Advanced Data Mining Final Project Report

Abstract:
The goal of the project is to build a framework for the analysis of big historic text data using various data mining techniques. As a case study, this framework is applied on a text collections comprised of 90,000 volumes that mentioned the "Chicago School" during 1850-2010. Several tools for data classification, summarization and filtering are used. My work focused on classification. I deployed the supervised method on text snippets. In addition, I built an ensemble classifier that included the results from other classifiers that were run on other, independent types of data and were provided by other students. Finally the ensemble classifier made it possible to re-assess the classification accuracy of the previous classifier (on the test data), and estimate the final ensemble classification accuracy of roughly 60% (45-84, depending on chosen model). The various narratives of "Chicago School" can be now disambiguated, and new collections of books can be built from the classification results.

Data:

- NER and Wikifier results for collection of 100,000 books (architectural and cultural history). For each book, we had a JSON file which contained below information:
  1. Structured data extracted through NER
  2. Structured data extracted through entity linking
- Extracted text snippets.
- Metadata of the books – a JSON file per book; holds information about the book.

Wikification is the task of identifying and linking expressions in text to their referent Wikipedia pages. The wikifier also identifies the name entities referred in the book.

The wikifier results on a book is a JSON object for the corresponding book (format is shown below).

{"id":"mdp.39015064582888","pages":[{"pid":"00000001","wikifier":[…],"ner":[…]} , …]}

"id" is the book id.

"pages" is the list of all pages in the book. Each page is uniquely mentioned by a page id (pid) contains the wikifier and NER tags in the page.

"wikifier" is a list of all expressions with some metadata about their occurrence in the page of the book.

NER is the list of all name entities referred in the page and few metadata about the occurrence.

An example of an item in wikifier is shown below:

{"text":"COLLEGE","link":"http://en.wikipedia.org/wiki/College","start":0,"end":7}

“text” is the entity of interest, “link” is the Wikipedia link for that expression, “start” and “end” are the start and end indices of the entity’s occurrence in the page.

An example of an item in NER list is shown below:

{"text":"VAN DYKE","type":"PER","start":43,"end":51}

“text” is a named entiry, “type” is any of PER/LOC/ORG/MISC, “start” and “end” are the start and end indices of the entity’s occurrence in the page.

An example of a text snippet:

“in the galleries and class-room to those who have applied. But no systematic effort has been made to introduce all Chicago school children to the Art Institute and to provide for them suitable instruction about the collections. To devise the best means”

Experiments:

1) Classification of Snippets

Provided Data:

190’000 Snippets, with Book ID and Page ID

10% labeled data; roughly 200 labels.

Classification algorithms used:

I used below classifiers for classification of text snippets:

1. Logistic Regression using tfidf for bigrams
2. Neural Networks using tfidf for bigrams
3. Support Vector Machine using tfidf for bigrams
4. Random Forest using tfidf for bigrams
5. Decision Tree classifier using tfidf for bigrams
6. LinearSVC with word2vec features; word2vec model trained on brown corpus.

Classifier Parameters:

I exhaustively tried the classifiers with different combinations of parameters i.e, performed Grid Search. Below is the parameter setting that worked best for each of the classifiers

1. Logistic Regression : C=100, class_weight = “balanced”, max_itr = 100

   C : regularization parameter;

   class_weight : balanced; weights the instance inversely proportional to their class frequencies.

   max_iter : 100; maximum 100 iterations till convergence.

2. Neural Networks : hidden_layer_sizes=(5,10), activation="logistic", solver="lbfgs", alpha=0.01, learning_rate="constant", max_iter=300, shuffle=False, early_stopping=True, random_state=123

   hidden_layer_sizes : (5,10); two hidden layers with number of neurons 5 and 10 resp.

   activation : logistic; uses logistic function for activation for all the neurons hidden layer

   solver : lbfgs; solver for weight optimization through backpropagation

   learning rate : constant; learning rate remains same for all the update of weights.

   alpha : 0.01; penalty parameter to be used during updation for l2 regularization.

3. Support Vector Machine : C=10, class_weight = “balanced”
   C : 10; penalty parameter for the error term, to be used to learn feature weights.

   class_weight : “balanced”; same as explained above

4. Random Forest : n_estimators=10,criterion="gini", max_features=None, max_depth=None, min_samples_split=2, min_samples_leaf=1, max_leaf_nodes=None, class_weight="balanced", random_state=123
   n_estimators : 10; build 10 different decision trees on the sample of data

   criterion : “gini”; to calculate which feature to split on split on at each level

max_features : None; maximin features to consider when looking for a best split

max_depth : None; maximum depth a tree can achieve

min_samples_split : 2; minimum samples required to split a node further

min_samples_leaf : 1; minimum samples that should be present at each leaf node

max_leaf_nodes : None; limits the number of leaf nodes in the tree

class_weight : “balanced” ; same as explained above

5. Decision Tree : criterion="gini", splitter="best", max_features=None, max_depth=None,min_samples_split=2, min_samples_leaf=1, max_leaf_nodes=None, class_weight="balanced", random_state=123, presort=False

   presort: False; should the data be sorted or not? can help in finding best split faster.

Implementation:

I used the scikit implementation of the above mentioned models. But, I did an extensive grid search of the parameters of each of the models to find the best parameter settings that works for my data i.e, the text snippets.

Evaluation/Accuracy Estimation:

For evaluating the model performances, I used the cross validation accuracy score of the classifiers.

Steps:

I used the below machine learning workflow to build the models:

1. Collect the data and preprocess it. I removed the stop words, punctuations, removed less frequent words. Compute ngrams. Remove rare ngrams. (bigrams were finally chosen)
2. Find the tfidf of the bigram in the data.

3. Split the data into two parts, train and test. Build the model on the train data.
4. Use the cross validation score to evaluate model accuracy and fine tune.
5. Run the model for the unseen test data to predict labels. Since we didn't have to true labels known for these data, we could not calculate the test accuracy. But, by checking the features and feature weights of the prediction models, we got to know that the model is performing well.

Estimated accuracies:

Model	Accuracy score train data shape : (164,19) (limited features by adding min_df in tfidfvectorizer)	Accuracy score train data shape : (1450,60) (limited featured by adding max_features to 60 in tfidfvectorizer)	Accuracy score train data shape : (24894,9672)
Logistic Regression	0.646	0.84688	0.835
Neural Networks	0.628	0.86623	0.4788
Linear SVM	0.571	0.84619	0.864
Random Forest	0.511	NA	0.83
Decision Tree	0.53	NA	0.79
Word2Vec	0.683	NA	0.891 (number of features increased to 211938)

Model analysis:

I used some of the available model features to understand how the model is performing and fine-tuned the model (source code is shown in appendix).

1. Printed out the features considered by the tfidf vectorizer.
2. Printed out the features weights of the features of every model to understand which of the features are important and not important for the classes.

By performing this in depth analysis of the model, I got to understand where the model my

models were performing well and poor, and could further fine tune the models

It is clear from the above table that Logistic Regression and Linear SVM with linear kernel performed well for my data as they make linear transformation of the data by learning the weights.

Even the DecisionTreeClassifier and the RandomForest (ensemble) classifier performed well with more data samples.

Neural Networks did well with lesser data and lesser features as it learnt the important features pretty well. But, once the data size increased and there was not restriction on the features to be used in the model, it performed poorly as unwanted features confused the model.

The best model is the word2vec model trained on brown corpus and using LinearSVC for classification.

```
w2v_model = Word2Vec(brown.sents(),size=50,window=5,min_count=1,workers=2)

w2v_model_wv = w2v_model.wv

del w2v_model
```

The features are built using the context of the word i.e. for each word along with its word2vec features, I used the previous and next words word2vec features. These helped in learning the context of the words better and also learn similar words that occur in similar context. Also, the word2vec model addresses the sparsity issue, which bigram model suffers from.

2) Ensemble Classification

Input:

Results from classification run on three types of data.

Classifier	Data used	Classifier type	Est. accuracy
S	text snippets of 42 words	**LR**, RF, NN	88%
N	NER results from one page of text	LR, **RF**, NN	65%
W	Wikifier results from one page of text	LR, **RF**, NN	76%

Procedure:

Based on the available results, take majority vote. If no majority vote available, take accuracy vote (source code is shown in appendix).

Output:

Ensemble classification (E). Re-assessed accuracy for classifiers S/N/W on test data. Estimated accuracy of classifier E.

Classifier matches:

All three classifiers match: 10% – taking majority vote

Two classifier match: 35% – taking majority vote

No two classifier match: 55% – taking accuracy vote, setting accuracy threshold

Re-assessing accuracies of S/N/W:

Through a mathematical model provided as part of the data mining project, the accuracy of previous classifier is re-assessed, and the new classification accuracy is estimated.

The previously estimated accuracy for the three classifiers was similar. The model therefore assumes that the three classifiers have the same accuracy. Given this assumption, is is possible to estimate the number of matches as follows:

p = average accuracy of classifiers / probability that the classifier predicts the label correctly

n = number of labels

Model conclusions:

(1-p)/(n-1) = probability that the classifier predicts any of the other (wrong) labels

All three classifier match: p^3

Two classifier match: 3*p^2(1-p)

Only one classifier is correct: 3*p(1-p)^2

All classifier are wrong: (p-1)^3

Based on this model, we can compute in our present case:

$p = 0.45$

Two classifier match, but the result is still wrong:

0.004%

All classifiers are wrong:

16%

We conclude that the match of two classifiers will give 100% correct results. Given that the match between two classifier occurs in 45% of the cases, for many implementations, this is a statistically relevant amount of data to draw conclusions. If it is enough to classify 45% of the data, an accuracy of 100% (theoretically) can be reached. On the other hand, if all data needs to be classified, and if the accuracy vote is always taken to find the correct result (which we cannot assume) the maximal accuracy might reach up to 84%, but most probably will lie below this value because the accuracy vote is not expected to always be correct.

Conclusion:

We succeeded in our goal of building the framework to analyse a big set of text data by using data mining techniques such as data summarization, filtering and classification.

With my experiments, I was successful disambiguating the mentions of Chicago School to one among the 200 possible labels by using just 10% of the actual data with above 80% accuracy. This is quite good results considering the side of the test data that we used as it is a huge collection which cannot be manually done by humans or it is very tedious task to be precise. Also, it is evident from our results that we achieve better results by building ensemble model by combining independent models.

Working on this project I learnt a lot by applying what I had learnt in classes. I got to understand in depth how the classifiers work (especially those I have used). One important thing that I learnt from this project is the importance of understanding the data that we have to work on, importance of data preprocessing, importance of fine tuning the classifiers to achieve better results by using parameter tuning.

I finally would like to thank my professor Irina Matveeva for having this course project. I learnt a lot in this process. One great thing about this project is the group size, as we were 6 students, each of us learnt from each other by working on independent tasks which has different data mining technique applied. I am sure it would have taken me a lot more time to learn what I have learnt in last three months if I was studying or working on this alone.

Another thing that helped us in learning is the frequent progress reports. The kind of questions that professor asked in the reports pushed us to dig in deep into the techniques and understand why it works or why it does not works. The reports made us think, rather than just code to solve a problem.

Appendix:

- Ensemble learning :

```
def makeFinalPrediction( preds_d ):
  for key in preds_d.keys():
     #all three classifiers agree on the label
     if preds_d[key]["s"]["l"] == preds_d[key]["w"]["l"]  and preds_d[key]["s"]["l"] == preds_d[key]["n"]["l"]:
        preds_d[key].update( { "f" : { "l" : preds_d[key]["s"]["l"], "p" : 1 } } )
     #snippets and wikifier agree on the label
     elif preds_d[key]["s"]["l"] == preds_d[key]["w"]["l"] :
        preds_d[key].update( { "f" : { "l" : preds_d[key]["s"]["l"], "p" : 2/3 } } )
     #snippets and NER agree on the label
     elif preds_d[key]["s"]["l"] == preds_d[key]["n"]["l"] :
        preds_d[key].update( { "f" : { "l" : preds_d[key]["s"]["l"], "p" : 2/3 } } )
     #NER and wikifier agree on the label
     elif preds_d[key]["w"]["l"] == preds_d[key]["n"]["l"] :
        preds_d[key].update( { "f" : { "l" : preds_d[key]["n"]["l"], "p" : 2/3 } } )
     #None of the three agree on the label, so predict based on the confidence
     else:
        #snippet is more confident
        if ( preds_d[key]["s"]["p"] > preds_d[key]["w"]["p"] ) and ( preds_d[key]["s"]["p"] > preds_d[key]["n"]["p"] ):
           preds_d[key].update( { "f" : { "l" : preds_d[key]["s"]["l"], "p" : 1/3 } } )
        #ner is more confident
        elif ( preds_d[key]["n"]["p"] > preds_d[key]["s"]["p"] ) and ( preds_d[key]["n"]["p"] > preds_d[key]["w"]["p"]
):
           preds_d[key].update( { "f" : { "l" : preds_d[key]["n"]["l"], "p" : 1/3 } } )
        #wikifier is more confident or all are equally confident of their respective results
        else:
           preds_d[key].update( { "f" : { "l" : preds_d[key]["w"]["l"], "p" : 1/3 } } )
  return preds_d
```

- Grid Search to find the best parameter settings for the model:

```
for c in Cs:
   for cw in CW:
      for itr in ITR:
         print("C : ",c,", class_weight : ",cw,", max_iter : ",itr)
         clf_lr = LogisticRegression(class_weight=cw,C=c,max_iter=itr,random_state=123)
         acc = cross_val_score(clf_lr,train,Y,scoring="accuracy",cv=cv)
```

- Word2Vec model with LinearSVC classifier:

```
from sklearn.feature_extraction import DictVectorizer
#build feature dictionary with wor2vec vectors and the token
train_dicts = make_feature_dicts(X, w2v_model_wv, w2v=True, token=True, caps=False, context=True)
vec = DictVectorizer()
X_train_v = vec.fit_transform(train_dicts)

clf_linear_svc = LinearSVC(C=c, random_state=123, class_weight=cw, max_iter=itr)
acc = cross_val_score(clf_linear_svc, X_train_v, Y,scoring="accuracy", cv=cv)
clf_linear_svc.fit(X_train_v, Y)
```

- Model Evaluation:

```
for i, cls in enumerate(clf_lr.classes_):
   print("\nFeature weights for class : ",cls,"\n")
   df = pd.DataFrame(data= {"Features" : tfidf.get_feature_names(), "weights" : clf_lr.coef_[i]})
   df = df.sort_values(axis=0,by='weights',ascending=False)
   print(df)
```

- tfidf vectorizer:

```
max_features = None #None means no limit

#which n-gram model to use? if low is 2 and high is 5, we will consider all 2-grams to 5-grams
low = 2
high = 2

#word occurences
min_freq = 6 #np.ceil(len(X) / 10)
max_freq = np.ceil(len(X) / 1.5)

tfidf = TfidfVectorizer(input='content', encoding='utf-8', decode_error='ignore', strip_accents='ascii',\
                        ngram_range=(low,high), stop_words='english', max_features=max_features, norm='l1',\
                        max_df=max_freq, min_df=min_freq)
train = tfidf.fit_transform(X)

print(len(tfidf.get_feature_names()))
```