Configuration Manual

Viraj Pawar
x16112521
MSc Research Project in Data Analytics

September 16, 2017

Python Environment

The python code was implemented in jupyter notebook. To download jupyter notebook use the following command in the terminal- 'pip3 install jupyter' in terminal.

Python modules

asr-evaluation (2.0.0) - To calculate Word Error Rate (belambert; n.d.)
gensim (2.2.0) - For Word2vec features (Řehůřek and Sojka; 2010)
google-api-python-client (1.6.2) - Communicate with Google API
matplotlib (2.0.2) - For plotting graphs (Hunter; 2007)
nltk (3.2.3) - Used for Pre-processing text (Bird et al.; 2009)
numpy (1.12.1) - For numpy array (Walt et al.; 2011)
pandas (0.20.1) - For importing dataset into a dataframe (McKinney; 2010)
scikit-learn (0.18.1) - Used for implementing classifiers and bag-of-words, tf-idf feature vectors (Buitinck et al.; 2013)
seaborn (0.7.1) - Used for plotting graphs (Waskom et al.; 2014)
SpeechRecognition (3.6.5) - Used for transcription using SpeechRecognition API (Zhang; n.d.)

Data Collection

Download audio recording from the following website: http://bit.ly/2oQLBqR

In [ ]:
```
cd /home/hduser/audio_data/High
```

Functions to transcribe audio file

In [ ]:
```
import speech_recognition as sr
from os import path

# Function to transcribe audio using IBM speech-to-text API.
def IBM(x):
    audio_file = path.join(path.dirname(path.realpath(x)), x)
    r = sr.Recognizer()
    with sr.AudioFile(audio_file) as source:
        audio = r.record(source)
    IBM_USERNAME = "ba305638-8b4b-479c-b6f9-6c3db42261e6"
    IBM_PASSWORD = "qicSAYXGLKLy"
    text = r.recognize_ibm(audio,username=IBM_USERNAME,password=IBM_PASSWORD)
    file = open("High_good_ibm.txt", "a")
    y = str(x)
    file.write(y + "\n" + str(text) + "\n")
    file.close()
    print('File Saved')
    return
```

In [ ]:
```
'''Initialize google cloud with 'gcloud init' then login into the account '''

# Function to transcribe using Google speech-to-text API.
def google(x):
    audio_file = path.join(path.dirname(path.realpath(x)), x)
    r = sr.Recognizer()
    with sr.AudioFile(audio_file) as source:
        audio = r.record(source)
    GOOGLE_CLOUD_SPEECH_CREDENTIALS = None
    text =\
    r.recognize_google_cloud(audio,\
                             credentials_json=\
                             GOOGLE_CLOUD_SPEECH_CREDENTIALS)
    file = open("High_Good_google.txt", "a")
    y = str(x)
    file.write(y + "\n" + str(text) + "\n")
    file.close()
    print('File Saved')
    return
```

In [ ]:
```
# Function to transcribe using CMU ASR.
def cmu(x):
    audio_file = path.join(path.dirname(path.realpath(x)), x)
    r = sr.Recognizer()
    with sr.AudioFile(audio_file) as source:
        audio = r.record(source)
    text = r.recognize_sphinx(audio)
    file = open("High_Good_cmu.txt", "a")
    y = str(x)
```

```
            file.write(y + "\n" + str(text) + "\n")
            file.close()
            print('File Saved')
            return
```

```
In [ ]: # Call functions to transcribe audio
        IBM('1.flac')
        google('1.flac')
        cmu('1.flac')
        # File is saved in txt format in the working directory.
```

Create and a csv file format with attribute name as IncidentID, Text and Class for each ASR output.

Preprocess the dataset

Pre-processing step 1. Regular expression

2. lower case
3. stemming
4. Remove stopwords

Functions to pre-process text

```
In [1]: # Import nltk modules for pre-processing

        import nltk # Import Natural Language Toolkit module
        import re # Import regular expression library
        from nltk.corpus import stopwords # import stopwords

        # create a set of pre-defined stopwords from nltk for speed
        stopwords = set(nltk.corpus.stopwords.words('english'))

        # Update stopwords list
        stopwords.update(['911','like', 'name', 'okay',
                          'ok', 'coming', 'could', 'days', 'everyone',
                          'get', 'give', 'going', 'liked', 'say', 'th',
                          'still', 'vs','call','operator',
                          'phone','hello','nine', 'address','one','building'])


        # create a stemmer variable. Assign this variable snowballstemmer.
        stemmer = nltk.stem.SnowballStemmer('english')

        # Set regular expression pattern to pattern variable
        pattern = r"(?u)\b\w\w+\b"

        # Function for stemming
        def stem_tokens(tokens, stemmer):
            stemmed = []
            for token in tokens:
                stemmed.append(stemmer.stem(token))
            return stemmed

        # Wrapped function for tokenization, Regular_expression,
        # Stop words removal and stemming
        def preprocess_transcripts(transcript,
                                   token_pattern = pattern,
                                   exclude_stopword=True,
                                   stem=True):
```

```
        # stop words are not removed for word2vec average features vectors
        token_pattern = re.compile(token_pattern, flags = 0 )
        tokens = [x.lower() for x in token_pattern.findall(transcript)]
        tokens_stemmed = tokens
        if stem:
            tokens_stemmed = stem_tokens(tokens, stemmer)
        if exclude_stopword:
            tokens_stemmed = [x for x in tokens_stemmed if x not in stopwords]

        return tokens_stemmed
```

Class for feature extraction

```
In [2]: import numpy as np # import modules
        import random

        class features(object):

            # Bag of words feature vectors. Binary occurence marker of features are used.
            def bow(train, test, num_features):
                from sklearn.feature_extraction.text import CountVectorizer
                # import countvectorizer module
                vectorizer = CountVectorizer(analyzer = "word",
                                             tokenizer = None,
                                             binary = True,
                                             preprocessor = None,
                                             stop_words = None,
                                             max_features = num_features)
                # fit the training dataset
                train_bw = vectorizer.fit_transform(train.map(lambda x: ' '.join(x)))
                test_bw = vectorizer.transform(test.map(lambda x: ' '.join(x)))
                print('Vectorized')
                return train_bw, test_bw # return bag-of-words feature vectors

            # Tf-idf vectorization function
            def tfidf(train, test, num_features):
                from sklearn.feature_extraction.text import TfidfVectorizer
                # Create unigrams
                vectorizer = TfidfVectorizer(sublinear_tf=True,
                                         max_df=0.8,
                                         min_df=2,
                                         ngram_range=(1,2),
                                         norm = 'l1',
                                         max_features = num_features)
                train_tf = vectorizer.fit_transform(train.map(lambda x: ' '.join(x)))
                test_tf = vectorizer.transform(test.map(lambda x: ' '.join(x)))
                print('Vectorized')
                return train_tf, test_tf # return tfidf features


            '''Class to create word2vec average vectors'''
        class word2vec(object):
            # Function to average all of the word vectors in a given transcript
            def AvgFeatures(words, model, num_features):
                # initialize an empty numpy array
```

```
                feature = np.zeros((num_features,),dtype="float32")
                xwords = 0.

                #index2word is a list od words in the model's vocabulary. convert it into set.
                model_words_set = set(model.wv.index2word)

                # Loop over each word in the transcript
                # and if that word is in the model's vocabulory,
                # add its feature vector to the total in numpy array
                for word in words:
                    if word in model_words_set:
                        xwords = xwords + 1.
                        feature = np.add(feature, model[word])
                # Divide the features by the total number of words in the transcript.
                avgfeatures = np.divide(feature, xwords)
                return avgfeatures

            # load pre-trained model
            def load_model(trained_model):
                from gensim.models import Word2Vec
                model = \
                gensim.models.KeyedVectors.load_word2vec_format(trained_model,
                                                                binary=True)
                print('Model Loaded')
                return model
```

```
In [3]: # Load pre-trained model
        # Loading takes about 10 mins depending upon the memory avaiable.
        # Download pre-trained word2vec model using:
        # 'git clone https://github.com/mmihaltz/word2vec-GoogleNews-vectors'
        # in terminal
        import gensim
        model = word2vec.load_model('/home/hduser/model/GoogleNews-vectors-negative300.bin')
        num_features = 300 # take 300 features
```

```
Model Loaded
```

Change directory to the dataset directory
Import train and test dataset

```
In [31]: cd /home/hduser/dataset/
```

```
/home/hduser/dataset
```

```
In [32]: # Function for pandas dataframe
         def pandas_DataFrame(filename):
             import pandas as pd
             return pd.DataFrame.from_csv(filename, sep=',' , encoding='utf-8')

         # import train dataset
         ibm = pandas_DataFrame('ibm.csv')
         cmu = pandas_DataFrame('cmu.csv')
         google = pandas_DataFrame('google.csv')
```

```
google_wv = pandas_DataFrame('google.csv')
# import test dataset
test = pandas_DataFrame('test.csv')
test_wv = pandas_DataFrame('test.csv')
# print first five rows
test.head()
```

Out[32]:

IncidentID	Text	Class
0	i have been shot	High
0	My parents were shot	High
0	I have a gun range officer that got shot in t...	High
0	There is a fire	High
0	somebody just threw acid on my mom. and i don'...	High

```
In [33]: # Randomize dataset
         # IBM ASR generated dataset
         ibm = ibm.iloc[np.random.permutation(len(ibm))]
         # CMU ASR generated dataset
         cmu = cmu.iloc[np.random.permutation(len(cmu))]
         # Google ASR generated dataset
         google = google.iloc[np.random.permutation(len(google))]

         google_wv = google_wv.iloc[np.random.permutation(len(google_wv))]
         # Manually transcribed first utterances.
         test = test.iloc[np.random.permutation(len(test))]
         test_wv = test_wv.iloc[np.random.permutation(len(test_wv))]
         # print first five rows
         test.head()
```

Out[33]:

IncidentID	Text	Class
1	i would like to report like two double murder	Medium
0	House upstairs going fast the bedroom is on f...	High
0	yes my name is Andrew kunkasi i m CEO of the c...	High
0	Yep. Hurry up he's unconscious, he's And he's ...	High
0	I have a nine-week old infant and he just stop...	High

Create unigrams from train and test dataset

Get list of unigrams from train and test dataset using map function.

```
In [34]: # Unigrams of training dataset
         # IBM
         ibm['unigram'] = ibm['Text'].map(lambda x: preprocess_transcripts(x))

         # Google
         google['unigram'] = google['Text'].map(lambda x: preprocess_transcripts(x))

         # cmu
         cmu['unigram'] = cmu['Text'].map(lambda x: preprocess_transcripts(x))

         # unigrams of test dataset
         test['unigram'] = test['Text'].map(lambda x: preprocess_transcripts(x))

         # print unigrams of test dataset
         test.head(5)
```

```
Out[34]:                                                        Text   Class  \
         IncidentID
         1                  i would like to report like two double murder   Medium
         0           House upstairs going fast  the bedroom is on f...    High
         0           yes my name is Andrew kunkasi i m CEO of the c...    High
         0           Yep. Hurry up he's unconscious, he's And he's ...    High
         0           I have a nine-week old infant and he just stop...    High

                                                              unigram
         IncidentID
         1                          [would, report, two, doubl, murder]
         0           [hous, upstair, go, fast, bedroom, fire, hous,...
         0           [yes, andrew, kunkasi, ceo, chief, financi, cr...
         0           [yep, hurri, unconsci, bayshor, court, ocean, ...
         0                            [week, old, infant, stop, breath]
```

Create unigrams for word2vec word embeddings. Stop words are not removed from the text as well as the words are not stemmed for word2vec word embeddings.

```
In [35]: # Unigrams from google ASR generated dataset for word2vec word embeddings
         google_wv['unigram'] =  google_wv['Text'].map(lambda x: preprocess_transcripts(x,\
                                                                 exclude_stopword=False,\
                                                                 stem=False))
         test_wv['unigram'] = test_wv['Text'].map(lambda x: preprocess_transcripts(x,\
                                                               exclude_stopword=False,\
                                                               stem=False))
         test_wv['unigram'].head(5)
```

```
Out[35]: IncidentID
         0    [the, tfa, agent, looks, like, he, is, having,...
         1    [have, someone, here, that, need, to, be, arre...
         1     [have, been, just, bitten, by, red, tailed, fox]
         1    [have, kind, of, particular, emergency, here, ...
         1          [car, just, coming, down, the, wrong, lane]
         Name: unigram, dtype: object
```

Vectorize using Bag-of-words

Create sparse matrix of feature vectors for train and test dataset

```
In [36]: # Creating feature vectors for training and test dataset
         # using 1000 feature i.e. vocabulary list

         X_train_ibm, X_test = features.bow(ibm['unigram'] , test['unigram'], 1000)

         X_train_google, X_test = features.bow(google['unigram'], test['unigram'], 1000)

         X_train_cmu, X_test = features.bow(cmu['unigram'], test['unigram'], 1000)
```

```
Vectorized
Vectorized
Vectorized
```

Vectorize using tf-idf vectors

Create spare matrix of tf-idf feature vectors for Google dataset

```
In [37]: # Using 1000 features
         X_train_google_tf, X_test_tf = features.tfidf(google['unigram'],
                                                       test['unigram'],
                                                       1000)
```

```
Vectorized
```

Take average word2vec features

Create average feature vectors for Google transcript

```
In [38]: def makeFeatureVec(words, model, num_features):
             # Function to average all of the word vectors in a given
             # paragraph
             #
             # Pre-initialize an empty numpy array (for speed)
             featureVec = np.zeros((num_features,),dtype="float32")
             #
             nwords = 0.
             #
             # Index2word is a list that contains the names of the words in
             # the model's vocabulary. Convert it to a set, for speed
             index2word_set = set(model.wv.index2word)
             #
             # Loop over each word in the review and, if it is in the model's
             # vocaublary, add its feature vector to the total
             for word in words:
                 if word in index2word_set:
                     nwords = nwords + 1.
                     featureVec = np.add(featureVec,model[word])
             #
             # Divide the result by the number of words to get the average
             featureVec = np.divide(featureVec,nwords)
             return featureVec


         def getAvgFeatureVecs(reviews, model, num_features):

             counter = 0.

             reviewFeatureVecs = np.zeros((len(reviews),
                 num_features),dtype="float32")

             for review in reviews:
                 if counter%1. == 0.:
                     print ("transcripts %d of %d" % (counter, len(reviews)))

                # Call the function (defined above) that makes average feature vectors
                 reviewFeatureVecs[int(counter)] = makeFeatureVec(review,
                                                                  model,
                                                                  num_features)
                #
```

```
            # Increment the counter
             counter = counter + 1.
        return reviewFeatureVecs
```

In [39]:
```
# take average vectors for word2vec word embeddings
X_train_google_wv = getAvgFeatureVecs(google_wv['unigram'], model, num_features )
X_test_wv = getAvgFeatureVecs(test_wv['unigram'], model, num_features )

from sklearn.preprocessing import normalize
# Normalise
X_train_google_wv =normalize(X_train_google_wv)
X_test_wv = normalize(X_test_wv)# Create targets
y_train_ibm = ibm['Class'].values
y_train_google = google['Class'].values
y_train_cmu = cmu['Class'].values
y_test = test['Class'].values
y_train_google_wv = google_wv['Class'].values
y_test_wv = test_wv['Class'].values
```

```
transcripts 0 of 152
transcripts 1 of 152
transcripts 2 of 152
transcripts 3 of 152
transcripts 4 of 152
transcripts 5 of 152
transcripts 6 of 152
transcripts 7 of 152
transcripts 8 of 152
transcripts 9 of 152
transcripts 10 of 152
transcripts 11 of 152
transcripts 12 of 152
transcripts 13 of 152
transcripts 14 of 152
transcripts 15 of 152
transcripts 16 of 152
transcripts 17 of 152
transcripts 18 of 152
transcripts 19 of 152
transcripts 20 of 152
transcripts 21 of 152
transcripts 22 of 152
transcripts 23 of 152
transcripts 24 of 152
transcripts 25 of 152
transcripts 26 of 152
transcripts 27 of 152
transcripts 28 of 152
transcripts 29 of 152
transcripts 30 of 152
transcripts 31 of 152
transcripts 32 of 152
transcripts 33 of 152
transcripts 34 of 152
transcripts 35 of 152
```

```
transcripts 36 of 152
transcripts 37 of 152
transcripts 38 of 152
transcripts 39 of 152
transcripts 40 of 152
transcripts 41 of 152
transcripts 42 of 152
transcripts 43 of 152
transcripts 44 of 152
transcripts 45 of 152
transcripts 46 of 152
transcripts 47 of 152
transcripts 48 of 152
transcripts 49 of 152
transcripts 50 of 152
transcripts 51 of 152
transcripts 52 of 152
transcripts 53 of 152
transcripts 54 of 152
transcripts 55 of 152
transcripts 56 of 152
transcripts 57 of 152
transcripts 58 of 152
transcripts 59 of 152
transcripts 60 of 152
transcripts 61 of 152
transcripts 62 of 152
transcripts 63 of 152
transcripts 64 of 152
transcripts 65 of 152
transcripts 66 of 152
transcripts 67 of 152
transcripts 68 of 152
transcripts 69 of 152
transcripts 70 of 152
transcripts 71 of 152
transcripts 72 of 152
transcripts 73 of 152
transcripts 74 of 152
transcripts 75 of 152
transcripts 76 of 152
transcripts 77 of 152
transcripts 78 of 152
transcripts 79 of 152
transcripts 80 of 152
transcripts 81 of 152
transcripts 82 of 152
transcripts 83 of 152
transcripts 84 of 152
transcripts 85 of 152
transcripts 86 of 152
transcripts 87 of 152
transcripts 88 of 152
transcripts 89 of 152
```

```
transcripts 90 of 152
transcripts 91 of 152
transcripts 92 of 152
transcripts 93 of 152
transcripts 94 of 152
transcripts 95 of 152
transcripts 96 of 152
transcripts 97 of 152
transcripts 98 of 152
transcripts 99 of 152
transcripts 100 of 152
transcripts 101 of 152
transcripts 102 of 152
transcripts 103 of 152
transcripts 104 of 152
transcripts 105 of 152
transcripts 106 of 152
transcripts 107 of 152
transcripts 108 of 152
transcripts 109 of 152
transcripts 110 of 152
transcripts 111 of 152
transcripts 112 of 152
transcripts 113 of 152
transcripts 114 of 152
transcripts 115 of 152
transcripts 116 of 152
transcripts 117 of 152
transcripts 118 of 152
transcripts 119 of 152
transcripts 120 of 152
transcripts 121 of 152
transcripts 122 of 152
transcripts 123 of 152
transcripts 124 of 152
transcripts 125 of 152
transcripts 126 of 152
transcripts 127 of 152
transcripts 128 of 152
transcripts 129 of 152
transcripts 130 of 152
transcripts 131 of 152
transcripts 132 of 152
transcripts 133 of 152
transcripts 134 of 152
transcripts 135 of 152
transcripts 136 of 152
transcripts 137 of 152
transcripts 138 of 152
transcripts 139 of 152
transcripts 140 of 152
transcripts 141 of 152
transcripts 142 of 152
transcripts 143 of 152
```

```
transcripts 144 of 152
transcripts 145 of 152
transcripts 146 of 152
transcripts 147 of 152
transcripts 148 of 152
transcripts 149 of 152
transcripts 150 of 152
transcripts 151 of 152
transcripts 0 of 80
transcripts 1 of 80
transcripts 2 of 80
transcripts 3 of 80
transcripts 4 of 80
transcripts 5 of 80
transcripts 6 of 80
transcripts 7 of 80
transcripts 8 of 80
transcripts 9 of 80
transcripts 10 of 80
transcripts 11 of 80
transcripts 12 of 80
transcripts 13 of 80
transcripts 14 of 80
transcripts 15 of 80
transcripts 16 of 80
transcripts 17 of 80
transcripts 18 of 80
transcripts 19 of 80
transcripts 20 of 80
transcripts 21 of 80
transcripts 22 of 80
transcripts 23 of 80
transcripts 24 of 80
transcripts 25 of 80
transcripts 26 of 80
transcripts 27 of 80
transcripts 28 of 80
transcripts 29 of 80
transcripts 30 of 80
transcripts 31 of 80
transcripts 32 of 80
transcripts 33 of 80
transcripts 34 of 80
transcripts 35 of 80
transcripts 36 of 80
transcripts 37 of 80
transcripts 38 of 80
transcripts 39 of 80
transcripts 40 of 80
transcripts 41 of 80
transcripts 42 of 80
transcripts 43 of 80
transcripts 44 of 80
transcripts 45 of 80
```

```
transcripts 46 of 80
transcripts 47 of 80
transcripts 48 of 80
transcripts 49 of 80
transcripts 50 of 80
transcripts 51 of 80
transcripts 52 of 80
transcripts 53 of 80
transcripts 54 of 80
transcripts 55 of 80
transcripts 56 of 80
transcripts 57 of 80
transcripts 58 of 80
transcripts 59 of 80
transcripts 60 of 80
transcripts 61 of 80
transcripts 62 of 80
transcripts 63 of 80
transcripts 64 of 80
transcripts 65 of 80
transcripts 66 of 80
transcripts 67 of 80
transcripts 68 of 80
transcripts 69 of 80
transcripts 70 of 80
transcripts 71 of 80
transcripts 72 of 80
transcripts 73 of 80
transcripts 74 of 80
transcripts 75 of 80
transcripts 76 of 80
transcripts 77 of 80
transcripts 78 of 80
transcripts 79 of 80
```

benchmark classifiers

In [40]:
```
# Import Sklearn classifier module
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

# Import sklearn metrics
from sklearn import metrics
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score

import sys
# Used for Graphs
import matplotlib.pyplot as plt
```

In [41]:
```
labels = ['High', 'Medium', 'Low']
```

```
def compare(clf):
    print('=' * 80)
    # training and testing of bag-of-words
    # feature vectors from IBM ASR generated transcripts
    print('Training on IBM dataset')
    clf.fit(X_train_ibm, y_train_ibm) # Fit the data for training
    pred_ibm = clf.predict(X_test)
    acc_ibm = metrics.accuracy_score(y_test, pred_ibm) # Get accuracy
    f_ibm = f1_score(y_test, pred_ibm, average='weighted') # Get f-score
    # Get Precision score
    pre_ibm = precision_score(y_test, pred_ibm, average='weighted')
    re_ibm = recall_score(y_test, pred_ibm, average='weighted')  # Get Recall score
    print('Accuracy on IBM:', acc_ibm)
    print('F-score on IBM:', f_ibm)
    print('Precision on IBM:', pre_ibm )
    print('Recall on IBM:', re_ibm )

    print('-' * 80)
    # training and testing of bag-of-words
    # feature vectors from Google ASR generated transcripts
    print('Training on Google dataset')
    clf.fit(X_train_google, y_train_google) # Fit the data for training
    pred_google = clf.predict(X_test)
    acc_google = metrics.accuracy_score(y_test, pred_google) # Get accuracy
    f_google = f1_score(y_test, pred_google, average='weighted')
    pre_google = precision_score(y_test, pred_google, average='weighted')
    re_google = recall_score(y_test, pred_google, average='weighted')
    print('Accuracy on Google:', acc_google)
    print('F-score on Google:', f_google)
    print('Precision on Google:', pre_google )
    print('Recall on Google:', re_google )

    print('-' * 80)
    # training and testing of bag-of-words
    # feature vectors from CMUSphinx ASR generated transcripts
    print('Training on CMU dataset')
    clf.fit(X_train_cmu, y_train_cmu) # Fit the data for training
    pred_cmu = clf.predict(X_test)
    acc_cmu = metrics.accuracy_score(y_test, pred_cmu) # Get accuracy
    f_cmu = f1_score(y_test, pred_cmu, average='weighted')
    pre_cmu = precision_score(y_test, pred_cmu, average='weighted')
    re_cmu = recall_score(y_test, pred_cmu, average='weighted')
    print('Accuracy on CMU:', acc_cmu)
    print('F-score on CMU:', f_cmu)
    print('Precision on CMU:', pre_cmu)
    print('Recall on CMU:', re_cmu )

    print('-' * 80)
    # training and testing of tf-idf
    # feature vectors from Google ASR generated transcripts
    print('Training on tfidf vectors')
    clf.fit(X_train_google_tf, y_train_google) # Fit the data for training
    pred_google_tf = clf.predict(X_test_tf)
    acc_google_tf = metrics.accuracy_score(y_test, pred_google_tf) # Get accuracy
```

```
    f_google_tf = f1_score(y_test, pred_google_tf, average='weighted')
    pre_google_tf = precision_score(y_test, pred_google_tf, average='weighted')
    re_google_tf = recall_score(y_test, pred_google_tf, average='weighted')
    print('Accuracy on Google:', acc_google_tf)
    print('F-score on Google:', f_google_tf)
    print('Precision on Google:', pre_google_tf )
    print('Recall on Google:', re_google_tf )


    print('-' * 80)
    # training and testing of word2vec
    # feature vectors from Google ASR generated transcripts
    print('Training on word2vec avg vectors')
    clf.fit(X_train_google_wv, y_train_google)
    pred_google_wv = clf.predict(X_test_wv)
    acc_google_wv = metrics.accuracy_score(y_test, pred_google_wv)
    f_google_wv = f1_score(y_test, pred_google_wv, average='weighted')
    pre_google_wv = precision_score(y_test, pred_google_wv, average='weighted')
    re_google_wv = recall_score(y_test, pred_google_wv, average='weighted')
    print('Accuracy on Google:', acc_google_wv)
    print('F-score on Google:', f_google_wv)
    print('Precision on Google:', pre_google_wv)
    print('Recall on Google:', re_google_wv)
    print('-' * 80)
    clf_descr = str(clf).split('(')[0]
    cls = clf_descr

    return clf_descr, acc_ibm, acc_google, acc_cmu, acc_google_tf, acc_google_wv
```

Train and test classifiers

In [42]:
```
# Train and test classifiers

# Results from SVM
results = []
print('=' * 80)
print("SVM")
results.append(compare(LinearSVC(penalty='l1', dual=False,
                                  multi_class='ovr',
                                  tol=1e-3)))
# Results from AdaBoost
print('=' * 80)
print('AdaBoost')
results.append(compare(AdaBoostClassifier(n_estimators=256,
                                           algorithm='SAMME.R')))

# Results from Bernoulli Naive Bayes
print('=' * 80)
print("BernoulliNB")
results.append(compare(BernoulliNB(alpha=1.0,
                                  binarize=0.0,
                                  class_prior=None,
                                  fit_prior=True)))
# Results from Logistic regression
print('=' * 80)
```

```
        print('Logistic Regression')
        results.append(compare(LogisticRegression(solver='liblinear',
                                                  penalty='l1',
                                                  class_weight='balanced',
                                                  multi_class = 'ovr')))
```

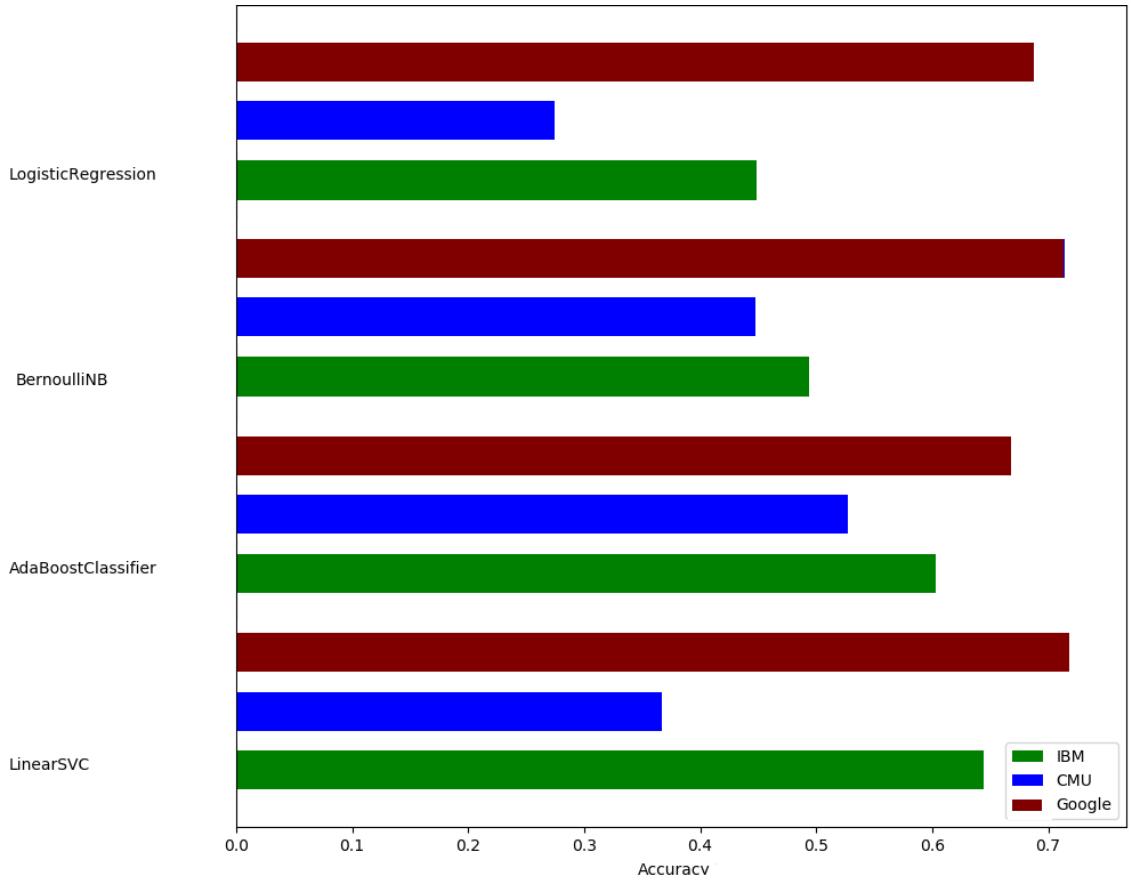
```
================================================================================
SVM
================================================================================
Training on IBM dataset
Accuracy on IBM: 0.653894736842
F-score on IBM: 0.635666465013
Precision on IBM: 0.635020933014
Recall on IBM: 0.657894736842
--------------------------------------------------------------------------------
Training on Google dataset
Accuracy on Google: 0.730789473684
F-score on Google: 0.741566283293
Precision on Google: 0.787996008279
Recall on Google: 0.735789473684
--------------------------------------------------------------------------------
Training on CMU dataset
Accuracy on CMU: 0.384615789474
F-score on CMU: 0.34831661298
Precision on CMU: 0.326553695143
Recall on CMU: 0.386315789474
--------------------------------------------------------------------------------
Training on tfidf vectors
Accuracy on Google: 0.538805263158
F-score on Google: 0.37465905998
Precision on Google: 0.295331125828
Recall on Google: 0.547105263158
--------------------------------------------------------------------------------
Training on word2vec avg vectors
Accuracy on Google: 0.458347368421
F-score on Google: 0.460689627294
Precision on Google: 0.35298825368
Recall on Google: 0.483947368421
--------------------------------------------------------------------------------
================================================================================
AdaBoost
================================================================================
Training on IBM dataset
Accuracy on IBM: 0.605347368421
F-score on IBM: 0.553041926851
Precision on IBM: 0.514462719298
Recall on IBM: 0.628947368421
--------------------------------------------------------------------------------
Training on Google dataset
Accuracy on Google: 0.682363157895
F-score on Google: 0.712696284217
Precision on Google: 0.752012034918
Recall on Google: 0.690263157895
--------------------------------------------------------------------------------
```

```
Training on CMU dataset
Accuracy on CMU: 0.538442105263
F-score on CMU: 0.37775913555
Precision on CMU: 0.297384868421
Recall on CMU: 0.546842105263
--------------------------------------------------------------------------------
Training on tfidf vectors
Accuracy on Google: 0.538426315789
F-score on Google: 0.555626682481
Precision on Google: 0.588433975564
Recall on Google: 0.545526315789
--------------------------------------------------------------------------------
Training on word2vec avg vectors
Accuracy on Google: 0.423068421053
F-score on Google: 0.415186241466
Precision on Google: 0.414298245614
Recall on Google: 0.427368421053
--------------------------------------------------------------------------------
================================================================================
BernoulliNB
================================================================================
Training on IBM dataset
Accuracy on IBM: 0.500147368421
F-score on IBM: 0.427568468683
Precision on IBM: 0.370439177414
Recall on IBM: 0.503947368421
--------------------------------------------------------------------------------
Training on Google dataset
Accuracy on Google: 0.713231578947
F-score on Google: 0.738743922039
Precision on Google: 0.735021907651
Recall on Google: 0.737631578947
--------------------------------------------------------------------------------
Training on CMU dataset
Accuracy on CMU: 0.472305263158
F-score on CMU: 0.39465905998
Precision on CMU: 0.395331125828
Recall on CMU: 0.427105263158
--------------------------------------------------------------------------------
Training on tfidf vectors
Accuracy on Google: 0.692368421053
F-score on Google: 0.672034914286
Precision on Google: 0.687050438596
Recall on Google: 0.692368421053
--------------------------------------------------------------------------------
Training on word2vec avg vectors
Accuracy on Google: 0.269231578947
F-score on Google: 0.25056030469
Precision on Google: 0.253616800622
Recall on Google: 0.277631578947
--------------------------------------------------------------------------------
================================================================================
Logistic Regression
================================================================================
```

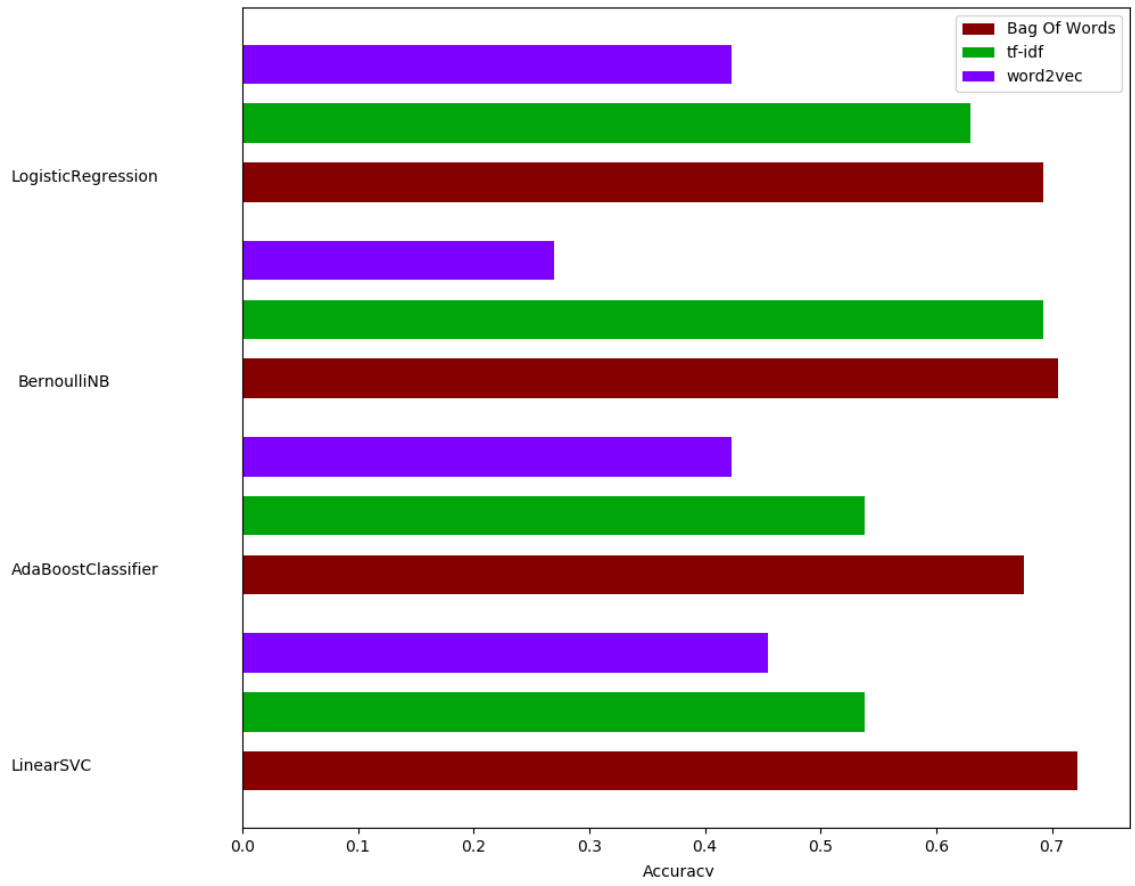
```
Training on IBM dataset
Accuracy on IBM: 0.484531578947
F-score on IBM: 0.462188805347
Precision on IBM: 0.465756578947
Recall on IBM: 0.467631578947
--------------------------------------------------------------------------------
Training on Google dataset
Accuracy on Google: 0.692389473684
F-score on Google: 0.633276413739
Precision on Google: 0.597283281734
Recall on Google: 0.692894736842
--------------------------------------------------------------------------------
Training on CMU dataset
Accuracy on CMU: 0.287673684211
F-score on CMU: 0.279845953642
Precision on CMU: 0.280505382775
Recall on CMU: 0.279473684211
--------------------------------------------------------------------------------
Training on tfidf vectors
Accuracy on Google: 0.667626315789
F-score on Google: 0.720422000948
Precision on Google: 0.732084487535
Recall on Google: 0.730526315789
--------------------------------------------------------------------------------
Training on word2vec avg vectors
Accuracy on Google: 0.423026315789
F-score on Google: 0.430422000948
Precision on Google: 0.462084487535
Recall on Google: 0.420526315789
--------------------------------------------------------------------------------
```

Print Graphs

In [43]:
```
# Print graph for comparison of accuracy of different IBM, Google,
# CMUSphinx ASR generated datasets.
indices = np.arange(len(results))
results = [[x[i] for x in results] for i in range(4)]
clf_names, ibm_acc, google_acc, cmu_acc = results
plt.figure(figsize=(12, 8))
plt.title("Score")
plt.barh(indices, cmu_acc, .2, label="CMU", color='blue')
plt.barh(indices + .3, google_acc, .2, label="Google",
         color='maroon')
plt.barh(indices + .6, ibm_acc, .2, label="IBM", color='green')
plt.yticks(())
plt.legend(loc='best')
plt.subplots_adjust(left=.25)
plt.subplots_adjust(top=.95)
plt.subplots_adjust(bottom=.05)
plt.xlabel('Accuracy')

for i, c in zip(indices, clf_names):
    plt.text(-.2, i, c)
plt.show()
```



In [44]:
```
# Print graph for comparison of accuracy on different feature vectors
# generated from Google ASR generated transcripts.

indices = np.arange(len(results))
results = [[x[i] for x in results] for i in range(4)]
clf_names, acc_google_tf,acc_google, acc_google_wv  = results
plt.figure(figsize=(12, 8))
plt.title("Score")
plt.barh(indices,cmu_acc , .2, label="Bag Of Words", color='maroon')
plt.barh(indices + .3, acc_google_tf, .2, label="word2vec",
         color='blue')
plt.barh(indices + .6, acc_google_wv, .2, label="tf-idf", color='green')
plt.yticks(())
plt.legend(loc='best')
plt.subplots_adjust(left=.25)
plt.subplots_adjust(top=.95)
plt.subplots_adjust(bottom=.05)
plt.xlabel('Accuracy')

for i, c in zip(indices, clf_names):
    plt.text(-.2, i, c)
plt.show()
```



In [ ]:

In [ ]:

References

belambert (n.d.). Python module for evaluating asr hypotheses (e.g. word error rate, word recognition rate) [software]. https://github.com/belambert/asr-evaluation.

Bird, S., Klein, E. and Loper, E. (2009). *Natural Language Processing with Python*, O'Reilly Media.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B. and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project, *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment, *Computing In Science & Engineering* **9**(3): 90–95.

McKinney, W. (2010). Data structures for statistical computing in python, *in* S. van der Walt and J. Millman (eds), *Proceedings of the 9th Python in Science Conference*, pp. 51 – 56.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora, *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, ELRA, Valletta, Malta, pp. 45–50. http://is.muni.cz/publication/884893/en.

Walt, S. v. d., Colbert, S. C. and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation, *Computing in Science & Engineering* **13**(2): 22–30.

Waskom, M., Botvinnik, O., Hobson, P., Cole, J. B., Halchenko, Y., Hoyer, S., Miles, A., Augspurger, T., Yarkoni, T., Megies, T., Coelho, L. P., Wehner, D., cynddl, Ziegler, E., diego0020, Zaytsev, Y. V., Hoppe, T., Seabold, S., Cloud, P., Koskinen, M., Meyer, K., Qalieh, A. and Allan, D. (2014). seaborn: v0.5.0 (november 2014).
URL: *https://doi.org/10.5281/zenodo.12710*

Zhang, A. (n.d.). Speech recognition (version 3.7) [software]. https://github.com/Uberi/speech_recognition#readme.