4^{th} Credit Final Project

Yizhi Fang

1 Introduction

Given a dataset of 430 paper, I would like to mine insightful knowledge to help people to understand the data science field without having to read the paper themselves. In order to achieve the useful knowledge from diverse aspects, I proposed 3 different data mining tasks based on various motivations arising from questions people may ask in real world. Table 1 shows a complete list of file structure in the project folder as reference including the data, Python codes and outputs. To run the whole project, run `main.py` with specific path to the whole project folder. For example, in my computer, I could run:

```
$ python main.py --base /Users/yzhfang/Documents/Course/Intro to Data Mining/
    Summer2017/Project
```

It may also be convenient to change the path in the `main.py` once for all which is defined inside the function `parse_options` shown below:

```
def parse_options():
    optparser.add_argument(
        "-b", "--base",
        dest="base",
        help="base folder to the project folder",
        default="/Users/yzhfang/Documents/Course/Intro to Data Mining"
        "/Summer2017/Project"    ⇐ Change default path here
        )
        ...
```

2 Data cube construction

After manually reading and annotating 20 paper myself, I categorized the keywords of each paper into 4 dimensions according to subjective judgments, i.e. each keyword belongs to one of Problem, Method, Dataset or Metric. The annotations of 20 paper is saved in `annotated.csv`. With all annotated dimension values from 20 paper, I could then partially annotate all 430 paper by iteratively searching through the paper content for each value. The code can be seen in `data_process.py` and it will return 4 variables: (1) a dictionary with values as keys and lists of corresponding paper (`value2paper`) and (2) a dictionary of value and its dimension (`value2dim`) (3) a list of 2-nearest neighbor words in all paper for

Table 1: File structure

data\	Raw paper
kdd15\	KDD paper in 2015
kdd16\	KDD paper in 2016
code\	Python scripts
data_process.py	Annotate all paper
cube_construction.py	Construct data cube
apriori.py	Frequent pattern mining with Apriori
kmeans.py	K-Means clustering for Methods
svm.py	Logistic SVM with SGD training
main.py	Main script
output\	All output files
annotated.csv	Annotated 20 paper by hand
annotated_all.pkl	Annotated 430 paper
Dim_Table.csv	Constructed dimension table and count
Freq_Patterns.csv	Frequent patterns
Associ_Rules.csv	Association rules
KMeans_Method.pdf	K-Means result figure
WordCloud_Topic0.pdf	Word cloud for topic 0
WordCloud_Topic1.pdf	Word cloud for topic 1
WordCloud_Topic2.pdf	Word cloud for topic 2
WordCloud_Topic3.pdf	Word cloud for topic 3
F1_Scores.pdf	F1 score plot for classification

each value (`neighbors_list`) and (4) a list of its corresponding dimension (`target`). The fist 2 dictionaries will be used to construct the data cube and future reference whenever needed while the later 2 lists will be used in classification task. It usually takes some time to search through all 430 paper therefore I saved these 4 variables in `annotated_all.pkl` such that once this file is created `main.py` doesn't need to go through this process anymore.

The data cube is built with the following concepts:

1. The cube has 4 dimensions: Problem, Method, Dataset and Metric.
2. Each dimension contains only 1 hierarchy level but multiple values for each paper.
3. The count of each cell is the number of paper that contains its all dimension values.

A base cell, for example, looks like (recommendation, feature extraction, realworld data, outperforms): 2. After annotating 430 paper, I noticed that the numbers of possible base cells are very large, i.e. $60 \times 62 \times 33 \times 28 = 3437280$ in total so it could be time and space consuming to write down all possible cells. In order to construct the cube efficiently, I introduced the iceberg condition with `min_support = 2` such that any base cells with count less than 2 will be discarded. The constructed data cube is saved in `Data_Cube.csv` with first 4 columns as dimensions and last column is the count of base cell. Generally, cube construction is only meant to be done once at the beginning therefore one has to specify when running the `main.py` to construct data cube:

```
$ python main.py --cube 1
```

3 Task 1- Frequent Pattern

The very first question people may ask about any field is what commonly discussed problems, methods or metrics are to have a sense of current trend in this field. Frequent patterns and strong association rules discovery in the whole annotated dimension values set will be able to tell. In this problem, each paper is a transaction and every dimension value is an item. I want to find frequent pair of (Problem, Method) or (Method, Metric).

In this project, I used Apriori algorithm to mine frequent pattern with `min_support = 0.025` and `min_confidence = 0.8`. To evaluate the frequent patterns, I chose the null-invariant Kulc measure (1 means positive association and 0 means negative) and Imbalance Ratio (1 means very skewed and 0 means perfectly balanced) to avoid asymmetric case which are both listed in `Associ_Rules.csv` together with the `support` and `confidence` for each strong association rule and the frequent patterns are saved in `Freq_Patterns.csv`.

After mining the frequent patterns, many strong association rules are like, for instance, {social network | deep learning | gradient descent} $\Rightarrow$ {outperforms} or {gradient descent | matrix factorization} $\Rightarrow$ {outperforms}. This is good but not very insightful because it is not surprising that the methods in paper would outperforms their competitors in some way.

This is not exactly what I expected when I designed the task so I made some modifications when generating the initial transaction list. Originally I have all 430 paper as my transaction list so there are many not-so-meaningful patterns like {outperforms}, {effectiveness}, {robust} etc..Therefore this time I tried only include paper related to "neural network" and "deep learning" because these two are among the hot topics in data science field and one

could find strongly related patterns with these two topics. Since the second transaction list is more related compared to the first one, I increased the `min_support` to 0.1 accordingly.

This time more meaningful patterns start to show up and selected frequent patterns and strong association rules are in shown in Table 2 (refer to `Associ_Rules.csv` and `Freq_Patterns.csv` for the complete list). According to Table 2, my conclusion for the field of neural network and deep learning is following:

1. Neural network and deep learning could be frequently applied to recommendation, social network, Informatics and Natural Language Processing;
2. The related methods are gradient descent, random forest, decision tree and feature extraction indicating they can all be employed to solve similar problems.
3. Wikipedia is the most common used dataset in neural network and deep learning area.
4. People like to describe their models with terms like outperforms, confidence, effectiveness, efficiency, scalability and robust.
5. Paper that uses matrix factorization to solve social network often talks about recommendation or vise versa.

Table 2: Selected frequent patterns and strong association rules

Association rule	Kulc	IR
recommendation ⇒ neural network	0.586466	0.6
social network ⇒ neural network	0.610909	0.508475
Informatics ⇒ neural network	0.549812	0.805195
Natural Language Processing ⇒ neural network	0.592105	0.815789
gradient descent ⇒ neural network	0.62594	0.506173
random forest ⇒ neural network	0.549708	0.74359
decision tree ⇒ neural network	0.598684	0.802632
feature extraction ⇒ neural network	0.536842	0.708861
Wikipedia ⇒ neural network	0.520335	0.520335
outperforms ⇒ neural network	0.752153	0.253012
confidence ⇒ neural network	0.530702	0.831169
effectiveness | efficiency ⇒ neural network	0.558158	0.6375
scalability ⇒ neural network	0.51886	0.782051
robust ⇒ neural network	0.654255	0.341176
matrix factorization | social network ⇒ recommendation	0.569805	0.566667
recommendation | matrix factorization ⇒ social network	0.589091	0.518519
...	...	...

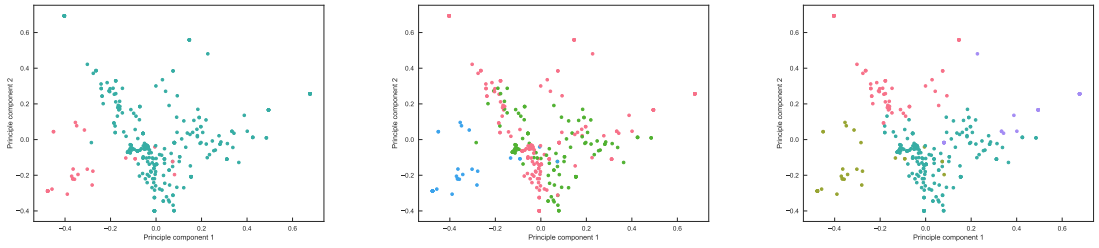
4 Task 2 - Clustering

We have learned many algorithms related to data science in classes but are they all up-to-date technique in current research field? Given all 430 paper, could we find how many types of methods have been applied so far? In order to answer this question, I want to use K-Means clustering method to group all methods in research paper into several classes. Each paper is a sample with all methods as its features.

In details, I first filtered only method values for each paper and then use Tfidf technique to vectorize each paper to construct the input `X`. The idea of using Tfidf here is similar to removal of stop words because other than numerical data, text data is usually more noisier and less distinctive. If a method appears in most of paper then it should have less weight otherwise it will make even harder to distinguish different clusters. The original `X` has shape of (331, 359). The reason why the total sample number is less than 430 is because the most of paper are partially annotated and some paper happen to not have any annotated methods at all. The total number of methods is 63; however, because Tfidf by default treats every single word as a feature while my actual features contains phrases such as "neural network" so I added `ngram_range = (1, 3)` to cover more meaningful features. The number of feature 359 is too high so I applied PCA to `X` and only used 30 principle components for clustering.

The disadvantages of K-Means are that it sometimes will converge to local minimum for a single run instead of global minimum and the number of clusters `K_clusters` will affect the results. For the first issue, the simplest way is to repeatedly run K-Means with random initialization and pick the best result which is covered in the sklearn K-Means package. For the second issue, I introduced the mean Silhouette Coefficient (a type of pair-wise distance measurement) of all samples to determine the `K_clusters`. The Silhouette Coefficient for a sample is calculated as $(b-a)/max(a,b)$ with range [-1, 1] where a is the mean intra-cluster distance and b is the mean nearest-cluster distance for each sample. The higher Silhouette Coefficient usually means that samples are better separated; however, this measurement is just the approximation of K-Means so for data as noisy as text I still need some subjective judgments.

For `K_clusters = 2`, the mean Silhouette Coefficient is 0.09; for `K_clusters = 3`, the mean Silhouette Coefficient is 0.09 and for `K_clusters = 4`, the mean Silhouette Coefficient is 0.14. Clearly, higher `K_clusters` yields higher mean Silhouette Coefficient but is higher `K_clusters` better? Fig. 1 shows the clustered samples with different `K_clusters` in 2 principle components space. The case where `K_clusters = 4` is indeed the best choice.



(a) `K_clusters = 2` (b) `K_clusters = 3` (c) `K_clusters = 4`

Figure 1: K-Means comparison with different `K_clusters`.

However, Fig. 1 doesn't have any knowledge to answer the original question as what types of methods are used because the data are actually words in Method. In order to map the real words to different clusters, I used the Word Cloud to display the words in each cluster. According to Fig. 2, I found 4 main types of methods and the overlaps between clusters indicate many paper have applied multiple methods together:

1. Gradient descent and SVM.
2. Matrix factorization.
3. Neural network related methods - this also agrees with the frequent patterns found in previous task.
4. Maximum Likelihood Estimation (MLE).

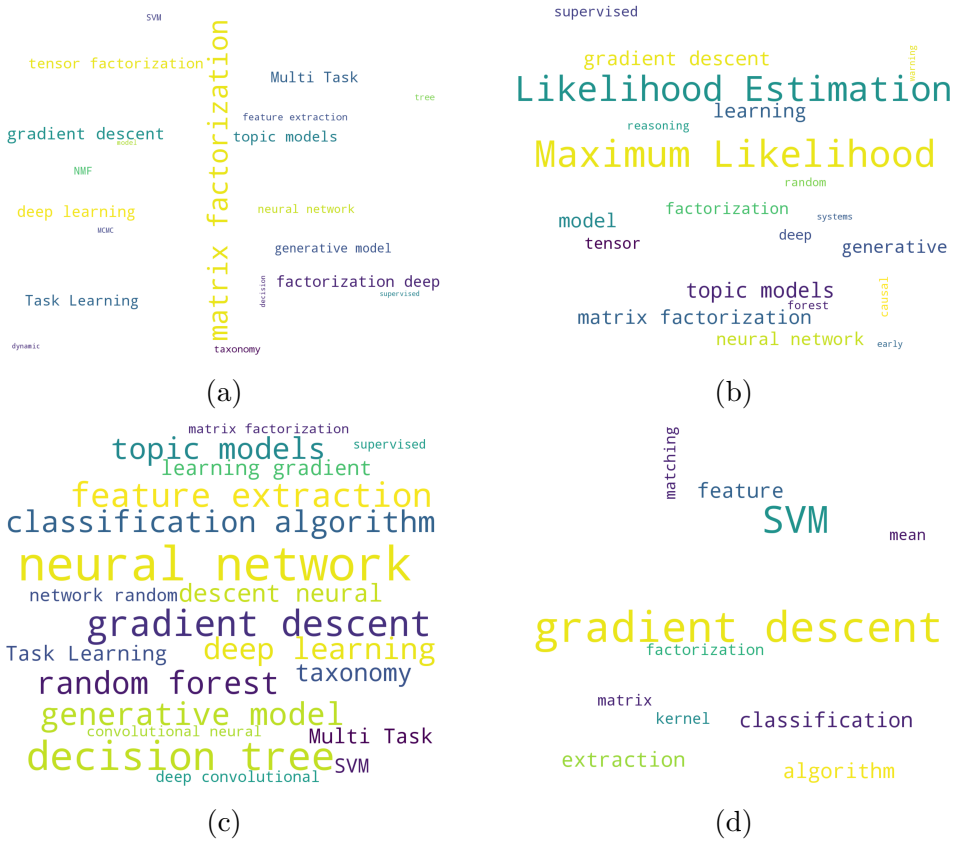


Figure 2: Display each word in cluster for `K_clusters = 4`.

5 Task 3 - Classification

In this project, I annotated 20 paper based my own knowledge to determine what dimension (i.e. Problem, Method, Dataset or Metric) a word/phrase belongs to. But one wants to ask whether we could teach the computer to annotate the paper automatically in the future so we don't need to read paper at all. To answer this question, I wanted to train the computer with my dimension values to classify whether a word/phrase is a Method. Similar

to how human beings would determine the word, one needs to look around the word in the paper content and search for signature words. For example, one should expect to find "propose", "apply" or "use" to in front of a method and "solve", "improve" or "correct" after the method.

To achieve this, I wrote a script to search the 2-nearest neighbor words for each annotated value in all 430 paper (refer to `data_process.py` for more details). Every dimension value is considered as one sample and 2-nearest neighbor words found for all dimension values are their features. For each dimension value, the neighbor word has count 1 if it appears near this value or 0 otherwise. Similar to the previous task, I again applied Tfidf with `ngram_range = (1, 3)` to vectorize the features rather than use the count directly because I realized many stop word such as "a", "of", "the" etc. also appear in the 2-nearest neighbor words.

The original input `X` has the shape (183, 687) so I still needed to apply PCA as well with 100 principle components. Then I randomly shuffled the total 183 samples and split them into training set (85% by default) and test set. To evaluate the quality of the classifier, I used F1 score which is defined below.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn}$$

Since there are many classifiers available and there is no definitely the best one. I started with simple Naive Bayes, the F1 score is bad (37%). The issue is that I initially used all 183 dimension values while the Method (label 1) vs the non-Method (label 0) is 62 vs 121, i.e. the sample is very skewed. Then I excluded all the Dataset and Metric such that Method (label 1) vs Problem (label 0) is 62 vs 60 such that `X` now has the shape of (122, 687). Then I tried Naive Bayes again, the F1 score increased to 55%. I then tried another Decision Tree but the F1 score is worse 44%. Finally, I tried the Logistic SVM with Stochastic Gradient Descent (SGD) training as it is considered as one of the popular method in text mining. The F1 score increased up to 80% which is a huge improvement.

I am not certain two main parameters: number of principle components `n_components` and percentage of training set `train_size` and they seem to affect the F1 score significantly during trials. To find out the best combinations of these two parameter to achieve the optimal F1 score, I iteratively searched `n_components` from 10 to 121 with step size 10 and `train_size` from 25% to 95% with step size 5%. The summary plot is shown in Fig. 3 and the optimal F1 score is 92% with `n_components = 30` and `train_size = 0.9`. Note that SGD training means the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing learning rate so there are small variances in F1 score for every run ranging from 88% to 92%. The result shows the model is robust to determine whether a word/phrase is a method.

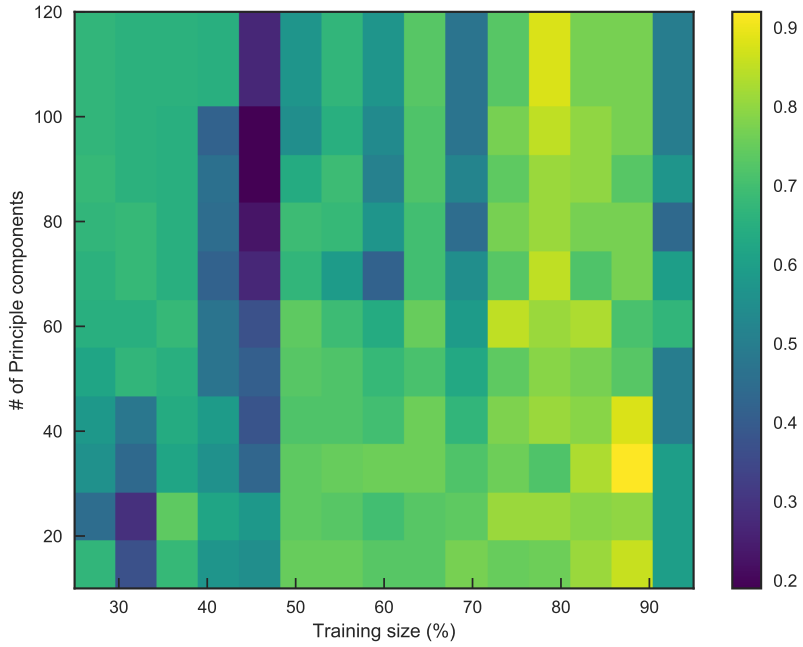


Figure 3: F1 score of Logistic SVM with SGD training.

6 Conclusion

In this project, I annotated all 430 paper to construct a data cube with 4 dimensions: Problem, Method, Dataset and Metric and the count of each cell is the number of paper that covers the content of this cell. Then I designed 3 different data mining tasks: (1) find frequent patterns and association rules in paper related to "neural network" and "deep learning" with Apriori, (2) find types of method used in 430 paper with K-Means clustering and (3) build a model that could automatically determine Method or Problem with Logistic SVM with SGD training classifier. The conclusions accordingly are:

1. Neural network and deep learning are often used to solve recommendation, social network etc., other commonly applied methods are gradient descent, random forest, decision tree etc. and Wikipedia is the widely used as dataset in such tasks.
2. There are 4 types of methods, among which neural network and deep learning indeed are the most mentioned ones together with their similar methods which agrees with the results from 1. Other three types that are employed with neural network are gradient descent, matrix factorization and Maximum Likelihood Estimation (MLE). The overlaps between methods indicate multiple methods are used in many paper.
3. In order to achieve higher accurate classification model (determined by F1 score), one needs to provide model with knowledge of both cases (symmetric rather than skewed), find the best suitable classifier and tune the parameters such as training data size and number of principle components. After these steps, I managed to increase the F1 score from 37% to 92%.