15):
            for topic_idx, topic in enumerate(lda):
                message  = "Topic #%d: " %(topic_idx+1)
                print(message)
                topic_terms_sorted = \
                        [terms[i] for i in 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):
                        message += '{:<15s}'.format(topic_terms_sorted[i+l])
                    print(message)
                if m> 0:
                    l = k*n
                    message = ''
                    for i in range(m):
                        message += '{:<15s}'.format(topic_terms_sorted[i+l])
                    print(message)
                print("")
            return
```

1.0.5 Attribute Map for Preprocessing Data

The following attribute map describes the data features. The attribute 'price' is the target in this problem. The attribute 'points' is the number of points assigned to the review with is highly correlated with the target. Fitting a model with points among the variables will dampen their importance in the model.

Attributes with a 2 as the first number are nominal. There are two nominal attributes in the data, Region, topic and attributes T1-T9. The topic attribute is the text topic cluster number. The attributes T1-T9 are the scores for individual documents for the topic cluster. In modeling, usually either the cluster topic assigment in the attribute topic or the topic scores in T1-T9 will enter the model.

Attributes with a 3 as their first number are attributes that will be ignore and will not be encoded or returned in the encoded dataframe. In this case, the attributes Review, winery and year are ignored becuase they either have too many classes or have more than 50% missing.

```
In [5]: 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,(0,5000),[0,0]],
      'T2':[0,(0,5000),[0,0]],
      'T3':[0,(0,5000),[0,0]],
      'T4':[0,(0,5000),[0,0]],
      'T5':[0,(0,5000),[0,0]],
      'T6':[0,(0,5000),[0,0]],
      'T7':[0,(0,5000),[0,0]],
      'T8':[0,(0,5000),[0,0]],
      'T9':[0,(0,5000),[0,0]]}
```

1.0.6 Read the Data File

The following code reads the Excel data file using Pandas. The maximum column width in Pandas needs to be increased to ensure the text are read without truncation.

```
In [6]: # Increase column width to let pandy read large text columns
        pd.set_option('max_colwidth', 32575)
        # Read N=13,575 California Cabernet Savignon Reviews
        file_path = '/Users/Home/Desktop/python/Excel/'
        df = pd.read_excel(file_path+"CaliforniaCabernet.xlsx")
```

1.0.7 Create Program Control Attributes

The files list is a list of the documents that will be processed. The remaing attributes are used to turn on and off tagging, stop words and stemming.

```
In [7]: # Setup program constants
        n_reviews   = len(df['description']) # Number of wine reviews
        m_features = 100                     # Number of SVD Vectors
        s_words    = 'english'               # Stop Word Dictionary
        ngram = (1,2)                        # n-gram POS modeling
        reviews = df['description']          # place all text reviews in reviews
        n_topics        =  9                 # number of topic clusters to extract
        max_iter        = 10                 # maximum number of itertions for LDA
        learning_offset = 10.                # learning offset for LDA
        learning_method = 'online'           # learning method for LDA
        tfidf = True                         # Set to True for TF-IDF Weighting
```

1.0.8 Tokenization, POS Tagging, Stop Removal & Stemming

There are two methods for text analysis: TF-IDF and Counts. The first is the term-frequency/inverse document frequency weighting. Raw term counts are weighted by the number of documents in which they appear. This down weights common terms used in all documents and up weights terms found in document clusters.

The second approach Counts is simply raw term frequencies. The term frequency matrix, tf contains the number of times a term appears in each review. The rows of this matrix are the reviews, N=13,515, and the columns are the terms. Most of the entries in this matrix are zero. That is not every term appears in every review. This matrix can be transformed using log(f+1) or binary.

The term descriptions are stored in a python list terms which has the same number of columns as tf, the maximum number of extracted terms. The term order is alphabetical rather than term frequency.

In [8]:
```
# Create Word Frequency by Review Matrix using Custom Analyzer
cv = CountVectorizer(max_df=0.95, min_df=2, max_features=m_features,\
                     analyzer=my_analyzer, ngram_range=ngram)
tf    = cv.fit_transform(reviews)
terms = cv.get_feature_names()
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]))
print("")
# if tfidf is requested, replace tf matrix with frequencies weighted by IDF
if tfidf == True:
    # Construct the TF/IDF matrix from the data
    print("Conducting Term/Frequency Matrix using TF-IDF")
    tfidf_vect = TfidfVectorizer(max_df=0.95, min_df=2,  \
                                 max_features=m_features,\
                                 analyzer=my_analyzer, ngram_range=ngram)
    tf     = tfidf_vect.fit_transform(reviews)
    terms  = tfidf_vect.get_feature_names()

    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):
        print('{:<15s}{:>8.2f}'.format(term_idf_scores[i][0], \
```

```
                    term_idf_scores[i][1]))
```

```
Terms with Highest Frequency:
flavor          8129
wine            7439
blackberry      7032
tannin          5135
cherry          5123
cabernet        4968
oak             4670
black           4596
currant         4404
dry             4143

Conducting Term/Frequency Matrix using TF-IDF
The Term/Frequency matrix has 13135  rows, and 100  columns.
The Term list has 100  terms.

Terms with Highest TF-IDF Scores:
flavor          1270.77
wine            1232.96
blackberry      1159.30
cherry          1025.16
tannin           988.14
cabernet         979.50
oak              941.74
dry              920.83
black            914.71
currant          886.45
```

1.0.9 Conduct the Latent Dirichlet Analysis to Create the Topic Matrix

The following code creates the topic matrix components_ describing the requested topic clusters identified using LDA.

```
In [9]: # In sklearn, LDA is synonymous with SVD (according to their doc)
        lda = LatentDirichletAllocation(n_components=n_topics, max_iter=max_iter,\
                                        learning_method=learning_method, \
                                        learning_offset=learning_offset, \
                                        random_state=12345)

        lda.fit_transform(tf)

        print('{:.<22s}{:>6d}'.format("Number of Reviews", len(reviews)))
        print('{:.<22s}{:>6d}'.format("Number of Terms",   len(terms)))
        print("\nTopics Identified using LDA with TF_IDF")
        display_topics(lda.components_, terms, n_terms=15)
```

```
Number of Reviews... 13135
Number of Terms...   100

Topics Identified using LDA with TF_IDF
Topic #1:
elegant        wine           balance        currant        flavor
could          dry            cab            structure      vintage
blackberry     show           tannin         quite          age

Topic #2:
palate         nose           black          body           plum
full           wine           offer          fruit          dark
note           finish         red            pepper         cherry

Topic #3:
delicious      best           soft           chocolate      blackberry
spice          flavor         yet            style          cabernet
rich           ripe           tannin         wine           oak

Topic #4:
oak            flavor         bit            sweet          cherry
berry          soft           smooth         lot            blackberry
drink          fruit          finish         tannin         wine

Topic #5:
year           hard           wine           blackberry     cellar
currant        tannin         young          cabernet       develop
long           flavor         black          age            oak

Topic #6:
price          good           dry            flavor         currant
great          body           blackberry     cabernet       full
tannic         black          cedar          drink          wine

Topic #7:
like           taste          fruity         mouth          aroma
flavor         wine           cherry         sweet          texture
blackberry     fruit          tannic         finish         ripe

Topic #8:
juicy          big            tobacco        sauvignon      wine
blend          cabernet       vineyard       finish         pepper
red            black          tannin         fruit          well

Topic #9:
jam            little         mocha          flavor         blackberry
valley         cherry         sweet          drink          cabernet
soft           one            acidity        ripe           oak
```

1.0.10 Score Individual Reviews

The results from the LDA analysis can be used to score each review. That is, calculate a score for each reviews for each topic cluster. The highest score is used to assigned each review to a a cluster.

First the LDA coefficients in components_ need to be normalized. That is the numbers need to be expressed as a probability. This is done by dividing each LDA coefficient by the sum of the coefficients for each topic.

Next this is used to calculate scores for each review and each topic.

```
In [10]: # Review Scores
         # Normalize LDA Weights to probabilities
         lda_norm = lda.components_ / lda.components_.sum(axis=1)[:, np.newaxis]

         # ***** SCORE REVIEWS *****
         rev_scores   = [[0]*(n_topics+1)] * n_reviews
         # Last topic count is number of reviews without any topic words
         topic_counts = [0] * (n_topics+1)

         for r in range(n_reviews):
             idx       = n_topics
             max_score = 0
             # Calculate Review Score
             j0 = tf[r].nonzero()
             nwords = len(j0[1])
             rev_score = [0]*(n_topics+1)
             # get scores for rth doc, ith topic
             for i in range(n_topics):
                 score  = 0
                 for j in range(nwords):
                     j1 = j0[1][j]
                     if tf[r,j1] != 0:
                             score += lda_norm[i][j1] * tf[r,j1]
                 rev_score [i+1]  = score
                 if score>max_score:
                     max_score = score
                     idx = i
             # Save review's highest scores
             rev_score[0]    = idx
             rev_scores [r] = rev_score
             topic_counts[idx]   += 1

         print('{:<6s}{:>8s}{:>8s}'.format("TOPIC", "REVIEWS", "PERCENT"))
         for i in range(n_topics):
             print('{:>3d}{:>10d}{:>8.1%}'.format((i+1), topic_counts[i], \
                   topic_counts[i]/n_reviews))
```

```
TOPIC  REVIEWS PERCENT
  1      801    6.1%
  2     1517   11.5%
  3      851    6.5%
  4     2168   16.5%
  5     1170    8.9%
  6     2565   19.5%
  7     1630   12.4%
  8     1242    9.5%
  9     1191    9.1%
```

1.0.11 Display Points/Price by Topics and Region

```
In [11]: print("****** Examining Topic Clusters Versus Points & Price *****")
         # 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)

         n_regions  = 18
         avg_points = [0] * n_topics
         avg_price  = [0] * n_topics
         n_price    = [0] * n_topics
         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])
             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]
             n_price   [j] += 1
             region[df['Region'].iloc[i]][2] += df['price'].iloc[i]
             region[df['Region'].iloc[i]][3] += 1
         print('{:<6s}{:>7s}{:>8s}'.format("TOPIC", "POINTS", "PRICE"))
         for i in range(n_topics):
             avg_points[i] = avg_points[i]/topic_counts[i]
             avg_price [i] = avg_price [i]/n_price[i]
             print('{:>3d}{:>10.2f}{:>8.2f}'.format((i+1), 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]
    region[r][2] = region[r][2]/region[r][3]
    print('{:<24s}{:>6d}{:>8.2f}{:>8.2f}'.format(r, region[r][1],\
                                    region[r][0], region[r][2]))
```

```
****** Examining Topic Clusters Versus Points & Price *****
TOPIC  POINTS   PRICE
  1     90.34   62.03
  2     89.59   59.67
  3     90.99   70.82
  4     87.67   51.33
  5     91.16   74.37
  6     88.00   40.50
  7     86.94   47.49
  8     89.74   70.69
  9     87.45   49.53

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
```