Publication Venue Suggestion in Heterogenous Graphs

Zubin Pahuja (zpahuja2@illinois.edu)

Big Picture

We build a machine-learning model to help suggest a publication venue to submit a research paper an individual researcher has written. While an intuitive baseline model would use the textual content such as paper title, our model utilizes additional features derived from a heterogenous information network on the DBLP dataset, that is constructed when scientific literature is published in venues and links are formed by publication and additional links are formed by papers citing other papers. We will then analyze the heterogenous features used in the model to understand the importance of each feature in the recommendation process.

We create a feature vector consisting of a bag-of-words text representation of the DBLP title for each paper, use these title-text features to learn a model that uses text-features to predict publication venue of a paper.

Next, we additionally create meta-path features across the heterogenous network such as a bag-of-words representation of the publication venues of each cited paper. (`Paper1 → cites → Paper2 → published_in → Venue1`). The venues of the cited papers can be used as additional features to the text-features to learn a model that uses text features and HIN-based features to predict Paper1's publication venue.

We compare prediction performance of both models using F1-Score on a test-set.

Libraries

[79]
```
import sys
import re
import csv
import logging
import warnings
import gensim
import pickle
import numpy as np

from os.path import join
from scipy.sparse import hstack
from sklearn import preprocessing, linear_model
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import f1_score, classification_report
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransfo

csv.field_size_limit(sys.maxsize)
warnings.filterwarnings('ignore')

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', l
```

Data Set

- The labels file contains a list of publication venues that are your target venues to suggest for a paper.
- The training/validation file contain five tab-delimited columns:

  ```
  Paper_Id    Paper_title    Publication_venue    Cited_Papers
  Cited_Papers_Venues
  ```

[2]
```
data_path = '/Users/zubin/Desktop/PubPredict/data'
train_filepath = join(data_path, 'train.txt')
```

1. Cleaning Data

[3]
```
def get_stopwords():
    """
    :return: dictionary of words: True for stopwords that occur less than
    """
    stopwords = {}
    word_count = {}

    with open(train_filepath, 'r', encoding='utf-8') as train_file:
        for line in train_file:
            row = line.rstrip().split('\t')
            title = row[1].lower()
            title = re.sub(r'[^a-z\s-]', '', title) # remove all but a-z

            # count tokens
            for token in title.split():
                word_count[token] = word_count.get(token, 0) + 1

    for token in word_count.keys():
        if word_count[token] < 5:
            stopwords[token] = True

    return stopwords
```

```
stopwords = get_stopwords()
```

[4]
```
def clean_data(filename):
    """
    Lowercase title, remove non-alphabet characters and tokens that occur
    :param filename: the filename of .TSV data set.
    """

    filepath = join(data_path, filename)
    output_filepath = join(data_path, 'cleaned_' + filename)

    with open(filepath, 'r', encoding='utf-8') as tsv_file, open(output_f
        cleaned_tsv_writer = csv.writer(cleaned_tsvout, delimiter='\t')

        for line in tsv_file:
            row = line.rstrip().split('\t')
            title = row[1].lower()
            title = re.sub(r'[^a-z\s\t-]', '', title) # remove all but a-
            title = ' '.join(filter(lambda x: not stopwords.get(x, False)
            row[1] = title
            cleaned_tsv_writer.writerow(row)

clean_data('train.txt')
clean_data('train_subset.txt')
clean_data('validation.txt')
clean_data('test.txt')
```

2. Create text-based feature vector for each paper

For each paper in the training, validation, and test set, title attribute is used to create a bag-of-words feature vector.

Additionally publication venue of each paper is encoded into an integer between 0 to (Number of Venues - 1).

References:

1. http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-representation
2. http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer
3. http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn.preprocessing.LabelEncoder

[5]
```
# encode labels
le = preprocessing.LabelEncoder()
```

```
    labels = []

    with open(join(data_path, 'labels.txt'), 'r', encoding='utf-8') as file:
        for line in file:
            labels.append(line.rstrip())

    encoded_labels = le.fit(labels)
```

[6]

```
    def vectorize_data(filename, ngram_range=(1, 2), tfidf=True, saveToDisk=F
        """
        Compute count feature vectors for text data and encode labels.
        :param filename: the filename of .TSV data set.
        :param ngram_range: lower and upper boundary of the range of n-values
        :param tfidf: normalize count vectors using tf-idf (default=True).
        """
        if train:
            vectorizer = CountVectorizer(ngram_range=ngram_range, tokenizer=l
            transformer = TfidfTransformer(smooth_idf=False)
        else:
            assert(vectorizer is not None)
            assert(transformer is not None)

        corpus = []
        Y = []

        filepath = join(data_path, filename)
        output_filepath = join(data_path, 'text_features_' + filename)

        with open(filepath, 'r', encoding='utf-8') as tsv_file:
            for line in tsv_file:
                row = line.rstrip().split('\t')
                try:
                    corpus.append(row[1])
                    Y.append(row[2])
                except:
                    print(row)
                    continue

            if train:
                X = vectorizer.fit_transform(corpus)
                if tfidf:
                    X = transformer.fit_transform(X)
            else:
                X = vectorizer.transform(corpus)
                if tfidf:
                    X = transformer.transform(X)

            if not test:
                Y = le.transform(Y)
                print (X.shape, Y.shape)
            else:
                Y = np.zeros(len(Y))
```

```
            if saveToDisk:
                Y2 = Y.reshape((Y.shape[0], 1))
                Z = hstack((X, Y2))
                np.savetxt(output_filepath, Z.A, delimiter=',')

            else:
                if train:
                    return X, Y, vectorizer, transformer
                return X, Y

train_X, train_Y, train_vectorizer, train_transformer = vectorize_data('c
```

```
(358429, 112092) (358429,)
```

[9]
```
vectorize_data('cleaned_train_subset.txt', ngram_range=(1, 1), saveToDisk
```

```
(10, 112092) (10,)
```

3. Classifier model for predicting venue given title features

Using SGDClassifier.

References:

1. http://scikit-learn.org/stable/tutorial/basic/tutorial.html
2. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
3. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score

Performance is evaluated on the validation set by calculating micro and macro F1 score. Additionally the precision and recall are reported per venue (class).

[10]
```
validation_X, validation_Y = vectorize_data('cleaned_validation.txt', ngr
```

```
(44629, 112092) (44629,)
```

[11]
```
clf = linear_model.SGDClassifier()
clf.fit(train_X, train_Y)

s = pickle.dumps(clf)
clf2 = pickle.loads(s)
```

Macro F1 score on validation set

[12]
```
y_pred = clf2.predict(validation_X)
f1_score(validation_Y, y_pred, average='macro')
```

```
0.19998591387183903
```

Micro F1 score on validation set

[13]
```
f1_score(validation_Y, y_pred, average='micro')
```

```
0.30419682269376414
```

[14]
```
f1_score(validation_Y, y_pred, average='weighted')
```

```
0.29035163275803644
```

Class-wise Precision Recall on validation set

[15]
```
print(classification_report(validation_Y, y_pred, target_names=le.classes
```

```
0.23      1662
                                                  icc       0.15      0.20
0.17      1380
                                                iccad       0.23      0.10
0.14       302
                                                 iccs       0.32      0.20
0.25       211
                                                 iccv       0.12      0.09
0.10       272
                                                icdar       0.56      0.68
0.62       284
                                                 icde       0.21      0.20
0.21       343
                                                icecs       0.17      0.03
0.05       293
                                                 icip       0.26      0.32
0.29       947
                                                 icis       0.27      0.04
0.07       407
                                                 icmc       0.54      0.73
0.62       435
                                                 icme       0.20      0.11
0.14       540
                                                 icml       0.16      0.12
0.14       264
                                                 icnc       0.26      0.05
0.08       210
                                                 icpr       0.12      0.02
```

```
0.04        456
                                                    icra        0.39      0.54
0.45       1793
```

Make predictions on test set

```
[17]  test_X, _ = vectorize_data('cleaned_test.txt', ngram_range=(1, 1), test=T
      y_pred = clf2.predict(test_X)
      test_pred_Y = le.inverse_transform(y_pred)
      test_data = np.loadtxt(join(data_path, 'cleaned_test.txt'), dtype=str, de
      paper_ids = test_data[:,0]
      test_predictions = np.vstack((paper_ids, test_pred_Y)).T
```

```
[18]  np.savetxt(
          join(data_path,'text_feature_predictions.txt'),
          test_predictions,
          delimiter='\t',
          fmt='%s'
      )
```

4. Using HIN features to supplement text-features

The cited venue string is concatenated to the title to create a bag-of-words feature of title + venues of cited papers.

```
[20]  def hin_vectorize_data(filename, ngram_range=(1, 2), tfidf=True, saveToDi
          """
          Compute count feature vectors for text data + cited venue strings and
          :param filename: the filename of .TSV data set.
          :param ngram_range: lower and upper boundary of the range of n-values
          :param tfidf: normalize count vectors using tf-idf (default=True).
          """
          if train:
              vectorizer = CountVectorizer(ngram_range=ngram_range, tokenizer=l
              transformer = TfidfTransformer(smooth_idf=False)
          else:
              assert(vectorizer is not None)
              assert(transformer is not None)

          corpus = []
          Y = []

          filepath = join(data_path, filename)
          output_filepath = join(data_path, 'text_hin_features_' + filename)

          with open(filepath, 'r', encoding='utf-8') as tsv_file:
              for line in tsv_file:
```

```
                row = line.rstrip().split('\t')
                try:
                    corpus.append(row[1] + ' ' + row[4])
                    Y.append(row[2])
                except:
                    print(row)
                    continue

            if train:
                X = vectorizer.fit_transform(corpus)
                if tfidf:
                    X = transformer.fit_transform(X)
            else:
                X = vectorizer.transform(corpus)
                if tfidf:
                    X = transformer.transform(X)

            if not test:
                Y = le.transform(Y)
                print (X.shape, Y.shape)
            else:
                Y = np.zeros(len(Y))

            if saveToDisk:
                Y2 = Y.reshape((Y.shape[0], 1))
                Z = hstack((X, Y2))
                np.savetxt(output_filepath, Z.A, delimiter=',')

            else:
                if train:
                    return X, Y, vectorizer, transformer
                return X, Y
```

[21]
```
train_X, train_Y, train_vectorizer, train_transformer = hin_vectorize_dat
```

```
(358429, 112168) (358429,)
```

[23]
```
hin_vectorize_data('cleaned_train_subset.txt', ngram_range=(1, 1), tfidf=
```

```
(10, 112168) (10,)
```

[24]
```
validation_X, validation_Y = hin_vectorize_data('cleaned_validation.txt',
```

```
(44629, 112168) (44629,)
```

[25]
```
clf = linear_model.SGDClassifier()
clf.fit(train_X, train_Y)

s = pickle.dumps(clf)
clf2 = pickle.loads(s)
```

Macro F1 score on validation set

```
[26] y_pred = clf2.predict(validation_X)
     f1_score(validation_Y, y_pred, average='macro')
```

```
0.7620302763937719
```

Micro F1 score on validation set

```
[27] f1_score(validation_Y, y_pred, average='micro')
```

```
0.9824329471868067
```

```
[28] f1_score(validation_Y, y_pred, average='weighted')
```

```
0.9824675480968728
```

Class-wise Precision Recall on validation set

```
[29] print(classification_report(validation_Y, y_pred, target_names=le.classes
```

```
                                bioinformatics       1.00      1.00
1.00        17
                                           cdc       1.00      1.00
1.00       843
                                           chi       1.00      1.00
1.00       418
                         chi_extended_abstracts       1.00      1.00
1.00       568
                                          cikm       0.92      0.86
0.89       377
                                        cogsci       1.00      1.00
1.00       434
                                        coling       1.00      1.00
1.00       367
                                  commun._acm       0.43      0.09
0.15        68
                                      compsac       1.00      1.00
1.00       296
                        comput._graph._forum       0.00      0.00
0.00         0
                                   comput._j.       0.33      0.05
0.09        19
                        computer_communications       0.00      0.00
0.00         0
```

```
                                computer_networks       0.33      0.50
0.40          2
                                             corr       0.00      0.00
0.00          0
                                             cvpr       0.99      0.89
```

Make predictions on test set

[30]
```
test_X, _ = hin_vectorize_data('cleaned_test.txt', ngram_range=(1, 1), tf
y_pred = clf2.predict(test_X)
test_pred_Y = le.inverse_transform(y_pred)
test_data = np.loadtxt(join(data_path, 'cleaned_test.txt'), dtype=str, de
paper_ids = test_data[:,0]
test_predictions = np.vstack((paper_ids, test_pred_Y)).T
```

[31]
```
np.savetxt(
    join(data_path,'text_hin_feature_predictions.txt'),
    test_predictions,
    delimiter='\t',
    fmt='%s'
)
```

5. Analysis

[32]
```
validation_data = np.loadtxt(join(data_path, 'cleaned_validation.txt'), d
venues = validation_data[:,2]
cited_venues = [venues.split(' ') for venues in validation_data[:,4]]

ctr = 0.
for i in range(len(venues)):
    if venues[i] in cited_venues[i]:
        ctr += 1.

print("Percentage of papers published in venues that they cite papers at
```

```
Percentage of papers published in venues that they cite papers at is
96.16392928364965
```

Above statistic shows how important cited venues are because papers are usually published in the same venue as the papers they cite.

We observe that combining HIN-based features with text-based features greatly outperforms model that uses text-based features alone at both precision, recall and hence F-1 score. This is because textual content of title hardly gives us any information about the venue it may be suitable for, as well as there are several venues for the same research domain. It is plausible

that author cites his own paper, reads papers of a venue and submits to the venue of his/her preference. Cited venue also demonstrates the research domain of paper better than textual content of its title. This is because titles are sometimes intended to be catchy but are not very informative.

Therefore, heterogenous information networks contain richer information than simple text-based model. Concatenating cited venue strings in title essentially adds them to tokens in bag-of-words vector. Since venue names are unique and distinct from words found in vocabulary of paper titles, therefore their occurence in title name essentially serves as concatenating title bag-of-words vector with one-hot-vector-encoding of cited paper's venues. Thus, our model incorporates the meta-path `Paper1 → cites → Paper2 → published_in → Venue1`. This illustrates that more than content, context matters as well. I believe including more metapaths will enrich this model even more.

We also observe in later section, even with word embeddings, text is much less important than cited venues in determining venue of a publication.

The downside of bag-of-words unigram model is:

1. Features vectors are rather large and sparse, which results in slow training
2. Our model only uses word counts and for example, cannot differentiate between a statement and a question that use the same words, or a random permutation of the same words/ phrases which could mean something entirely else. Therefore, a better idea would be to use bi-grams or tri-grams, but this would explode the feature vector length.
3. Text features used are a simple bag-of words, as such similar words are not considered when doing classification. Contextual similarity can be incorporated using word embeddings.

Ways to improve:

1. Add more heterogeneity such as author nodes which will encode author's preferences to publish at a certain venue.
2. Add more metapaths such as `Paper-> Keywords -> Venue`
3. Longer metapaths that include cited venues of cited papers as well.
4. Better data cleaning such as removing stopwords.
5. Reduce feature vector length by using word embeddings such as word2vec.
6. Count Vectors should be tf-idf weighted since not all words carry same information.
7. Rather than unigram bag of words model, use bi-grams or tri-grams.

6. More Sophisticated Features

The text features used are a simple bag-of words, as such similar words are not considered when doing classification. Word embedding with word2vec is performed to utilize embedding features for the model.

[33]
```
train_data = np.loadtxt(join(data_path, 'cleaned_train.txt'), dtype=str,
validation_data = np.loadtxt(join(data_path, 'cleaned_validation.txt'), d
test_data = np.loadtxt(join(data_path, 'cleaned_test.txt'), dtype=str, de
```

[34]
```
# train word2vec on titles
word2vec_size = 200
train_titles = [title.split() for title in train_data[:,1]]
word2vec_model = gensim.models.Word2Vec(train_titles, min_count=5, size=w
```

```
2018-05-01 17:22:48,280 : INFO : collecting all words and their counts
2018-05-01 17:22:48,282 : INFO : PROGRESS: at sentence #0, processed 0
words, keeping 0 word types
2018-05-01 17:22:48,306 : INFO : PROGRESS: at sentence #10000, processed
94688 words, keeping 13386 word types
2018-05-01 17:22:48,329 : INFO : PROGRESS: at sentence #20000, processed
190049 words, keeping 20160 word types
2018-05-01 17:22:48,361 : INFO : PROGRESS: at sentence #30000, processed
284608 words, keeping 25601 word types
2018-05-01 17:22:48,414 : INFO : PROGRESS: at sentence #40000, processed
379019 words, keeping 30240 word types
2018-05-01 17:22:48,450 : INFO : PROGRESS: at sentence #50000, processed
474081 words, keeping 34482 word types
2018-05-01 17:22:48,489 : INFO : PROGRESS: at sentence #60000, processed
568406 words, keeping 38446 word types
2018-05-01 17:22:48,539 : INFO : PROGRESS: at sentence #70000, processed
663009 words, keeping 42211 word types
2018-05-01 17:22:48,573 : INFO : PROGRESS: at sentence #80000, processed
758268 words, keeping 45648 word types
2018-05-01 17:22:48,608 : INFO : PROGRESS: at sentence #90000, processed
853726 words, keeping 49055 word types
2018-05-01 17:22:48,645 : INFO : PROGRESS: at sentence #100000, processed
948877 words, keeping 52213 word types
2018-05-01 17:22:48,686 : INFO : PROGRESS: at sentence #110000, processed
1044116 words, keeping 55290 word types
2018-05-01 17:22:48,719 : INFO : PROGRESS: at sentence #120000, processed
1139489 words, keeping 58202 word types
2018-05-01 17:22:48,749 : INFO : PROGRESS: at sentence #130000, processed
1234580 words, keeping 61117 word types
2018-05-01 17:22:48,781 : INFO : PROGRESS: at sentence #140000, processed
```

[68]
```
def word2vec_vectorize_data(filename, word2vec_model, saveToDisk=False, t

    X = []
    Y = []

    filepath = join(data_path, filename)
    output_filepath = join(data_path, 'text_word2vec_features_' + filenam

    with open(filepath, 'r', encoding='utf-8') as tsv_file:
        for line in tsv_file:
            row = line.rstrip().split('\t')
            words = [token for token in row[1].split(' ') if token in wor

            w2v_words = np.zeros(word2vec_size)
            if len(words) != 0:
                w2v_words = word2vec_model[words]
```

```
                    w2v_words = np.mean(w2v_words, axis=0)
                X.append(w2v_words)
                Y.append(row[2])

        X = np.array(X)

        if not test:
            Y = le.transform(Y)
            print (X.shape, Y.shape)
        else:
            Y = np.zeros(len(Y))

        if saveToDisk:
            Y2 = Y.reshape((Y.shape[0], 1))
            Z = np.concatenate((X, Y2), axis=1)
            np.savetxt(output_filepath, Z, delimiter=',')

        else:
            return X, Y
```

[69]
```
train_X, train_Y = word2vec_vectorize_data('cleaned_train.txt', word2vec_
```

```
(358429, 200) (358429,)
```

[70]
```
word2vec_vectorize_data('cleaned_train_subset.txt', word2vec_model, saveT
```

```
(10, 200) (10,)
```

[71]
```
validation_X, validation_Y = word2vec_vectorize_data('cleaned_validation.
```

```
(44629, 200) (44629,)
```

[72]
```
clf = linear_model.SGDClassifier()
clf.fit(train_X, train_Y)

s = pickle.dumps(clf)
clf2 = pickle.loads(s)
```

Macro F1 score on validation set

[73]
```
y_pred = clf2.predict(validation_X)
f1_score(validation_Y, y_pred, average='macro')
```

```
0.11890360989735102
```

Micro F1 score on validation set

[74]
```
f1_score(validation_Y, y_pred, average='micro')
```

```
0.22043962445943222
```

Class-wise Precision Recall on validation set

[75]
```
print(classification_report(validation_Y, y_pred, target_names=le.classes
```

```
                                                precision    recall  f1-
score   support

                                          aaai       0.11      0.13
0.12       528
                                         aamas       0.24      0.06
0.09       349
                                           acc       0.00      0.00
0.00       109
                                acm_multimedia       0.14      0.08
0.10       379
                             acm_trans._graph.       0.00      0.00
0.00         1
                                         amcis       0.10      0.03
0.05       665
                                          amia       0.33      0.03
0.06       121
                                       asp-dac       0.20      0.01
0.01       319
                                bioinformatics       0.00      0.00
0.00        17
                                           cdc       0.34      0.56
0.43       843
                                           chi       0.13      0.09
0.10       418
                         chi_extended_abstracts       0.15      0.17
0.16       568
                                          cikm       0.09      0.20
0.12       377
                                         cogsci       0.49      0.10
```

Make predictions on test set

[76]
```
test_X, _ = word2vec_vectorize_data('cleaned_test.txt', word2vec_model, t
y_pred = clf2.predict(test_X)
test_pred_Y = le.inverse_transform(y_pred)
test_data = np.loadtxt(join(data_path, 'cleaned_test.txt'), dtype=str, de
paper_ids = test_data[:,0]
test_predictions = np.vstack((paper_ids, test_pred_Y)).T
```

[77]
```
np.savetxt(
    join(data_path,'text_word2vec_feature_predictions.txt'),
    test_predictions,
    delimiter='\t',
    fmt='%s'
)
```

6.1 Word2Vec Embeddings with HIN Features

[184]
```
def one_hot_encode(labels_space_delimited):
    encoding = np.zeros(len(labels))
    labels_ = labels_space_delimited.split(' ')
    label_indices = []

    for label in labels_:
        try:
            idx = le.transform([label])[0]
            label_indices.append(idx)
        except:
            continue

    for i in label_indices:
        encoding[i] += 1.

    return encoding
```

[189]
```
def hin_word2vec_vectorize_data(filename, word2vec_model, saveToDisk=Fals

    X = []
    Y = []

    filepath = join(data_path, filename)
    output_filepath = join(data_path, 'text_word2vec_hin_features_' + fil

    with open(filepath, 'r', encoding='utf-8') as tsv_file:
        for line in tsv_file:
            row = line.rstrip().split('\t')
            words = [token for token in row[1].split(' ') if token in wor

            w2v_words = np.zeros(word2vec_size)
            if len(words) != 0:
                w2v_words = word2vec_model[words]
                w2v_words = np.mean(w2v_words, axis=0)

            w2v_words = np.concatenate((w2v_words, one_hot_encode(row[4])
            X.append(w2v_words)
            Y.append(row[2])
```

```
            X = np.array(X)

            if not test:
                Y = le.transform(Y)
                print (X.shape, Y.shape)
            else:
                Y = np.zeros(len(Y))

            if saveToDisk:
                Y2 = Y.reshape((Y.shape[0], 1))
                Z = np.concatenate((X, Y2), axis=1)
                np.savetxt(output_filepath, Z, delimiter=',')

            else:
                return X, Y
```

[188]
```
train_X, train_Y = hin_word2vec_vectorize_data('cleaned_train.txt', word2
```

```
(358429, 316) (358429,)
```

[196]
```
hin_word2vec_vectorize_data('cleaned_train_subset.txt', word2vec_model, s
```

```
(10, 316) (10,)
```

[197]
```
validation_X, validation_Y = hin_word2vec_vectorize_data('cleaned_validat
```

```
(44629, 316) (44629,)
```

[198]
```
clf = linear_model.SGDClassifier()
clf.fit(train_X, train_Y)

s = pickle.dumps(clf)
clf2 = pickle.loads(s)
```

Macro F1 score on validation set

[199]
```
y_pred = clf2.predict(validation_X)
f1_score(validation_Y, y_pred, average='macro')
```

```
0.7887026160652078
```

Micro F1 score on validation set

[200]
```
f1_score(validation_Y, y_pred, average='micro')
```

0.9783772883102915

Class-wise Precision Recall on validation set

[201]
```
print(classification_report(validation_Y, y_pred, target_names=le.classes
```

```
                                                precision    recall  f1-
score   support

                                          aaai       0.83      0.94
0.88       528
                                         aamas       1.00      1.00
1.00       349
                                           acc       1.00      1.00
1.00       109
                                acm_multimedia       1.00      1.00
1.00       379
                             acm_trans._graph.       0.00      0.00
0.00         1
                                         amcis       1.00      1.00
1.00       665
                                          amia       1.00      1.00
1.00       121
                                       asp-dac       1.00      1.00
1.00       319
                                bioinformatics       1.00      1.00
1.00        17
                                           cdc       1.00      1.00
1.00       843
                                           chi       1.00      1.00
1.00       418
                       chi_extended_abstracts       1.00      1.00
1.00       568
                                          cikm       0.72      0.86
0.79       377
                                        cogsci       1.00      1.00
```

Make predictions on test set

[202]
```
test_X, _ = hin_word2vec_vectorize_data('cleaned_test.txt', word2vec_mode
y_pred = clf2.predict(test_X)
test_pred_Y = le.inverse_transform(y_pred)
test_data = np.loadtxt(join(data_path, 'cleaned_test.txt'), dtype=str, de
paper_ids = test_data[:,0]
test_predictions = np.vstack((paper_ids, test_pred_Y)).T
```

[203]
```
np.savetxt(
```

```
        join(data_path,'text_word2vec_hin_feature_predictions.txt'),
        test_predictions,
        delimiter='\t',
        fmt='%s'
    )
```

6.2 TF-IDF Weighted Bag of Words Model

[205]
```
train_X, train_Y, train_vectorizer, train_transformer = vectorize_data('c
```

```
(358429, 112092) (358429,)
```

[207]
```
vectorize_data('cleaned_train_subset.txt', ngram_range=(1, 1), saveToDisk
```

```
(10, 112092) (10,)
```

[209]
```
validation_X, validation_Y = vectorize_data('cleaned_validation.txt', ngr
```

```
(44629, 112092) (44629,)
```

[210]
```
clf = linear_model.SGDClassifier()
clf.fit(train_X, train_Y)

s = pickle.dumps(clf)
clf2 = pickle.loads(s)
```

Macro F1 score on validation set

[212]
```
y_pred = clf2.predict(validation_X)
f1_score(validation_Y, y_pred, average='macro')
```

```
0.22451030875267966
```

Micro F1 score on validation set

[213]
```
f1_score(validation_Y, y_pred, average='micro')
```

```
0.31983687736673466
```

Class-wise Precision Recall on validation set

[214]
```
print(classification_report(validation_Y, y_pred, target_names=le.classes
```

```
                                                    precision    recall  f1-
score   support

                                              aaai       0.09      0.07
0.08       528
                                             aamas       0.32      0.40
0.36       349
                                               acc       0.05      0.03
0.04       109
                                    acm_multimedia       0.23      0.22
0.23       379
                               acm_trans._graph.       0.00      0.00
0.00         1
                                             amcis       0.31      0.24
0.27       665
                                              amia       0.44      0.60
0.51       121
                                           asp-dac       0.16      0.13
0.14       319
                                    bioinformatics       0.06      0.06
0.06        17
                                               cdc       0.43      0.67
0.53       843
                                               chi       0.26      0.24
0.25       418
                          chi_extended_abstracts       0.39      0.36
0.38       568
                                              cikm       0.12      0.11
0.12       377
                                            cogsci       0.47      0.61
```

Make predictions on test set

[215]
```
test_X, _ = vectorize_data('cleaned_test.txt', ngram_range=(1, 1), test=T
y_pred = clf2.predict(test_X)
test_pred_Y = le.inverse_transform(y_pred)
test_data = np.loadtxt(join(data_path, 'cleaned_test.txt'), dtype=str, de
paper_ids = test_data[:,0]
test_predictions = np.vstack((paper_ids, test_pred_Y)).T
```

[216]
```
np.savetxt(
    join(data_path,'text_feature_tfidf_predictions.txt'),
    test_predictions,
    delimiter='\t',
    fmt='%s'
)
```