1. **Introduction (Brian/NJTPA)**
   1. **PRIME, some brief introduction of PRIME**

The NJTPA’s Planning Recommendations Information Management Engine (PRIME) interconnects planning study findings. Built upon the NJTPA Enterprise GIS, PRIME is intended to support the packaging and advancement of recommendations toward implementation. …

* 1. **Problem Description**

Data will be accumulated and deposited into the PRIME database. The PRIME system is a traditional DBMS (database management systems). Although some descriptive and/or diagnostic statistics may be summarized from the data in PRIME, given the more and more data expected to accumulate, we expect to fully exploit the data to maximize its value, by using artificial intelligence (AI-) powered predictive and prescriptive data analytics. As a pilot exploration we will build on initial population of the PRIME database and enhance the ability of users to categorize and link recommendations and needs. Several avenues may be pursued.

* 1. **Goals/aims**

Aggregate PRIME data into packages that may support future planning studies, project development and prioritization initiatives

….

1. **Data**
   1. **Data Description**

Describe the documents that have been used, and/or the documents will be added

* 1. **Data Processing**

Describe how you have done with the raw documents (e.g. the work by Eugene)

1. **Methods/Modelling**

In this section, we focus on presenting our approaches to identifying the linkages among recommendations.

* 1. **Methods**

Given the recommendations provided by NJTPA, there are two techniques we could use to explore recommendations linkages based on the availability of human annotated labels (whether two recommendations are related).

1. **Unsupervised learning**

When starting with the unlabeled data (there’s no Yes/No answer), we call this “unsupervised” learning in machine learning. We need to draw inferences and conclusions from data without any labeled annotation. One most common approach is clustering analysis, which could explore the hidden patterns and separate data into different groups.

1. **Supervised learning**

When human annotated labels are provided, we should be able to do much more exploration since we could utilize the “answer” to supervise our model training procedure. We call this “supervised” learning and this is also the most representative approach in natural language processing area. The most popular method is Support Vector Machine (SVM) which can achieve surprisingly success in numerous real-world applications.

In this project, we used both unsupervised learning and supervised learning techniques, and the details are documented in the following sections.

* 1. **Natural Language Processing Techniques**

To do unsupervised clustering, we need a measure of similarity. In natural language processing area, the most popular similarity used is cosine similarity. And to calculate the similarity, we need a bunch of numerical features. To train a supervised model, normally we need a numerical matrix with bunch of columns representing the features of the data. But what we have is a set of textual description for recommendations. So before doing any in-depth analysis, we need to leverage some natural language processing techniques to derive a numeric vector to represent each recommendation in our data.

Natural Language Toolkit (NLTK) is a free, open-sourced leading platform for building Python programs when working with natural language data. It provides all the necessary libraries for text processing, like tokenization, stemming, tagging, parsing, and much more. In this project, most of the data processing steps are done by interfaces from NLTK.

* + 1. Corpus Preprocessing

Before we do any basic transformations for data, the first and most important thing we need to do is data preprocessing. In natural language processing area, the popular preprocessing steps include conversion of lowercase letters, removal of punctuations, numbers and stop words.

However, there’s no rule-of-thumb on what to do and how to do it. It totally depends on the corpus and your goal. For example, punctuation is redundant when it comes to analyze news or blogs but it’s important for tweets with strong sentiments. In our case, numbers representing indexing or monetary values are irrelevant to our analytics. But when it represents the name of highway, numbers would become very informative to our goal.

In this project, we used personalized preprocessing steps instead of the general interfaces to remove all the punctuation and numbers.

1. The first step was normalizing the text by replacing the abbreviation and special characters. For example, replacing “Rt.” to “Route”, “lb” to “pound”, “ft” to “foot”, “%” to “percent”, “<” to “smaller than”, and so on.
2. Next, we removed the numbers representing the indexing and monetary values, but kept the highway and route number. For example, numbers in “$70”, “MP 80.23” and “1.9 acre” were removed, but numbers in “I-78”, “Route 21” and “Highway 17” were kept.
3. Meanwhile, the alphabet representing the indexing were removed together with numbers. For example: “a./b./c.”,”1)/2)3)”, and “1./2./3.”.
4. Then when we removed the punctuations, we only removed the punctuations like “”, (), [], and ?, /, : when ending with whitespace. But others like “&” or “/” were replaced with text by approaches described in first step.
5. The last step was to remove the stop words. And we used the default stop word list provided by NLTK.
   * 1. Tokenization

Given a description of recommendation, what tokenization does is to chop the text sequence into pieces of words, or more formally called “token”. After a particular recommendation is chopped into a group of tokens, it would be easier to group them together as a useful semantic unit for processing.

* + 1. Stemming

Stemming usually refers to a grammatical process that finds the common forms of derivationally related words. For example, a recommendation is going to use “improve” and “improving” while another recommendation is going to use “improves” and “improvement”. The goal of stemming is to remove morphological affixes from those words and reduce them to their common base form: “improv”.

In NLTK, there are several popular interfaces for stemming English, like Porter Stemmer, Snowball Stemmer, WordNet Lemmatizer, and so on. In this project, we used Snowball Stemmer.

* + 1. POS tagging

Part-of-Speech (POS) tagging is one of the most important text analysis tasks used to classify words into their part-of-speech classes or lexical categories. A simplified form of such categories includes identifications such as nouns, verbs, adjectives, adverbs, etc. This is important for us to understand the sentence structure and somewhat follow the meaning of the text to do further analysis. For example, the word “break” could be a noun and a verb as well, but lead to totally different meanings given specific scenarios like “step on car break” and “break the records”.

The POS tagger we used in this project is the build-in tagger provided by NLTK. It may not be the best tagger, but good enough for our analytical purpose. Other taggers are also available in NLTK, like CRF tagger, HMM tagger, and Stanford tagger.

* + 1. Ngram/Bag-Of-Words

After all the necessary processing described above, the last step would be how to build a numeric vector for a given recommendation description. And this is the Ngram or bag-of-words representation. For each recommendation, the vector to represent its context would have exactly the same size. And the size is determined by the length of the dictionary which contains all terms in the corpus after preprocessing steps described above.

There are several ways to fill up the numeric vectors, including: binary presence, count number, term frequency, TF-IDF(term frequency - inverse document frequency), and so on.

1. Binary presence: there’s only 0 or 1 in the numeric vector to represent whether the term in dictionary appears or not. 0 means the term is not in the text, 1 means the term appears in the text.
2. Count number: the co-occurrence of a term appears in the text. 0 means the term is not in the text.
3. Term Frequency: the first step would be to count the number of times a term occurs in a recommendation. Then people would calculate the frequency based on the size of text/recommendation. The intention is to get rid of the factor of text size to have a better intuition on how important a term is. For example, “bike lane” occurs 3 times in recommendation 1 while twice in recommendations 2. But recommendation 1 contains 30 tokens while recommendations 2 only have 10 tokens. The value for “bike lane” would be 0.1 and 0.2 for recommendation 1 and 2. Then we would know “bike lane” is comparably more important in recommendation 2 rather than 1.
4. TF-IDF (term frequency - inverse document frequency): The inverse document frequency is another statistics to evaluate how important a term is for a given recommendation. In fact certain terms that occur too frequently in most of the recommendations would have little power in discovering the relevant pattern we want. Also the terms that occur less in some recommendations can carry more information. The intuition of IDF is to weigh down the effects of too frequently occurring terms while weigh up the less frequently occurring terms. There’s several variants on the formula to calculate IDF. Also, there’s several variants to calculate the TF\*IDF as well. We won’t put too many mathematical details here. It’s more important to understand that the numeric value is weighted representation of each token in recommendation, and would reflect its importance compared with other recommendations.

Again, there’s no rule-of-thumb for which one should use to fill up the numeric value. It depends on the corpus, your goal, and the machine learning techniques you are going to use. For example, it’s more popular to choose binary presence or raw count number for short text analysis, like tweeter corpus. It’s more effective to choose TF-IDF for long document analysis and based on your purpose there would be other principles on how to choose the right TF-IDF formula among variants.

In our project, we tried binary presence, count number, and term frequency with L2 normalization. The experiment shows that the term frequency could achieve better performance.

* 1. **Unsupervised Learning with Topic Models**

The goal for our unsupervised learning approach is to build a baseline for link prediction, so we would have an idea how well we could make predictions if no annotation is given. The approaches we took focused on discovering the potential underlying patterns behind the context of recommendations. The machine learning technique we used is Latent Dirichlet Allocation(LDA) which is a popular topic model used to explore the topics of the data. And we use the cosine similarity as the measurement to calculate the relevance of link between recommendations.

The steps we took to build the unsupervised learning model include:

1. Preprocessing the data by approaches described in section 3.2.1.
2. Building a bag-of-words representation for recommendations after tokenization and stemming which are described in previous sections.
3. Using the LDA topic model to find the topic distribution of each recommendation
4. Calculating the cosine similarity of the pairwise link between recommendations by combining the bag-of-words representation and topic distributions.

In the end, we set a cut-off value for the similarity score. The link with similarity score above the threshold will be predicted as “YES”(have relation), otherwise “NO”(no relation).

* + 1. Topic Model/LDA

Latent Dirichlet Allocation (LDA) is a popular statistical model in natural language processing that used to explain why some parts of a set of text documents are similar by grouping them into different topics. A topic here is not strongly defined, but identified by the automatic detection of likelihood of the word co-occurrence given a set of documents. A word may occur in several topics with a different probability, however, with a different typical set of words in each topic. Since each document is a collection of words, the intuition of LDA is that the document is a mixture of a small number of topics and each word’s creation is attributable to one of the topics.

Meanwhile, LDA is an unsupervised learning model as well. The dilemma when using it is to decide the number of topics to be used. If the number of topics is quite small, there would be too many semantic overlaps between topics and the documents won’t be fully explained. On the other hand, if the number of topics is too large, it will be impractical to understand the documents. Generally, a practical way would be to try out different values of topic number, then select the one that could distinguish more topic distributions from each other.

The LDA interface we used in this project is from the SciKit-Learn, a famous machine learning platform in Python.

* + 1. Similarity Score/Cosine similarity

The similarity score we used to measure how close the link between two recommendations is cosine similarity. And we used both bag-of-words representation and topic distribution to calculate it. After we have the bag-of-words representation for each recommendation, it’s represented by a numeric vector. And the topic distribution of each recommendation is a numeric vector as well. We put them together as a single vector by concatenating them.

You could envision the image of recommendations as points in an n-dimensional space. The numeric value would be the value for each coordinate and the n is the number of coordinates which is the size of vector. If two recommendations are related to each other, they would point to a similar direction from original or relatively close in this high dimensional space. Then the cosine of the angle between two vectors representing recommendations leads to how closely they are related to each other.

* 1. **Supervised learning**

The goal for our supervised learning approach is to refine our models by utilizing the annotation from NJTPA and make better link prediction. The model we used is SVM, and we focus on designing more powerful features to uncover the hidden patterns.

The steps we took to build the supervised learning model include:

1. Preprocessing the data by approaches described in section 3.2.1.
2. Building a bag-of-words representation for recommendations after tokenization and stemming which are described in previous sections.
3. Using the LDA topic model to find the topic distribution of each recommendation.
4. Calculating the cosine similarity of the pairwise link between recommendations by combining the bag-of-words representation after expanding the text with semantic similar words and topic distribution.

Then each link would be represented by a vector contains Recommendation 1(bag-of-words, topic distribution, Action), Recommendation 2(bag-of-words, topic distribution, Action), and their similarity score.

In the end, with the observation of links as input and annotation from NJTPA as output, we used SVM to train a prediction model.

* + 1. Ground Truth

Based on the similarity score we derived from section 3.3.2, NJTPA investigated the detail of recommendations again and annotated the link as YES(have relation) or NO(no relation). We used this annotation as our ground truth to train the model and evaluate our results.

In our data, we have 664 pairs of link between recommendations. In which we have 191 “YES” annotation and 473 “No”.

* + 1. Feature

Since our model is to predict the link relation between recommendations, so our feature space should be designed to compose the information of the recommendation pairs in the linkage. In this project, it should contains the bag-of-words representation and topic distribution for each recommendation, and the action of each recommendation. At the same time, the similarity score between the recommendations are also included in the feature space.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Recommendation 1 | | | Recommendation 2 | | | Similarity | |
| Link Vector | Bag-of-Words | Topic Distribution | Action | Bag-of-Words | Topic Distribution | Action | Bag-of-Words  Similarity | Semantic  Similarity |

* + 1. Similarity Score/Cosine similarity

The way we calculate the cosine similarity in supervised approach is same as unsupervised learning but with two different similarity score: (1) Bag-of-Words Similarity: cosine similarity based on recommendation description and topic distribution; (2) Semantic Similarity: cosine similarity based on semantic expansion to the recommendation description. The bag-of-words representations used to calculate the semantic similarity score are different with the other one.

* + - 1. Semantic Expansion

During our exploration, we found that part of the link are annotated as “NO” even the recommendations share common words or similar topic distribution. The reason is that they may share common nouns or verbs, but have different neighboring verbs or nouns. For example, “expand NJ TRANSIT GO-BUS brand of rapid bus service enhancements” and “encourage NJ TRANSIT to schedule better coordinated connections” are the descriptions for two recommendations. They share the common “NJ TRANSIT” but with different neighboring verb “expand” and “encourage”. On the other hand, some link are annotated as “YES” while the recommendations don’t share common words. The reason is that they are actually talking about same issue with semantic similar but not exactly same nouns or verbs.

So expanding the short description with more semantic similar words would leverage such problems. The way we did it is for each word in a given recommendation, first we got their POS tagger as describe in 3.2.4, then added their semantic similar words to each recommendation if the word is noun or verb.

Please note that we used recommendation description and planning title to generate the bag-of-words representation. But we only did the semantic expansion on the recommendation description. Meanwhile, we only did the semantic expansion for calculating similarity score, but didn’t change the bag-of-words representation for each recommendation in feature space. The reason is that it will introduce new problems to generate the n-gram tokens after adding those semantic words.

* + - 1. Binning

Another issue that the unsupervised learning won’t have is how to add the similarity score to the feature space. The similarity score is a single numeric value, we used a cut-off value to make decision in unsupervised learning approach. But in supervised learning approach, we need a feature vector, and the bag-of-words representation would be a vector with size up to thousands. If we append the numeric value to this vector, the information carried by this value would be lost. So we use binning to flat this value out into a long vector, and then append this vector to the feature space.

1. **Results**

In this section, we report the results of our exploration for the linkage prediction, including the baseline built by unsupervised learning approach, and the best result from our supervised learning investigation.

* 1. Platform and Software

Development Platform: Windows 7

Software: Anaconda 5.0.1 + Python 3.6

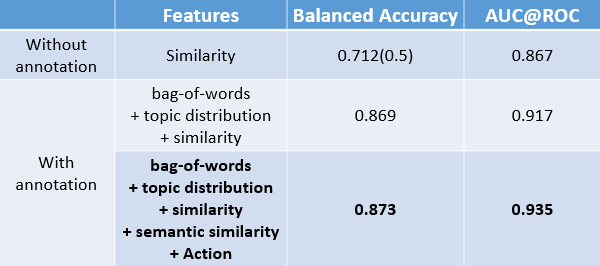
Python Packages:

scikit-learn: 0.19.1

wordcloud: 1.3.1

* 1. Performance Summary Statistics

The metrics we used to evaluate our results include: balanced accuracy and Area Under Curve (AUC) score. Those two metrics are popular used for imbalanced data analysis, since we got much more “NO” annotation than “YES” annotation.



* + 1. Balanced Accuracy
    2. AUC
  1. Some representative application examples
  2. Some representative word cloud plots

1. **Conclusions**
2. **Future Extension** 
   1. Collect more data to improve model training
   2. Integration with PRIME
   3. Reinforcement learning along with the daily usage with PRIME
   4. Bring AI to optimize other modules/functions in PRIME