Stat 656 Week 12 Assignment

Name-Mayank Jaggi

Data

The data for this example consists of N=2,734 driver complaint to the National Transportation and Highway Safety Administration, NTHSA, regarding their automobile. It contains not only the complaints but also other information about the reason for their complaint and the type and condition of their automobile. The NTHSA uses these complaints to identify potential needs for automobile recalls.

The primary objective is to build a model that predicts whether the car was involved in a crash using the complaint and automobile characteristics.

Import Packages

In [8]:

```
#  classes provided for the course
from AdvancedAnalytics import ReplaceImputeEncode
from AdvancedAnalytics import logreg
from sklearn.linear_model import LogisticRegression

from AdvancedAnalytics import DecisionTree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from pydotplus.graphviz import graph_from_dot_data
import graphviz

import pandas as pd
import numpy  as np
import string
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 TfidfTransformer
from sklearn.decomposition import TruncatedSVD
```

The User Functions

In [9]:

```
def my_analyzer(s):
    # Synonym List
    syns = {'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
```

Attribute Map for Preprocessing Data

In [10]:

```
attribute_map = {
    'nthsa_id':['O',(0, 1e+12),[0,0]],
    'Year':['N',(2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011),[0,0
]],
    'make':['N',('CHEVROLET', 'PONTIAC', 'SATURN'),[0,0]],
    'model':['N',('COBALT', 'G5', 'HHR', 'ION', 'SKY', 'SOLSTICE'),[0,0]],
    'description':['O',(''),[0,0]],
    'crashed':['B',('N', 'Y'),[0,0]],
    'abs':['B',('N', 'Y'),[0,0]],
    'mileage':['I',(1, 200000),[0,0]],
    'topic':['N',(0,1,2,3,4,5,6,7,8),[0,0]],
    'T1':['I',(-1e+8,1e+8),[0,0]],
    'T2':['I',(-1e+8,1e+8),[0,0]],
    'T3':['I',(-1e+8,1e+8),[0,0]],
    'T4':['I',(-1e+8,1e+8),[0,0]],
    'T5':['I',(-1e+8,1e+8),[0,0]],
    'T6':['I',(-1e+8,1e+8),[0,0]],
    'T7':['I',(-1e+8,1e+8),[0,0]],
    'T8':['I',(-1e+8,1e+8),[0,0]],
    'T9':['I',(-1e+8,1e+8),[0,0]]}
```

Read the Data File

In [11]:

```
# Increase column width to let pandy read large text columns
pd.set_option('max_colwidth', 32000)
# Read NHTSA Comments
file_path = 'D:\\Work\\Course Work\\Semester 4\\STAT 656\\Lectures & Assign
ment\\Week 12\\'
df = pd.read_excel(file_path+"GMC_Complaints.xlsx")
```

Create Program Control Attributes

In [12]:

```
# Setup program constants
n_comments  = len(df['description']) # Number of wine reviews
m_features = None                    # Number of SVD Vectors
s_words    = 'english'               # Stop Word Dictionary
comments = df['description']         # place all text reviews in reviews
n_topics =  9                        # number of topic clusters to extract
max_iter = 10                        # maximum number of itertions
max_df   = 0.5                       # learning offset for LDAmax proportio
n of docs/reviews allowed for a term
```

Tokenization, POS Tagging, Stop Removal & Stemming

In [13]:

```
# 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=(1,2))
tf    = cv.fit_transform(comments)
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("")
```

```
Terms with Highest Frequency:
vehicle         6996
steer           2924
contact         2604
power           2131
failure         1745
drive           1670
problem         1466
turn            1256
recall          1239
go              1207
```

Create TFIDF Matrix

In [14]:

```
# Modify tf, term frequencies, to TF/IDF matrix from the data
print("Conducting Term/Frequency Matrix using TF-IDF")
tfidf_vect = TfidfTransformer(norm=None, use_idf=True) #set norm=None
tf         = tfidf_vect.fit_transform(tf)

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]))
```

```
Conducting Term/Frequency Matrix using TF-IDF
The Term/Frequency matrix has 2734  rows, and 3277  columns.
The Term list has 3277  terms.

Terms with Highest TF-IDF Scores:
vehicle         8162.39
contact         5371.28
steer           5114.56
power           4081.67
failure         3553.85
problem         3208.86
drive           2958.99
recall          2788.68
turn            2765.90
go              2757.42
```

Singular Value Decomposition

In [15]:

```
# 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)
```

```
********** GENERATED TOPICS **********
Topic #1:
+vehicle       +steer         +contact       +power         +p
roblem
+would         +go            +recall        +failure       +d
rive
+turn          +time          +start         +gm            +d
ealer

Topic #2:
-contact       -failure       -mileage       -state         -o
wn
+problem       -manufacturer  +go            -fuel          -r
epair
-current       +start         -chevrolet     -mph           +f
ix

Topic #3:
-steer         -power         +fuel          +ignition      +k
ey
+start         +pump          -drive         +switch        +s
aturn
-go            +leak          -turn          -wheel         +s
mell

Topic #4:
-power         -steer         +brake         +front         +t
ire
+air           +side          -fuel          +bag           +d
oor
+vehicle       +deploy        -recall        +driver        +h
it

Topic #5:
+fuel          -ignition      -key           +pump          -s
tart
-switch        +leak          +recall        +smell         +g
asoline
-saturn        -would         +tank          -contact       +g
as

Topic #6:
+door          +open          +handle        -brake         -s
tart
-vehicle       +side          +issue         +insid         +k
ey
+driver        -saturn        +plastic       +safety        +p
assenger

Topic #7:
```

```
-gm             -bag            -air            +fuel           +s
tart
+tire           -recall         +brake          +vehicle        -p
art
-deploy         +door           -problem        -crash          +t
urn

Topic #8:
+key            -start          -saturn         +ignition       -d
oor
+remove         -problem        +gear           -ion            +r
elease
+position       -cold           +gm             +turn           +s
hift

Topic #9:
+tire           -fuel           +firestone      +tread          -d
oor
-vehicle        +rim            +new            +tell           -b
ag
-deploy         -passenger      +saturn         -seat           +p
art
```

Add Topic Scores to Dataframe

In [16]:

```
# Store topic group for each doc in topics[]
topics       = [0] * n_comments
topic_counts = [0] * (n_topics+1)
for i in range(n_comments):
    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
    topic_counts[topics[i]] += 1

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

# Create comment_scores[] and assign the topic groups
comment_scores = []
for i in range(n_comments):
    u = [0] * (n_topics+1)
    u[0] = topics[i]
    for j in range(n_topics):
        u[j+1] = U[i][j]
    comment_scores.append(u)

# Augment Dataframe with topic group information
cols = ["topic"]
for i in range(n_topics):
    s = "T"+str(i+1)
    cols.append(s)
df_topics = pd.DataFrame.from_records(comment_scores, columns=cols)
df        = df.join(df_topics)
```

```
TOPIC COMMENTS PERCENT
  1      1765   64.6%
  2       433   15.8%
  3        99    3.6%
  4       157    5.7%
  5       117    4.3%
  6        51    1.9%
  7        37    1.4%
  8        49    1.8%
  9        26    1.0%
```

Logistic Regression

In [17]:

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

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']
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 = LogisticRegression(C=1e+16, tol=1e-16)
lr = lr.fit(X,np_y)

logreg.display_coef(lr, X.shape[1], 2, col)
logreg.display_binary_metrics(lr, X, y)
```

```
********** Data Preprocessing ***********
Features Dictionary Contains:
10 Interval,
2 Binary,
4 Nominal, and
3 Excluded Attribute(s).

Data contains 2734 observations & 19 columns.


Attribute Counts
.................. Missing  Outliers
nthsa_id.....         0         0
Year.........         0         0
make.........         0         0
model........         0         0
description..         0         0
crashed......         0         0
abs..........        18         0
mileage......       419        15
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..        -1.3440
mileage....        -0.0000
abs........         0.4742
Year0......        -0.3134
Year1......        -0.3024
Year2......        -0.6347
Year3......        -0.4346
Year4......        -0.3068
Year5......        -0.0261
Year6......        -0.0039
Year7......        -0.3864
make0......         0.0538
make1......        -0.2328
model0.....         0.3680
model1.....         0.8432
model2.....        -0.3142
model3.....        -0.4488
model4.....        -0.7162
```

```
topic0.....         0.6577
topic1.....         0.9815
topic2.....        -1.5971
topic3.....         3.5882
topic4.....        -2.7471
topic5.....        -0.5237
topic6.....         1.5550
topic7.....        -1.8164

Model Metrics
Observations.............      2734
Coefficients.............        26
DF Error.................      2708
Mean Absolute Error........    0.3248
Avg Squared Error.........    0.1622
Accuracy..................    0.7714
Precision.................    0.8011
Recall (Sensitivity).......    0.1975
F1-Score..................    0.3169
MISC (Misclassification)...     22.9%
     class 0..............      1.8%
     class 1..............     80.2%


     Confusion
       Matrix     Class 0   Class 1
Class 0.....      1964        36
Class 1.....       589       145
```

Decision Tree

In [18]:

```
scale = None # Not needed for Decision Trees
rie = ReplaceImputeEncode(data_map=attribute_map, nominal_encoding=encoding
, \
                          interval_scale = scale, drop=False, display=True)
encoded_df = rie.fit_transform(df)
varlist = [target, 'T1', 'T2', 'T3', 'T4', 'T5', 'T6', 'T7', 'T8', 'T9']
X = encoded_df.drop(varlist, axis=1)
y = encoded_df[target] # These are not standardized
np_y = np.ravel(y) #convert dataframe column to flat array
col  = rie.col
for i in range(len(varlist)):
    col.remove(varlist[i])

dtc = DecisionTreeClassifier(max_depth=7, \
                min_samples_split=5, min_samples_leaf=5)
dtc = dtc.fit(X,np_y)

DecisionTree.display_importance(dtc, col, plot=False)
DecisionTree.display_binary_metrics(dtc, X, y)
```

```
********** Data Preprocessing ***********
Features Dictionary Contains:
10 Interval,
2 Binary,
4 Nominal, and
3 Excluded Attribute(s).

Data contains 2734 observations & 19 columns.


Attribute Counts
.................. Missing  Outliers
nthsa_id.....         0         0
Year.........         0         0
make.........         0         0
model........         0         0
description..         0         0
crashed......         0         0
abs..........        18         0
mileage......       419        15
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

FEATURE.... IMPORTANCE
topic3.....   0.4967
mileage....   0.1480
make2......   0.0899
model2.....   0.0408
topic4.....   0.0364
topic2.....   0.0328
topic6.....   0.0279
model5.....   0.0257
Year4......   0.0215
topic1.....   0.0192
Year2......   0.0179
topic0.....   0.0131
Year3......   0.0095
abs........   0.0090
make0......   0.0062
model0.....   0.0042
Year1......   0.0012
Year0......   0.0000
```

```
Year5......   0.0000
Year6......   0.0000
Year7......   0.0000
Year8......   0.0000
make1......   0.0000
model1.....   0.0000
model3.....   0.0000
model4.....   0.0000
topic5.....   0.0000
topic7.....   0.0000
topic8.....   0.0000


Model Metrics
Observations..............      2734
Features..................        29
Maximum Tree Depth.........         7
Minimum Leaf Size..........         5
Minimum split Size.........         5
Mean Absolute Error........    0.3107
Avg Squared Error..........    0.1553
Accuracy...................    0.7835
Precision..................    0.8142
Recall (Sensitivity).......    0.2507
F1-Score...................    0.3833
MISC (Misclassification)...     21.7%
     class 0...............      2.1%
     class 1...............     74.9%


     Confusion
       Matrix     Class 0   Class 1
Class 0.....      1958        42
Class 1.....       550       184
```