Transfer Learning

August 18, 2019

[14]:
```
from models import get_model
import argparse
import pickle
import string
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import roc_auc_score
import preprocessor as p
from collections import Counter
import os
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.contrib import learn
from tflearn.data_utils import to_categorical, pad_sequences
from scipy import stats
import tflearn
```

[15]:
```
models = [ 'cnn', 'lstm', 'blstm', 'blstm_attention']
word_vectors = ["random", "glove" ,"sswe"]
EMBED_SIZE = 50
EPOCHS = 5
BATCH_SIZE = 128
MAX_FEATURES = 2
NUM_CLASSES = 2
DROPOUT = 0.25
LEARN_RATE = 0.01
HASH_REMOVE=None
```

[16]:
```
def load_data(filename):
    print("Loading data from file: " + filename)
    data = pickle.load(open(filename, 'rb'))
    x_text = []
    labels = []
    for i in range(len(data)):
        if(HASH_REMOVE):
            x_text.append(p.tokenize((data[i]['text']).encode('utf-8')))
        else:
```

```
            x_text.append(data[i]['text'])
        labels.append(data[i]['label'])
    return x_text,labels

def get_filename(dataset):
    global HASH_REMOVE
    if(dataset=="twitter"):
        HASH_REMOVE = True
        EPOCHS = 10
        BATCH_SIZE = 128
        MAX_FEATURES = 2
        filename = "/home/sujeendra/Desktop/data/data/twitter_data.pkl"
    elif(dataset=="formspring"):
        HASH_REMOVE = False
        EPOCHS = 10
        BATCH_SIZE = 128
        MAX_FEATURES = 2
        filename = "/home/sujeendra/Desktop/data/data/formspring_data.pkl"
    elif(dataset=="wiki"):
        HASH_REMOVE = False
        EPOCHS = 5
        BATCH_SIZE = 512
        MAX_FEATURES = 5
        filename = "/home/sujeendra/Desktop/data/data/wiki_data.pkl"
    return filename
```

[17]:
```
def get_embedding_weights(filename, sep):
    embed_dict = {}
    file = open(filename,'r')
    for line in file.readlines():
        row = line.strip().split(sep)
        embed_dict[row[0]] = row[1:]
    print('Loaded from file: ' + str(filename))
    file.close()
    return embed_dict

def map_embedding_weights(embed, vocab, embed_size):
    vocab_size = len(vocab)
    embeddingWeights = np.zeros((vocab_size , embed_size))
    n = 0
    words_missed = []
    for k, v in vocab.iteritems():
        try:
            embeddingWeights[v] = embed[k]
        except:
            n += 1
            words_missed.append(k)
```

```
            pass
    print("%d embedding missed"%n, " of " , vocab_size)
    return embeddingWeights

def get_embeddings_dict(vector_type, emb_dim, data):
    if vector_type == 'sswe':
        emb_dim==50
        sep = '\t'
        vector_file = 'word_vectors/sswe-u.txt'
    elif vector_type =="glove":
        sep = ' '
        if data == "wiki":
            vector_file = 'word_vectors/glove.6B.' + str(emb_dim) + 'd.txt'
        else:
            vector_file = 'word_vectors/glove.twitter.27B.' + str(emb_dim) + 'd.
 ↪txt'
    else:
        print ("ERROR: Please specify a correst model or SSWE cannot be loaded␣
 ↪with embed size of: " + str(emb_dim))
        return None

    embed = get_embedding_weights(vector_file, sep)
    return embed
```

[18]:
```
def evaluate_model(model, testX, testY):
    temp = model.predict(testX)
    y_pred  = np.argmax(temp, 1)
    y_true = np.argmax(testY, 1)
    precision = metrics.precision_score(y_true, y_pred, average=None)
    recall = metrics.recall_score(y_true, y_pred, average=None)
    f1_score = metrics.f1_score(y_true, y_pred, average=None)
    print("Precision: " + str(precision) + "\n")
    print("Recall: " + str(recall) + "\n")
    print("f1_score: " + str(f1_score) + "\n")
    print(confusion_matrix(y_true, y_pred))
    print(":: Classification Report")
    print(classification_report(y_true, y_pred))
    return precision, recall, f1_score
```

[19]:
```
def print_scores(precision_scores, recall_scores, f1_scores):
    for i in range(NUM_CLASSES):
        print("\nPrecision Class %d (avg): %0.3f (+/- %0.3f)" % (i,␣
 ↪precision_scores[:, i].mean(), precision_scores[:, i].std() * 2))
        print( "\nRecall Class %d (avg): %0.3f (+/- %0.3f)" % (i, recall_scores[:
 ↪, i].mean(), recall_scores[:, i].std() * 2))
        print( "\nF1 score Class %d (avg): %0.3f (+/- %0.3f)" % (i, f1_scores[:,␣
 ↪i].mean(), f1_scores[:, i].std() * 2))
```

[20]:
```
def get_data(data, oversampling_rate):

    x_text, labels = load_data(get_filename(data))

    if(data=="twitter"):
        dict1 = {'racism':1,'sexism':1,'none':0} #Transfer learning only two␣
↪classes
        labels = [dict1[b] for b in labels]

        racism = [i for i in range(len(labels)) if labels[i]==2]
        sexism = [i for i in range(len(labels)) if labels[i]==1]
        x_text = x_text + [x_text[x] for x in racism]*(oversampling_rate-1)+␣
↪[x_text[x] for x in sexism]*(oversampling_rate-1)
        labels = labels + [2 for i in range(len(racism))]*(oversampling_rate-1)␣
↪+ [1 for i in range(len(sexism))]*(oversampling_rate-1)

    else:

        NUM_CLASSES = 2
        bully = [i for i in range(len(labels)) if labels[i]==1]
        x_text = x_text + [x_text[x] for x in bully]*(oversampling_rate-1)
        labels = list(labels) + [1 for i in␣
↪range(len(bully))]*(oversampling_rate-1)

    print("Counter after oversampling")
    from collections import Counter
    print(Counter(labels))

    filter_data = []
    for text in x_text:
        filter_data.append("".join(l for l in text if l not in string.
↪punctuation))

    return x_text, labels
```

[21]:
```
def train(data, x_text, labels, model_type, vector_type, embed_size,␣
↪max_document_length=None):
    X_train, X_test, Y_train, Y_test = train_test_split( x_text, labels,␣
↪random_state=121, test_size=0.10)

    if(max_document_length==None):
        post_length = np.array([len(x.split(" ")) for x in x_text])
        if(data != "twitter"):
            max_document_length = int(np.percentile(post_length, 95))
        else:
            max_document_length = max(post_length)
        print("Document length : " + str(max_document_length))
```

```
   vocab_processor = learn.preprocessing.
↪VocabularyProcessor(max_document_length, MAX_FEATURES)
   vocab_processor = vocab_processor.fit(x_text)

   trainX = np.array(list(vocab_processor.transform(X_train)))
   testX = np.array(list(vocab_processor.transform(X_test)))

   vocab_size = len(vocab_processor.vocabulary_)
   print("Vocabulary Size: {:d}".format(vocab_size))

   vocab = vocab_processor.vocabulary_._mapping

   trainY = np.asarray(Y_train)
   testY = np.asarray(Y_test)

   trainX = pad_sequences(trainX, maxlen=max_document_length, value=0.)
   testX = pad_sequences(testX, maxlen=max_document_length, value=0.)

   trainY = to_categorical(trainY, nb_classes=NUM_CLASSES)
   testY = to_categorical(testY, nb_classes=NUM_CLASSES)

   print("Running Model: " + model_type + " with word vector initiliazed with "␣
↪+ vector_type + " word vectors.")
   model = get_model(model_type, trainX.shape[1], vocab_size, embed_size,␣
↪NUM_CLASSES, LEARN_RATE)

   if(model_type == 'cnn'):
       if(vector_type!="random"):
           print("Word vectors used: " + vector_type)
           embeddingWeights = tflearn.
↪get_layer_variables_by_name('EmbeddingLayer')[0]
           model.set_weights(embeddingWeights,␣
↪map_embedding_weights(get_embeddings_dict(vector_type, embed_size, data),␣
↪vocab, embed_size))
           model.fit(trainX, trainY, n_epoch = EPOCHS, shuffle=True,␣
↪show_metric=True, batch_size=BATCH_SIZE)
       else:
           model.fit(trainX, trainY, n_epoch = EPOCHS, shuffle=True,␣
↪show_metric=True, batch_size=BATCH_SIZE)
   else:
       if(vector_type!="random"):
           print("Word vectors used: " + vector_type)
           model.layers[0].
↪set_weights([map_embedding_weights(get_embeddings_dict(vector_type,␣
↪embed_size, data), vocab, embed_size)])
```

```
            model.fit(trainX, trainY, epochs=EPOCHS, shuffle=True,␣
↪batch_size=BATCH_SIZE,
                      verbose=1)
        else:
            model.fit(trainX, trainY, epochs=EPOCHS, shuffle=True,␣
↪batch_size=BATCH_SIZE,
                      verbose=1)

    precision, recall, f1_score = evaluate_model(model, testX, testY)

    model_dict = {
        "model": model,
        "testX": testX,
        "testY": testY,
        "trainX" : trainX,
        "trainY" : trainY,
        "vocab": vocab_processor,
        "length": max_document_length,
        "data": data
    }

    return model_dict
```

[22]:
```
def transfer_learning_1(dict_1, dict_2, model_type=None, vector_type=None,␣
↪embed_size=None):

    model = dict_1["model"]
    length = dict_1["length"]
    vocab1 = dict_1["vocab"]
    vocab2 = dict_2["vocab"]
    testX = dict_2["testX"]
    testY = dict_2["testY"]

    temp = list(vocab2.reverse(testX))
    testX  = np.array(list(vocab1.transform(temp)))
    testX = pad_sequences(testX, maxlen=length, value=0.)

    evaluate_model(model, testX, testY)
```

[23]:
```
def map_embedding_trained_weights(embed, vocab_1, vocab_2):
    vocab_size = len(vocab_2)
    embeddingWeights = np.zeros((vocab_size , embed.shape[1]))
    n = 0
    words_missed = []
    for k, v in vocab_2.iteritems():
        try:
            embeddingWeights[v] = embed[vocab_1[k]]
        except:
```

```
            n += 1
            words_missed.append(k)
            pass
    print("%d embedding missed"%n, " of " , vocab_size)
    return embeddingWeights

def transfer_learning_2(dict_1, dict_2, model_type, vector_type, embed_size):
    trainX = dict_2["trainX"]
    trainY = dict_2["trainY"]
    testX = dict_2["testX"]
    testY = dict_2["testY"]

    EPOCHS = 5
    BATCH_SIZE = 128
    vocab_processor = dict_2["vocab"]
    vocab_size = len(vocab_processor.vocabulary_)
    print("Vocabulary Size: {:d}".format(vocab_size))

    vocab = vocab_processor.vocabulary_._mapping

    print("Running Model: " + model_type + " with word vector initiliazed with␣
 ↪word vectors trained on dataset 1.")
    model = get_model(model_type, trainX.shape[1], vocab_size, embed_size,␣
 ↪NUM_CLASSES, LEARN_RATE)

    if(model_type == 'cnn'):
        embeddingWeights = tflearn.
 ↪get_layer_variables_by_name('EmbeddingLayer')[0]
        embed = dict_1["model"].get_weights(embeddingWeights)
        model.set_weights(embeddingWeights, map_embedding_trained_weights(embed,␣
 ↪dict_1["vocab"].vocabulary_._mapping, vocab))
        model.fit(trainX, trainY, n_epoch = EPOCHS, shuffle=True,␣
 ↪show_metric=True, batch_size=BATCH_SIZE)
    else:
        embed = dict_1["model"].layers[0].get_weights()[0]
        model.layers[0].set_weights([map_embedding_trained_weights(embed,␣
 ↪dict_1["vocab"].vocabulary_._mapping, vocab)])
        model.fit(trainX, trainY, epochs=EPOCHS, shuffle=True,␣
 ↪batch_size=BATCH_SIZE,
              verbose=1)

    evaluate_model(model, testX, testY)
```

[24]:
```
def transfer_learning_3(dict_1, dict_2, model_type, vector_type, embed_size):

    trainX = dict_2["trainX"]
    trainY = dict_2["trainY"]
```

```
    testX = dict_2["testX"]
    testY = dict_2["testY"]

    vocab_processor = dict_2["vocab"]
    vocab_size = len(vocab_processor.vocabulary_)
    print("Vocabulary Size: {:d}".format(vocab_size))

    vocab = vocab_processor.vocabulary_._mapping

    print("Running Model: " + model_type + " with word vector initiliazed with␣
↪word vectors trainned on dataset 1.")
    model = get_model(model_type, trainX.shape[1], vocab_size, embed_size,␣
↪NUM_CLASSES, LEARN_RATE)

    EPOCHS = 5
    BATCH_SIZE = 128

    if(model_type == 'cnn'):

        embeddingWeights = tflearn.
↪get_layer_variables_by_name('EmbeddingLayer')[0]
        embed = dict_1["model"].get_weights(embeddingWeights)
        model.set_weights(embeddingWeights, map_embedding_trained_weights(embed,␣
↪dict_1["vocab"].vocabulary_._mapping, vocab))

        layer1Weights = tflearn.get_layer_variables_by_name('layer_1')[0]
        model.set_weights(layer1Weights, dict_1["model"].
↪get_weights(layer1Weights))

        layer2Weights = tflearn.get_layer_variables_by_name('layer_2')[0]
        model.set_weights(layer2Weights, dict_1["model"].
↪get_weights(layer2Weights))

        layer3Weights = tflearn.get_layer_variables_by_name('layer_1')[0]
        model.set_weights(layer3Weights, dict_1["model"].
↪get_weights(layer3Weights))

        fcWeights = tflearn.get_layer_variables_by_name('layer_1')[0]
        model.set_weights(fcWeights, dict_1["model"].get_weights(fcWeights))

        model.fit(trainX, trainY, n_epoch = EPOCHS, shuffle=True,␣
↪show_metric=True, batch_size=BATCH_SIZE)

    elif(model_type == 'blstm_attention'):
        embed = dict_1["model"].layers[0].get_weights()[0]
```

```
        model.layers[0].set_weights([map_embedding_trained_weights(embed,␣
↪dict_1["vocab"].vocabulary_._mapping, vocab)])
        model.layers[2].set_weights(dict_1["model"].layers[2].get_weights())
        model.layers[3].set_weights(dict_1["model"].layers[3].get_weights())
        model.layers[5].set_weights(dict_1["model"].layers[5].get_weights())
        model.fit(trainX, trainY, epochs=EPOCHS, shuffle=True,␣
↪batch_size=BATCH_SIZE,
              verbose=1)
    else:
        embed = dict_1["model"].layers[0].get_weights()[0]
        model.layers[0].set_weights([map_embedding_trained_weights(embed,␣
↪dict_1["vocab"].vocabulary_._mapping, vocab)])
        model.layers[2].set_weights(dict_1["model"].layers[2].get_weights())
        model.layers[4].set_weights(dict_1["model"].layers[4].get_weights())
        model.fit(trainX, trainY, epochs=EPOCHS, shuffle=True,␣
↪batch_size=BATCH_SIZE,
              verbose=1)

    evaluate_model(model, testX, testY)
```

[25]:
```
data_1 = "formspring"
data_2 = "twitter"
data_3 = "wiki"
model_type ="blstm"
vector_type = "random"
embed_size = 50
oversampling_rate = 3
```

[26]:
```
x_text, labels = get_data(data_1, oversampling_rate)
dict_1 =  train(data_1, x_text, labels, model_type, vector_type, embed_size)

'''x_text, labels = get_data(data_2, oversampling_rate)
dict_2 =  train(data_2, x_text, labels, model_type, vector_type, embed_size)

x_text, labels = get_data(data_3, oversampling_rate)
dict_3 =  train(data_3, x_text, labels, model_type, vector_type, embed_size)'''
```

```
Loading data from file: /home/sujeendra/Desktop/data/data/formspring_data.pkl
Counter after oversampling
Counter({0: 11997, 1: 2328})
Document length : 62
Vocabulary Size: 7190
Running Model: blstm with word vector initiliazed with random word vectors.
Epoch 1/5
12892/12892 [==============================] - 38s 3ms/step - loss: 0.4071 -
acc: 0.8397
Epoch 2/5
12892/12892 [==============================] - 44s 3ms/step - loss: 0.1930 -
```

```
acc: 0.9300
Epoch 3/5
12892/12892 [==============================] - 42s 3ms/step - loss: 0.0949 -
acc: 0.9654
Epoch 4/5
12892/12892 [==============================] - 40s 3ms/step - loss: 0.0591 -
acc: 0.9813
Epoch 5/5
12892/12892 [==============================] - 39s 3ms/step - loss: 0.0420 -
acc: 0.9871
Precision: [0.99310345 0.87545788]

Recall: [0.97133221 0.96761134]

f1_score: [0.98209719 0.91923077]

[[1152   34]
 [   8  239]]
:: Classification Report
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1186
           1       0.88      0.97      0.92       247

    accuracy                           0.97      1433
   macro avg       0.93      0.97      0.95      1433
weighted avg       0.97      0.97      0.97      1433
```

[26]:
```
'x_text, labels = get_data(data_2, oversampling_rate)\ndict_2 =  train(data_2,
x_text, labels, model_type, vector_type, embed_size)\n\nx_text, labels =
get_data(data_3, oversampling_rate)\ndict_3 =  train(data_3, x_text, labels,
model_type, vector_type, embed_size)'
```

[28]:
```
transfer_learning = {
    1: transfer_learning_1,
   # 2: transfer_learning_2,
    #3: transfer_learning_3
}

#data_dict = [dict_1, dict_2, dict_3]
data_dict = [dict_1]
```

[39]:
```
def get_results(data_dict, model_type, vector_type, embed_size, ind):
    print(transfer_learning[ind](data_dict[0],data_dict[0], model_type,␣
 ↪vector_type, embed_size))
```

[40]:
```
get_results(data_dict, model_type, vector_type, embed_size, 1)
```

```
Precision: [0.99310345 0.87545788]

Recall: [0.97133221 0.96761134]

f1_score: [0.98209719 0.91923077]

[[1152   34]
 [   8  239]]
:: Classification Report
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1186
           1       0.88      0.97      0.92       247

    accuracy                           0.97      1433
   macro avg       0.93      0.97      0.95      1433
weighted avg       0.97      0.97      0.97      1433

None
```

[ ]:
```
#get_results(data_dict, model_type, vector_type, embed_size, 2)
```

[ ]:
```
#get_results(data_dict, model_type, vector_type, embed_size, 3)
```

[41]:
```
data_dict
```

[41]:
```
[{'model': <keras.engine.sequential.Sequential at 0x7fb676792f90>,
  'testX': array([[2274,  203,  381, ...,    0,    0,    0],
         [  40,  166,    8, ...,    0,    0,    0],
         [ 108,  226, 2589, ...,    0,    0,    0],
         ...,
         [  65,    1,   56, ...,    0,    0,    0],
         [   7,    1,  272, ...,    0,    0,    0],
         [   2,    7,   10, ...,    0,    0,    0]], dtype=int32),
  'testY': array([[1., 0.],
         [1., 0.],
         [1., 0.],
         ...,
         [1., 0.],
         [1., 0.],
         [1., 0.]]),
  'trainX': array([[   2,   23,   42, ...,    0,    0,    0],
         [  51,    7,   22, ...,    0,    0,    0],
         [4945,  688,    1, ...,    0,    0,    0],
         ...,
         [ 144, 1655,   21, ...,    0,    0,    0],
         [   7,    1,   77, ...,    0,    0,    0],
         [  34,   98, 1300, ...,    3,    5,    0]], dtype=int32),
  'trainY': array([[1., 0.],
```

```
        [1., 0.],
        [1., 0.],
        ...,
        [0., 1.],
        [1., 0.],
        [1., 0.]]),
  'vocab':
<tensorflow.contrib.learn.python.learn.preprocessing.text.VocabularyProcessor at
0x7fb6769cc090>,
  'length': 62,
  'data': 'formspring'}]
```

[ ]:

[ ]: