Gender Prediction on Twitter Posts by Using LSI and KNN

Jianle Chen
University of Washington, Tacoma
Tacoma, Washington, the United States
Lukechen1991@hotmail.com

Tianqi Xiao
University of Washington, Tacoma
Tacoma, Washington, the United States
xiaotianqi93@outlook.com

***Abstract*— Gender prediction on twitter posts is usually regarded as a text classification problem in machine learning perspective. K-Nearest neighbor has its restriction in text classification due to some reasons. Combining what we learned in linear algebra class, we would like to apply LSI to make KNN as a better candidate for text classification.**

Keywords—Gender Prediction; Text Classification; LSI; KNN;

I. INTRODUCTION

Gender prediction is a popular topic among computer science areas. The rise of social media offers us tons of data for training and validate gender prediction model. As a deliver format of social media, posts data plays an important role in gender prediction. In a machine learning perspective, gender prediction on Twitter posts is usually regarded as a text classification problem. The most popular techniques for text classification are SVM (Support Vector Machine) and Naïve Bayes. However, KNN (K-Nearest Neighbor) has its restriction on text classification due to some reasons. For example, KNN shows a really low prediction accuracy on high dimensional feature space. Combining what we learned in linear algebra class, we would like to apply LSI to make KNN as a better candidate for text classification.

Former relevant analysis mostly focus on KNN with the help of BOW (Bag of Words) to solve text classification problems. Different from Bag of Words, LSI took advantage of SVD (Singular Value Decomposition) to reduce less informative features, which theoretically matches perfectly with KNN. Other researches such as Pang' paper [1] have compared LSI + KNN with KNN, CenKNN and mainly focusing on accuracy optimization in terms of the value of k, k as the number of neighbors. However, our project will focus on accuracy optimization based on the number of rank k in LSI. In the meantime, we show the concrete implementation details for a real text classification problem.

II. METHODOLOGY

A. Twitter Posts Dataset

Twitter Posts Dataset is a published data set for non-profit usage. It is collect from 1200 anonymous twitter users who accept their information to be published for non-profit usage. The data set can be download online. It contains two files. One file contains a "profile.csv" document. It has two columns, "userId" and "gender". Another file contains 1200 text files that are related to the "userId" in "profile.csv".

B. K-Nearest Neighbor (KNN)

K-NN is type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. [1]

The main idea of K-Nearest Neighbor is to measure the similarity of the new instances with training instances in an n dimensional space. (n equals to the number of features of instances) So when the number of selected features scales up, the search space becomes larger. Thus, N-Nearest Neighbor is sensitive to the number of features. Besides, an irrelative feature may cause two similar instances become far away from each other. So it is necessary to extract small amount of highly relative features to describe the document.

C. Singular Value Decomposition (SVD)

In linear algebra, the singular value decomposition is a factorization of a real or complex matrix. It is the generation of the eigendecompostion of a positive semidefinite normal matrix to any m*n matrix via an extension of polar decomposition. Formally, the singular value decomposition of an m*n real or complex matrix M is a factorization of the from $U\Sigma V^*$, where U is an m*m real or complex unitary matrix, Σ is a m*n rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an n*n real or complex unitary matrix. [2]

D. Latent Semantic Indexing (LSI)

Latent semantic indexing (LSI) is an indexing and retrieval method that uses a mathematical technique called singular value decomposition (SVD) to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings. A key feature of LSI is its ability to extract the conceptual content of a body of text by establishing associations between those terms that occur in similar contexts.

LSI is also an application of correspondence analysis, a multivariate statistical technique developed by Jean-Paul

Benzécri[16] in the early 1970s, to a contingency table built from word counts in documents.

Called Latent Semantic Indexing because of its ability to correlate semantically related terms that are latent in a collection of text, it was first applied to text at Bellcore in the late 1980s. The method, also called latent semantic analysis (LSA), uncovers the underlying latent semantic structure in the usage of words in a body of text and how it can be used to extract the meaning of the text in response to user queries, commonly referred to as concept searches. Queries, or concept searches, against a set of documents that have undergone LSI will return results that are conceptually similar in meaning to the search criteria even if the results don't share a specific word or words with the search criteria.

III. IMPLEMENTATION

By strictly following the modularization rules in developing code, we can easily control the input and output of each model as well as testing them. The architecture of the project contains 5 models: Dataset Model, Data Preprocessing Model, LSI Model, KNN Prediction Model and Accuracy Calculate Model.

The process and dependency show in Figure 1.

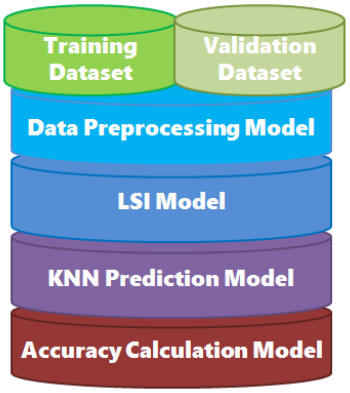

Fig. 1. Implimentation Architecure

Each model has its own functionality and plays different role in the architecture.

A. *Dataset Model*

As an initial part of the whole project, the Dataset Model reads all the data from public twitter post data files from local disk: profile file and text files. As the output of this model, all the text files are extracted through the file name, which is identical to the 'userid' field in profile file. And then these text documents are stored in a List data structure. This list will be later used by Data Preprocessing Model. The detail code shows in Figure 2.

After the unified preprocessing process, the list will be split into two parts, known as Training Dataset and Validation Dataset. It is showed in Figure 3.

```
''' Dataset Model'''
df = pandas.read_csv('D:\\training\\test\\profile\\Profile.csv')
fwrite = codecs.open('D:\\algebra function\\bigtext.txt','a')
text_line=[]

for i in range(0,1201):
    f = codecs.open('D:\\training\\text\\'+df['userid'][i]+'.txt', encoding='latin-1')
    text = f.readline()
    text_line.append(text)

print "ready to remove the stopwords and stems!"
''' Dataset Model'''
```

Fig. 2. Dataset Model code

```
training_texts = []
validation_texts = []

for i in range(0, 1201):
    if(i<1000):
        training_texts.append(texts[i])
    else:
        validation_texts.append(texts[i])
```

Fig. 3. Dataset spliting

The training dataset will have 1000 instances (83%) and the rest 200 instances (17%) will be in validation dataset. Since the original dataset is randomly sorted, we did not form the training dataset by randomly picking. In later part, the training dataset will be used to train the prediction model and the validation dataset will be used to examine the accuracy of the result model.

B. *Data Preprocessing Model*

In machine learning research, data preprocessing is critical for building accurate prediction model. On the one hand, data preprocessing transform data into required format that following model can take advantage of directly; On the other hand, most importantly, data preprocessing is when we extract potentially useful features and label for training process. A successful data preprocessing usually extract all the useful information and omit noises, and thus leading to prediction model with high accuracy.

Common data preprocessing approaches for text data include removing stop words, stemming, TF-IDF and so on. And these are exactly what we used in our project.

Step 1: Word Tokenize

Since the main features that we observe are words in documents, in order to operate and analyze them, we need to tokenize them from documents first. By using the NLTK (Nature Language Toolkit) for Python, we can easily accomplish this. We form a List as the vocabulary for later use.

Step 2: Remove stop words.

It is a common sense that in English, words like "a, the, are, is" has no emotional information and will not contribute to distinguish gender. So, it is reasonable for us to remove them before we do any analysis. The details show in Figure 4.

```
english_stopwords = stopwords.words('english')
for i in range (0,len(text_line)):
    texttemp = text_line[i]
    text_line[i] = texttemp.lower()

tknzr = TweetTokenizer()
for i in range (0,len(text_line)):
    text_line[i] = tknzr.tokenize(text_line[i])

texts_filtered_stopwords = [[word for word in document if not word in english_stopwords] for document in text_line]
```

Fig. 4. Remove stop words

Step 3: Stemming

Another common sense is that there are huge amount of English words have same word stems (base or root form), they have the same emotional information, but distinguished by their extended parts. For example, “stems”, “stemming”, “stemmed” and “stemmer” have the same base “stem”. So we would like to somehow map the related words to their same stems. This process is commonly known as stemming. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation. [4] Obviously, stemming process reduces the number of words need to be considered as features.

In our project, we use stemming process to reduce the number of words in vocabulary thanks to the NLTK. The details show in below.

```
st = LancasterStemmer()
texts_stemmed = [[st.stem(word) for word in docment] for docment in texts_filtered_stopwords ]
all_stems = sum(texts_stemmed, [])
stems_once = set(stem for stem in set(all_stems) if all_stems.count(stem) == 1)
texts = [[stem for stem in text if stem not in stems_once] for text in texts_stemmed]
```

Fig. 5. Stemming

Step 4: TF-IDF

In information retrieval, TF-IDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. [5] For gender prediction, TF-IDF somehow weights for words by giving higher score to words that contribute to distinguish male and female documents.

By applying TF-IDF to the documents List, the words in vocabulary are weighted according to their contribution of gender classification. Then we obtain a list of training documents, each document is represented by a list of words, each word is paired with a specific weight value. This complex data structure will serve as the input of LSI model. The details show in Figure 6.

```
dictionary = corpora.Dictionary(training_texts)
corpus = [dictionary.doc2bow(text) for text in training_texts]
tfidf = models.TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]
```

Fig. 6. TF-IDF

C. LSI Model

LSI Model serves as the core part of this project. It helps to reduce the number of dimension of the feature space as well as indexing the new document in that vector space. The implementation steps show as follows:

Step 1: Score term weights and construct the term-document matrix A and query matrix.

Step 2: Decompose matrix A and find the U, S and V matrices, where A = USV*, this is achieved by given function in Python package.

Step 3: Implement a Rank k Approximation by keeping the first k columns of U and V and the first k columns and rows of S.

Step 4: Find the new document vector coordinates in this reduced k-dimensional space.

Step 5: Find the new query vector coordinates in the reduced k-dimensional space, then return this result for next step to calculate similarities.

D. KNN Prediction Model

The basic idea of K-Nearest Neighbor model is selecting top K most similar instances, and then predicting the target label based on the majority or average label values among those K instances.

After all the documents (training documents as well as validation documents) are indexed in the reduced-dimensional in favor of LSI, there will be many ways to calculate the distance between two documents according to their coordinates. Here we used cosine similarity to measure the distance between the training instance and a validation instance. By sorting the similarity, we can pick top 7 (after experiment several times, 7 gives the best the predictive accuracy) most similar instances, and predict the gender of our new instance as the majority gender among them. In this way, we predict gender of each document in validation dataset. The concrete steps show as below.

```
for i in range(0,200):
    algebra_course = validation_texts[i]
    real_gender = df['gender'][1000 + i]
    algebra_bow = dictionary.doc2bow(algebra_course)
    algebra_lsi = lsi[algebra_bow]
    sims = index[algebra_lsi]
    sort_sims = sorted(enumerate(sims), key=lambda item: -item[1])

    male_vote = 0
    female_vote = 0
    for j in range (0,7):

        row = sort_sims[j][0]
        gender = df['gender'][row]
        if(gender == 0):
            male_vote += 1
        else:
            female_vote += 1
```

Fig. 7. KNN prediction

E. Accuracy Calculate Model

To examine the quality of our prediction model and for comparison with other machine learning models, we need to calculate the accuracy. The approach is simply calculating the ratio of predicting the gender correctly among 200 instances.

IV. Result

A. Dictionary formed by removing stop words and stemming

```
'gahh': 6070, u'troubl': 543, u'\u\xe6re': 2499, u'gmail': 5510, u'superm': 5240,
 u'closet': 3585, u'\xa8': 5197, u'junky': 4148, u'supery': 4517, u'bought': 168
4, u'jos': 3376, u'jot': 6299, u'jou': 1607, u'jov': 5377, u'joy': 1336, u'exag'
: 5840, u'job': 125, u'entir': 213, u'jod': 4240, u'joe': 3222, u'spoil': 3593,
u'jog': 4906, u'underwear': 6311, u'exclam': 6212, u'jok': 2416, u'jol': 2846, u
'jon': 133, u'arizon': 4315, u'tierd': 3007, u'makin': 5408, u'walk': 6, u'walt'
```

Fig. 8. Words in dictionary paired with its time of appearance

B. *Output of LSI model*

Documents are indexed by new coordinates in reduced-dimensional space.

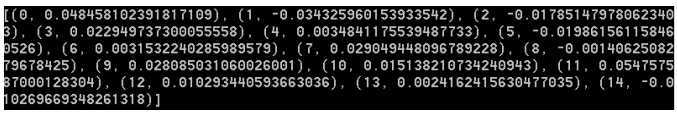

Fig. 9. New coordinates for the training documents

C. *Top 7 nearest neighbors for a new document*

Fig. 10. Indexes of Top 7 documents for a sample document

D. *Result Accuracy and Related Analysis*

When applying Rank-k approximation in LSI model, we realize the final prediction accuracy changes when k changes. This makes sense because k directly represents the dimension of the new vector space as well as the number of observed features. Thus, we need to obtain the k that contributes to the highest accuracy. The result of the experiment shows in below chart.

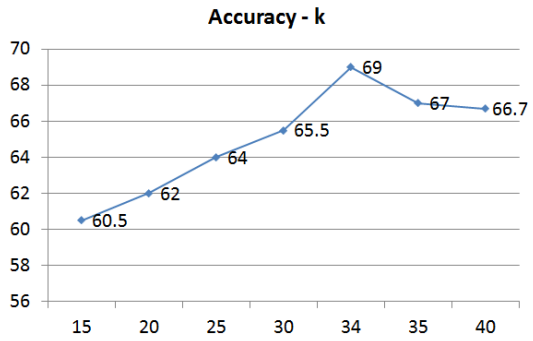

Fig. 11. Accuracy vs. Value of k

The model achieve its highest predictive accuracy when k equels to 34. And the highest accuracy is 69% based on our validation dataset. The accuracy tends to decrease when we increase or decrease the value of k.

To see how exactly our prediction model works, we would like to compare it with Naïve Bayes which works pretty well on text classification. By using "sklearn" package for python, we can easily the accuracy for Naïve Bayes is 71%, which is 2% higher than our prediction model.

Does our approach improve the predictive ability for K-Nearest Neighbor for text classification? Take one step back, we would like to compare it with the result of the project from machine learning course. See in Figure 12.

	KNN	Naïve Bayes
Facebook Posts	65%	73%
Twitter Posts	69%	71%

Fig. 12. Comparion with former project

In former project, the training dataset is Facebook posts data, when applying KNN and Bag of Words, it shows quite low performance in comparison with Naïve Bayes (almost 8 percentage). Surprisi ngly, with the twitter posts dataset, our prediction model – KNN plus LSI, though still defeated by Naïve Bayes, narrows down the gap (only 2 percentage). Although it is reasonable to say that the difference between training datasets may cause the improvement, the sudden deflation of the gaps somehow proves KNN and LSI is a better candidate for text classification than KNN with Bag of Words.

CONTRIBUTION AND DIVISION OF LABOR

Different from most former researches and Pang's research [2] which mainly focuses on the effect of the number of nearest neighbors, our project puts effort on determining rank k in order to optimize the predictive accuracy as well as showing concrete implementation details in a classic text classification scenario.

Future extension for our research may question that if we can always find the peak of performance when changing rank k. One step further, can we induct a mathematical conclusion to determine the "best" value of k instead of obtaining it by enumerating.

The division of labor for the group project.

Jianle Chen:

1. Implemented dataset model, data preprocessing model, KNN prediction model
2. Result analysis
3. Report: Wrote Introduction, part of Implementation, Result and Contribution

Tianqi Xiao:

1. Implemented LSI model and accuracy calculation model
2. Research on LSI and KNN
3. Report: Methodology, part of Implementation

REFERENCES

[1] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[2] Pang G, Jin H, Jiang S. CenKNN: a scalable and effective text classifier[J]. Data Mining and Knowledge Discovery, 2015, 29(3): 593-625.

[3] https://en.wikipedia.org/wiki/Singular_value_decomposition

[4] https://en.wikipedia.org/wiki/Latent_semantic_analysis

[5] https://en.wikipedia.org/wiki/Stemming

[6] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[7] Garcia E. Latent Semantic Indexing (LSI) A Fast Track Tutorial[J]. 2006.

[8] Wang Z, Qian X. Text categorization based on LDA and SVM[C]//Computer Science and Software Engineering, 2008 International Conference on. IEEE, 2008, 1: 674-677.

[9] Kim H, Howland P, Park H. Dimension reduction in text classification with support vector machines[C]//Journal of Machine Learning Research. 2005: 37-53.

CODES

Main.py:

```
import pandas
import nltk
import numpy
import codecs
import sys
import pickle
from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer
from nltk.stem.lancaster import LancasterStemmer
from gensim import corpora, models, similarities
from os import listdir
from os.path import isfile, join
''' Dataset Model'''
df = pandas.read_csv('D:\\training\\test\\profile\\Profile.csv')
fwrite = codecs.open('D:\\algebra function\\bigtext.txt','a')
text_line=[]
for i in range(0,1201):
    f = codecs.open('D:\\training\\text\\'+df['userid'][i]+'.txt', encoding='latin-1')
    text = f.readline()
    text_line.append(text)
print "ready to remove the stopwords and stems!"
''' Dataset Model Ends'''
'''Data Preprocessing Model'''
english_stopwords = stopwords.words("english")
for i in range (0,len(text_line)):
    texttemp = text_line[i]
    text_line[i] = texttemp.lower()
tknzr = TweetTokenizer()
for i in range (0,len(text_line)):
    text_line[i] = tknzr.tokenize(text_line[i])

texts_filtered_stopwords = [[word for word in document if not word in english_stopwords] for document in text_line]

st = LancasterStemmer()
texts_stemmed = [[st.stem(word) for word in docment] for docment in texts_filtered_stopwords ]
all_stems = sum(texts_stemmed, [])
stems_once = set(stem for stem in set(all_stems) if all_stems.count(stem) == 1)
texts = [[stem for stem in text if stem not in stems_once] for text in texts_stemmed]

training_texts = []
validation_texts = []
for i in range(0, 1201):
    if(i<1000):
        training_texts.append(texts[i])
    else:
        validation_texts.append(texts[i])
dictionary = corpora.Dictionary(training_texts)
corpus = [dictionary.doc2bow(text) for text in training_texts]
tfidf = models.TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]
'''Data Preprocessing Model Ends'''
for doc in corpus_tfidf:
    print doc
lsi = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=15)
index = similarities.MatrixSimilarity(lsi[corpus])
'''KNN prediction Model'''
right = 0.0
male_count = 0
female_count = 0
for i in range(0,200):
    algebra_course = validation_texts[i]
    real_gender = df['gender'][1000 + i]
    algebra_bow = dictionary.doc2bow(algebra_course)
    algebra_lsi = lsi[algebra_bow]
    sims = index[algebra_lsi]
    sort_sims = sorted(enumerate(sims), key=lambda item: -item[1])

    male_vote = 0
    female_vote = 0
    for j in range (0,7):
        row = sort_sims[j][0]
        gender = df['gender'][row]
        if(gender == 0):
            male_vote += 1
        else:
            female_vote += 1
    '''KNN prediction Model Ends'''
    '''Accurary Calculate Model'''
    if(male_vote > female_vote):
        male_count += 1
        if(real_gender == 0):
          right += 1
    else:
        female_count += 1
        if(real_gender == 1):
            right += 1
    '''Accurary Calculate Model Ends'''
print right / 200
```