CSCI 6405, Winter 2017

Data Mining Project

An Improved KNN Text Classification Algorithm Based on Density

Instructor: Dr. Qigang Gao

Date of Submission: April 11, 2017

Submitted By:

Kundan Kumar (B00763592)

Shalav Verma (B00756911)

DALHOUSIE UNIVERSITY

Faculty of Computer Science

Dalhousie University

Halifax, Nova Scotia

Table of Contents

Abstract 2
Chapter 1: Introduction 3
1.1 About KNN for Text Classification 4
1.2 Challenges with KNN for Text Classification 4
1.3 Application Scenario 4
1.4 Background and related work 5
Chapter 2: Data Pre-Processing and Modeling 6
2.1 Data Description 6
2.1.1 Dataset Instance 6
2.1.2 Data Description 7
2.2 Data Pre-Processing 8
2.2.1 Extracting text from HTML 8
2.2.2 Data Cleaning by Stop-Words Removal and Stemming 9
2.3 Data Modeling 9
2.3.1 Feature Selection 9
2.3.2 Vector Space Model Representation 10
Chapter 3: System Design 11
3.1 System Architecture 11
3.2 Algorithms and Methods 11
3.5 Running Demo 16
Chapter 4: Evaluation 19
4.1 Comparison of our KNN against Weka 19
4.2 Comparison of TF-IDF against TF-IDF-CI for classification 20
4.3 Evaluation of density based decision function 21
Chapter 5: Conclusion and Future Work 22

Abstract

Document classification is the task of assigning a document to one of the predefined categories. A document could be text files, news articles, books, web-pages etc. and categories could be a well-defined ideological group (like sports, politics, entertainment, finance etc.) or arbitrarily defined categories, where document in each category share high degree of similarity in content (like web pages of a website having health policy information, pages having information regarding career opportunities etc.). With an enormous number of web documents getting generated and updated daily, it has become a topic of interest and research to efficiently and accurately classify web documents into a set of pre-defined categories, defined by the varied purposes. Amongst a pool of available classification algorithms for text classification, K-Nearest Neighbor(KNN), is a widely-used algorithm. It has emerged as an effective approach for text classification for its simplicity, effectiveness and nonparametric nature. Although KNN is known to perform with good accuracy, it requires meticulous selection of features and training dataset. An uneven distribution of class data in training dataset adversely affects the performance of KNN based classifiers. This uneven distribution is quite common in the documents on the web. In our work, we explore an improved TF-IDF approach for better feature selection. Further, we verify the approach stated in [1] to improved KNN for classification request for large sets having unevenly distributed documents using a density based KNN (denoted as DBKNN). Our experiments show a significant improvement in classification performance using enhanced TF-IDF approach and a marginal improvement using density based KNN.

Chapter 1: Introduction

Document classification is a task where we are given a document and a set labels and it is required to apply the most appropriate label to the document. There are many interesting variants of basic document classification. For example, multi-class classification where a document needs to be classified into one of the more than two classes. In multi-label classification, multiple attributes can be applied to on document. A few examples of multi-class document classification are as follows:

1. Classification of a mail document as 'spam' or 'not spam'.
2. Classification of a document based on gender of the author.
3. Classification of web page document from a company's landing page into Business-to Business or Business-to-Customer category.
4. Classification of a news articles into 'sports', 'entertainment', 'finance', 'politics', 'technology' categories.

In each of the above cases, the documents which have high content similarity share the same label. To design an algorithm to automatically classify the documents into one of the pre-defined categories, the document must share similarity in their content. An algorithm to design a classifier could be based on supervised, un-supervised or semi-supervised approach. In supervised approach a training corpora having correct label for each category is required. The diagram illustrates the supervised classification paradigm.

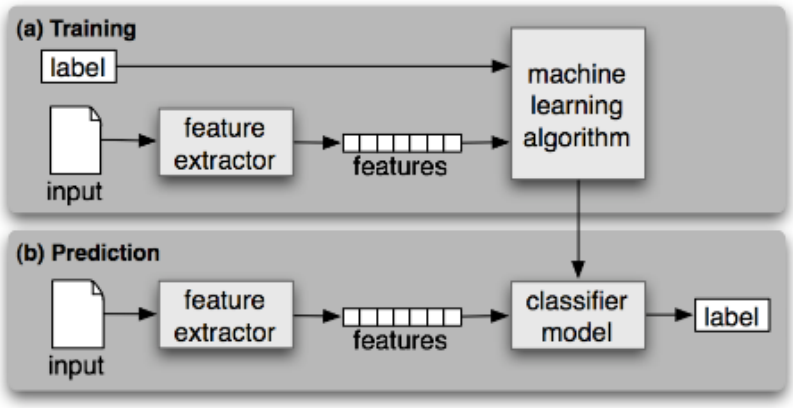

Fig 1.1: Supervised Document Classification. Source [2]

KNN is a supervised classifier and thus, requires labeled training data. In the following section, we discuss more about the KNN for text classification, its short comings and the current application scenario where recommendation from [1] and [3] are used to enhance KNN performance.

1.1 About KNN for Text Classification

In KNN for Text Classification, every document is represented as a vector of words in Vector Space Model (VSM). Every term is a document is a dimension. A word-by-document matrix is used for a collection of documents, where each entry represents the occurrence of a word in a document. Two documents in a vector space model are considered similar, it the similar kind of term appear the two text documents. Vector space model is based on linear algebra where term weights not binary. KNN also allows computing a continuous degree of similarity between queries and documents.

1.2 Challenges with KNN for Text Classification

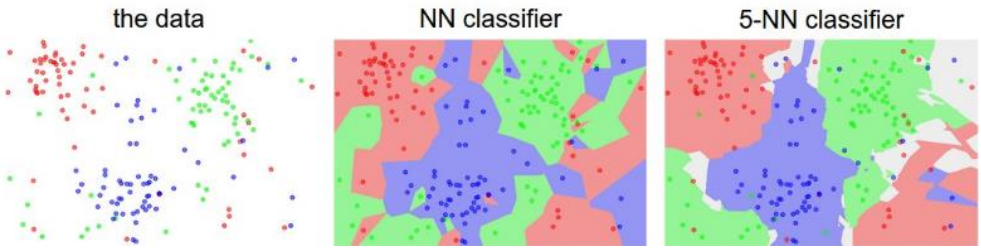

Fig 1.2: Decision boundary in KNN. Source [12]

KNN is sensitive to noisy data. The above diagram shows decision boundary formed due to noisy data. The other key challenges of KNN are as follows:

1. As KNN does not have a training phase, all computations are done at test phase. Therefore, it is computationally expensive.
2. KNN is sensitive to noise in training data which creates faulty decision boundaries.
3. Since KNN requires the selection of right value K and right distance function (Euclid Distance/Cosine Similarity etc) hyper-parameter tuning is required to achieve optimal performance.

1.3 Application Scenario

In our current exploration, we aim to verify and benchmark KNN's performance to classify web documents. We first implement our own KNN algorithm and compare it against the standard implementation (using Weka[16]). Further, we enhance our implementation with the recommendations from two research papers and verify the stated claims. The first paper [1] provides recommendation on feature selection using improved TF*IDF approach, denoting the innovative approach by TF*IDF*Ci. It accounts for class based word frequency to assign higher weights to words which can help in classification. The second paper [3] provides recommendation for a density based decision function in KNN. For our work, we will use the data from Web Mining Dataset [4] provided by University of Waterloo. This dataset is a

collection of web documents and is meant for web mining. The dataset contains 314 web pages from various web sites at University of Waterloo and some Canadian websites.

As part of conclusion of our work, we provide the following three key comparisons:

1. Performance comparison of our implemented KNN against Weka
2. Performance comparison of TF*IDF vs TF*IDF*Ci approach for features selection.
3. Performance comparison of density based vs similarity weighted majority voting decision function.

1.4 Background and related work

A lot of research has been done on methods to improve the text classification methods. However, the distribution of training set is often neglected in these studies, even though KNN classifier is quite sensitive to the distribution of samples [14]. But there are some studies where the issue has been addressed. The algorithm discussed in [13] a sample cut approach is suggested. The sample cut approach makes the density of the distribution of the training set more uniform, reducing the misclassification of the test sample in the boundary area. Also, a new distance calculation method is proposed in [14], which adjusts the distance between samples located in dense or sparse area of data distribution. Using this approach, the data tends to uniformity which reduces the influence of the uneven distribution of the data to the algorithm. In [15], the density of the training set is considered. When several samples which are in the dense area of data distribution are selected, it leads to a lesser adverse impact on the classification performance. The current paper describes an improved KNN text classification algorithm based on density, denoted as DBKNN. Its key idea is that if the density of test sample is relatively low, while the density of the nearest neighbors is much lower, then using a density based decision function we can amplify the similarity to enhance classification. Meanwhile, if the density of test sample is relatively high, while the density of the nearest neighbors is much higher, then using a density based decision function can reduce the similarity between them to improve classification. Both approaches reduce the impact of uneven data distribution to the classification.

Chapter 2: Data Pre-Processing and Modeling

2.1 Data Description

We used web mining dataset developed by University of Waterloo. It consists of 314 web pages from university websites. It has 10 categories and the documents for each category are available in separate folder in html format. Details of each category, distribution of documents in each category and description their metadata are described in following sub section.

2.1.1 Dataset Instance

The provided dataset comes in the zipped file format, which yields the following folders (signifying a category) and contains all documents of that category. Below is a snapshot of the same.

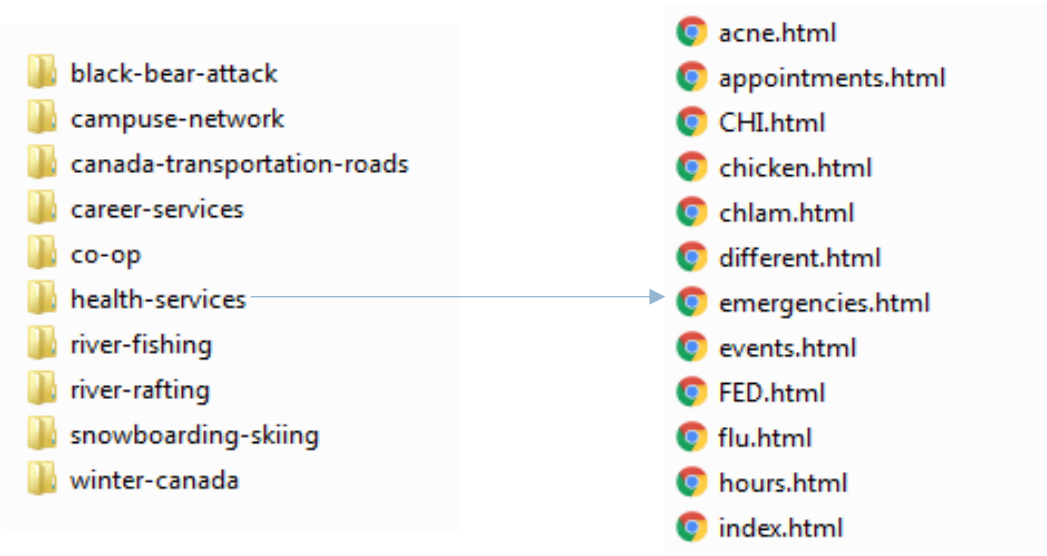

Fig: 2.1.1 Dataset as downloaded from [4]

The dataset contains 10 document categories viz, black-bear-attack, campus-network, Canada-transportation-roads, career-services, co-op, health-services, river-fishing, river-rafting, snowboarding-skiing and winter-Canada. Each folder has category specific documents in HTML format. The HTML document contains textual class data intertwined with HTML tags. These documents are representative of web documents after extraction from their source. In our work, we focus on using only the textual content displayed by the web page on the browser for classification purpose, hence a pre-processing and data clean-up step to extract text data from HTML web pages is required.

```
<p align="center"></font><em><font size="6" color="#000080"><b>Acne</b></font></em
<p><font color="#000080">Acne is a chronic and inflammatory disease that
involves the skin sites where the hair originates - hair follicles - and the
sebaceous glands. Acne may be a life-long process, and usually starts in the
preadolescence years. However, it can begin in adulthood or elderly people. <stron
affected</strong> are: face, neck, upper chest, back, and shoulders.</font></p>
<p><font color="#000080">Acne affects both sexes, males and females. At age 18
acne is more frequent in men than women. Beyond the age of 23 it is more common
in women. The severe inflammatory form of acne is more often in men at any age.
Scars rarely occurs in females, except when the lesions are manipulated by the
patient.<br wp="br1">
<br wp="br2">
<b><font size="4">How does acne originate?</font></b></font></p>
<p><font color="#000080">The increased production of the hormone androgen in
puberty, triggers a higher production of sebum by the sebaceous glands in the
skin. The sebum is a natural subtract for bacterial growth, and allows the
bacteria to increase dramatically in numbers, especially <em>Propionibacterium
acnes</em> (P.acnes). The sebum is transformed by P. acnes in both, free fatty
acids and other chemicals that produce the inflammation process.</font></p>
<p><font color="#000080">The combination of increased androgen and
pro-inflammatory chemicals causes a higher cohesion of the epidermal cells in
the pore areas forming the pimples, becoming the characteristic lesions of acne.</
<p><font color="#000080">There are closed and open pimples: a <strong>closed
pimple</strong> is a closed pore with a tiny plug. An <strong>open pimple</strong>
also known as a blackhead, is also a closed pore with a larger follicular plug.
Blackheads are not dirty. The dark colour is due to increased skin pigment and
the sebum contacting the oxygen present in the air.</font></p>
```

Fig: 2.1.2 HTML content of web page

The figure above shows a snapshot of the content of a HTML document taken from the dataset. In the following section, we will see the dataset characteristics and pre-processing steps taken before data modeling phase.

2.1.2 Data Description

The following histogram generated using Weka shows the distribution of documents in each class.

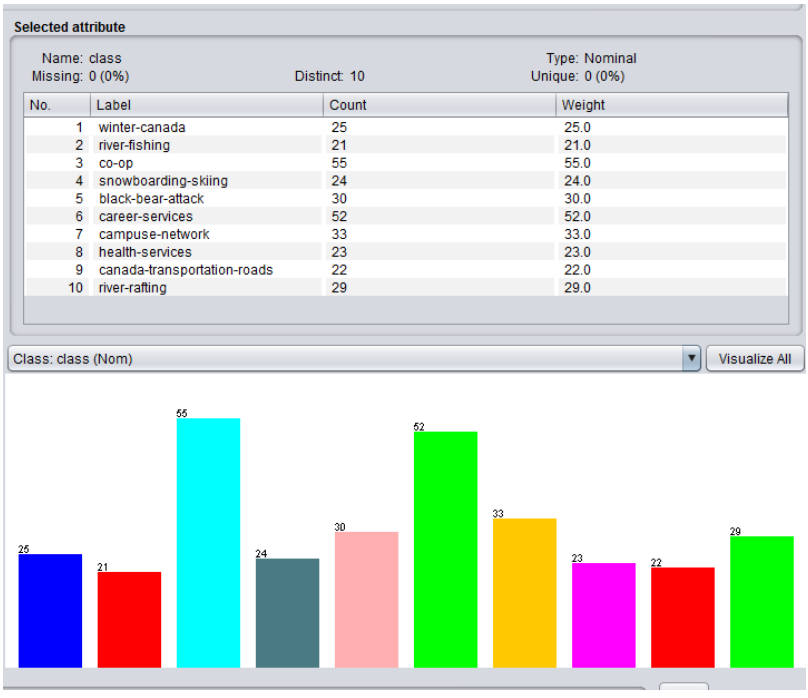

Fig: 2.1.3: Document distribution histogram generated using Weka

2.2 Data Pre-Processing

The following diagram illustrates the various steps involved in data pre-processing phase.

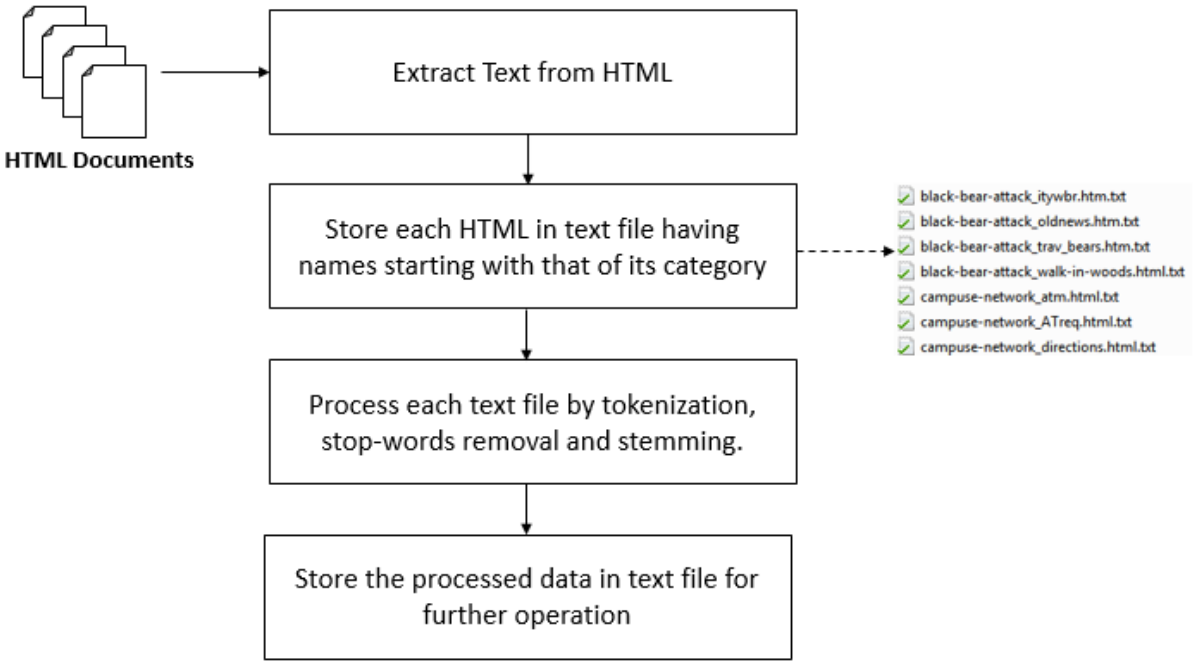

Fig 2.2.1: Data Preprocessing Steps

The keys steps of data-preprocessing were text extraction from HTML and pre-processing to achieve uniformity in data. The details of each step are explained in detail below.

2.2.1 Extracting text from HTML

Since the data was available in HTML format and had text intertwined with HTML tags, the first step required was to extract text from the HTML pages. We used Python based library called '*BeautifulSoup*' to extract text from HTML. The script used is shown below:

```
def _get_html_text_lines(self,html_path):
    with open(html_path,'r') as fh:
        html = fh.read()
    soup = BeautifulSoup(html,'lxml')
    # Remove all script and style elements
    for script in soup(["script", "style"]):
        script.extract()
    data = soup.get_text().lower()
    lines = (line.strip() for line in data.splitlines())
    chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
    text = ' '.join(chunk for chunk in chunks if chunk).splitlines()
    return text
```

Fig 2.2.2: Script to extract text from HTML

After extracting the text from HTML, the data was stored in text files with names as concatenation of their category and filename. As the next step, the data was cleaned by removing frequently occurring stop words. Stop words occur in

2.2.2 Data Cleaning by Stop-Words Removal and Stemming

Stop words are the most common words which appear in almost all documents and do not have category wise significance. Therefore, it is a common practice to remove the stop-words before creating models to be used to develop the classifier. Also, to remove ambiguities and have uniformity in data, it is desired to map singular and plural forms of nouns, conjugated forms verb to map to the same word. This is done by stemming. We removed stop-words and performed stemming using the Python NLTK package. The script used is shown below:

```
def _preprocess_text(self,text):
    words = []
    text = text.decode("ascii","ignore").encode("ascii")
    for word in word_tokenize(text):
        word = word.lower()
        if word not in stop_words:
            try:
                word = porter.stem(word)
            except:
                pass
            words.append(word)
    return words
```

Fig 2.2.3: Script to remove stop-words and stem using NLTK

2.3 Data Modeling

After the data is pre-processed to remove noise and achieve uniformity, as the next step data-modeling was performed. The detailed steps are discussed below:

2.3.1 Feature Selection

Use of TF-IDF is a common approach to identify words which hold significance for the corpus and can be used as features for classification. TF-IDF is short for term frequency-inverse document frequency. It is a numerical static that reflects how important a word is to a document [5]. The tf-idf value is directly proportional to the number of times a word occurs in document and is inversely proportional to the number of times documents in corpus where the word appears. The measure helps in identifying the words which are important to a document in a corpus. However, TF-IDF does not account for class or category base distribution of words. To handle this shortcoming, we used the approach suggested by [3] to identify most important words to be used as features. By using this approach, we identified top N features and used them to represent each document in Vector Space Model(VSM) representation. The details of VSM representation are described in next section.

Below is the snapshot of the top 10 feature along with their TF-IDF-Ci weight.

```
------------------------
  TOP 10 Features
------------------------
Word     | tf*idf*ci Score
------------------------
grizzli      0.082
resum        0.049
tb           0.040
panel        0.036
mont         0.036
subnet       0.032
snowboard    0.031
wart         0.031
steelhead    0.028
ctt          0.028
------------------------
```

Fig 2.2.4: Top 10 features identified using TF-IDF-CI algorithm

The above list of features shows the stemmed from of words. The words are sorted in the descending order of their TF-IDF-CI score. The score of these words are used to define VSM representation of the document. For our experiment, we chose top 400 words as the feature set.

2.3.2 Vector Space Model Representation

Vector Space Model is an algebraic model used to represent text documents as vectors [6]. The documents are represented as vectors as follows:

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$

Fig 2.2.5: Vector Space Model Representation of a document

In VSM representation of the documents each word represents a dimension. In the above equation, d_j represents a document from corpus of j documents. $w_{1,j}$ represents the weight for the word for j^{th} document. For our experiment, we used 400 features, hence our dimension was 400 and the weight for a word for a document was calculated using the TF-IDF-CI score of the word for the document. We used Vector space model as it allows computing a continuous degree of similarity between two documents. Once the dataset has been represented in Vector Space Model, we can compute similarity by measuring Cosine Similarity, Euclidean Distance or other distance measurement strategies defined by linear algebra principles. Vector Space Model representation useful as it allows partial matching of the documents and ranking of the documents based on their relevance.

In our work, we implemented a method to take dataset and its N features as input and generate a CSV format file having the documents represented in VSM representation.

Chapter 3: System Design

In this section, we discuss the system design, algorithms and methods used in our work. We also demonstrate the sample execution of our improved classifier.

3.1 System Architecture

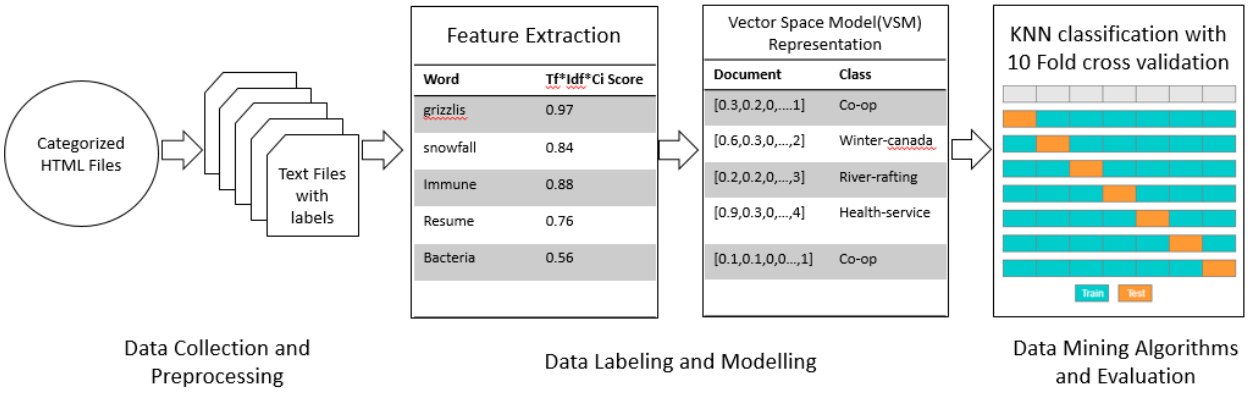

Fig 3.1 System Design

The above diagram shows the system architecture adopted. We first pre-process and clean the data. The pre-processed data is used for feature extraction. Once the top N features are available, a data model having VSM representation of the documents along with their categories is created. This model is used for running KNN and improved KNN algorithm. We measure the performance of classifier by running 10-fold validation and the average weighted F-Score for final evaluation of the classifier.

3.2 Algorithms and Methods

In this section, we discuss in detail the key algorithms and methods used in our approach

TF-IDF-Ci

TF·IDF is widely used to express the text feature weight. However, TF-IDF can't reflect the distribution of terms in the text. It can't reflect the importance degree and the difference between categories. In [3] TF-IDF-Ci is proposed. A new weight Ci is added to account for difference between classes. It defined as follows:

$$TF \cdot IDF \cdot C_i = TF \times IDF \times C_i$$
$$= TF \times \log\frac{N}{n} \times 1/(n-m+1)$$

N is the total number of texts,
n is the number of the texts containing term t,
M is the maximum number of texts containing term t in a category

It is claimed in [3] that TF-IDF-Ci not only improves the classification precision but also decreases the sensitivity towards feature dimensions to some extent. We evaluate this claim in our work. The evaluation results are described section 4 of this report.

KNN Algorithm

Below is the pseudocode for the KNN algorithm. The training phase in KNN requires only pre-processing the data and modeling it. Most of the computations happen in the test or application phase where the nearest neighbors are computed using one of the many similarity measure functions (like Euclidean distance, cosine similarity etc.).

```
TRAIN-KNN(ℂ, 𝔻)
1  𝔻′ ← PREPROCESS(𝔻)
2  k ← SELECT-K(ℂ, 𝔻′)
3  return 𝔻′, k

APPLY-KNN(ℂ, 𝔻′, k, d)
1  S_k ← COMPUTENEARESTNEIGHBORS(𝔻′, k, d)
2  for each c_j ∈ ℂ
3  do p_j ← |S_k ∩ c_j|/k
4  return arg max_j p_j
```

Fig 3.2: KNN Classification Algorithm Pseudocode [7]

Cosine Similarity Measure

Cosine similarity is the measure of similarity between two non-zero vectors, and it measures the cosine of the angle between them [8]. Cosine similarity of two vectors a and b is calculated as follows:

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|}$$

Cosine similarity measure generates a metric which quantifies how related two documents are by considering the angle instead of magnitude. The following diagram illustrates the same.

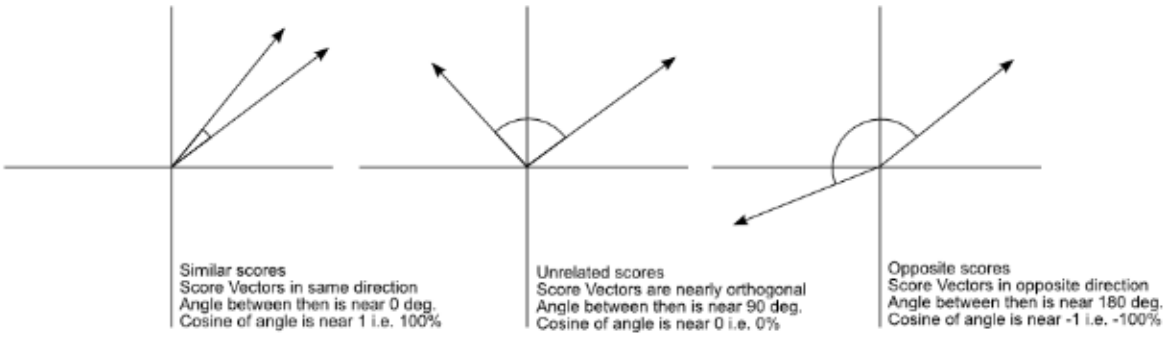

Fig 3.3: Cosine Similarity Score. Source: [9]

In our work, the KNN decision function uses cosine similarity as one of the factors to evaluate a document's similarity with other documents.

Similarity Weighted Majority Vote Decision Function

The decision function in KNN requires the cosine similarity of the document. In general approach, similarity weighted decision function is used to decide the final class of the document. The similarity weighted majority voting can be stated as follows:

$p(D,C_j)$: Similarity score of a test document D for class j

$$p(D,C_j) = \sum_{i=1}^{K} sim(D,D_i)\sigma(D_i,C_j)$$

If D_i belongs to category C_j, $\sigma(D_i,C_j) = 1$ else 0

Fig 3.4: Similarity Weighted Majority Vote Decision Function in KNN

Density based Decision Function

To handle the uneven distribution of data in dataset, [1] proposes a density based decision function defined as follows.

$$p(D,C_j) = \sum_{D_i \in KNN} sim(D,D_i)\sigma(D_i,C_j) \qquad p(D,C_j) \text{ : Similarity score of a test document D for class j}$$

$$\sigma(D_i,C_j) = \exp(m(D))/\exp(m(D_i)) \qquad m(D_i) = \sum_{D_j \in KNN} sim(D_i,D_j)/K$$

Fig 3.5: Density based decision function in KNN

The density based decision function improves the performance of KNN classifier by increasing similarity weight if test data is in has less density space but neighborhood density is lesser. Also, it reduces the similarity weight if test data is in high density space but neighborhood density is higher.

F-Score

F-Score is a measure of a classifier's accuracy. It considers precision and recall of the classifier to compute this score [10]. F-Score is calculated as follows:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Fig 3.6 F-Score Evaluation [10]

In our evaluation, we use the F-Score to benchmark the performance of KNN based classifier.

Algorithm for density based improved KNN

Below is the algorithm proposed in [11] to improve KNN performance using the density based decision function.

Step1: Do word segmentation, filter the stop words and construct the text-term matrix of TR and TE, the elements in the matrix is the number of occurrences of each entry in each text

Step2: Do dimension reduction for TR and TE, the dimension of which is $mnum$ after reduction

Step3: Calculate the feature weight for each text in TR and TE using $TF - IDF$, represent all the texts in VSM

Step4: If $TE \neq \varnothing$, turn to Step5; Otherwise, turn to Step8;

Step5: For each text D in TE, use equation (1) to calculate the similarity between D and the texts in TE, select the K texts in TE which have the largest similarity with the test sample as the K nearest neighbors of it

Step6: Calculate the weight for each category according to equation(5) and equation (6), the category which has the largest $p(D, C_j)$ is the category of the test sample

Step7: $TR = TR \cup D$, $TE = TE - D$, turn to Step4;

Step8: Calculate $F - Measure$ and output the classification results, the algorithm ends.

Fig 3.7: Algorithm for density based decision function. Source [1]

K-Fold cross validation

Cross Validation is a model validation technique. It is used to assess how the results of a statistical analysis will generalize to an independent dataset [11]. In this approach, the original sample is jumbled and partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the test data for testing the model, and the remaining k-1 subsamples are used as training data. This process is then repeated k times with each k subsamples being used exactly once as the test data.

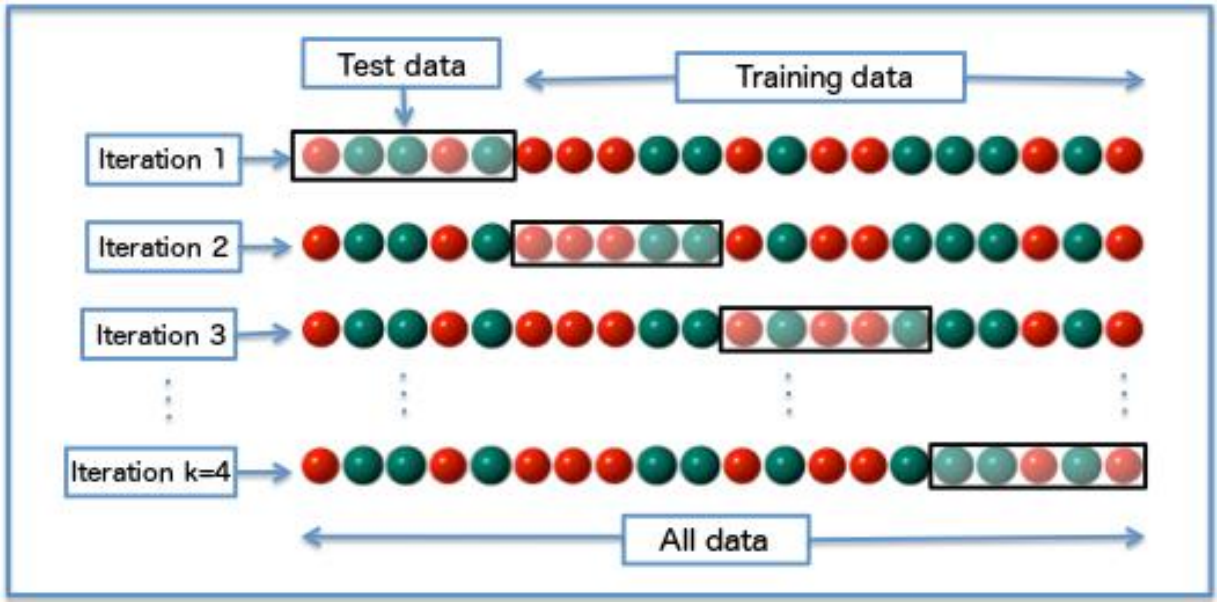

Fig 3.7: k-fold-cross validation with k=4 [11]

The K fold validation yields K results and the results are averaged to produce a single final evaluation measure. The diagram above shows the k-fold-cross validation with k=4. In our experiment, we use 10-fold validation to evaluate KNN's classification performance.

3.5 Running Demo

Running Using Weka

For comparison of our KNN against other implementation of KNN, we used Weka [16] to run and validate our results. Weka is a widely-used machine learning tool and it supports a broad range of algorithms for classification, clustering, regression etc. We used our dataset having 400 features, to run KNN classification with 10-fold cross validation. Below is the screenshot of the same.

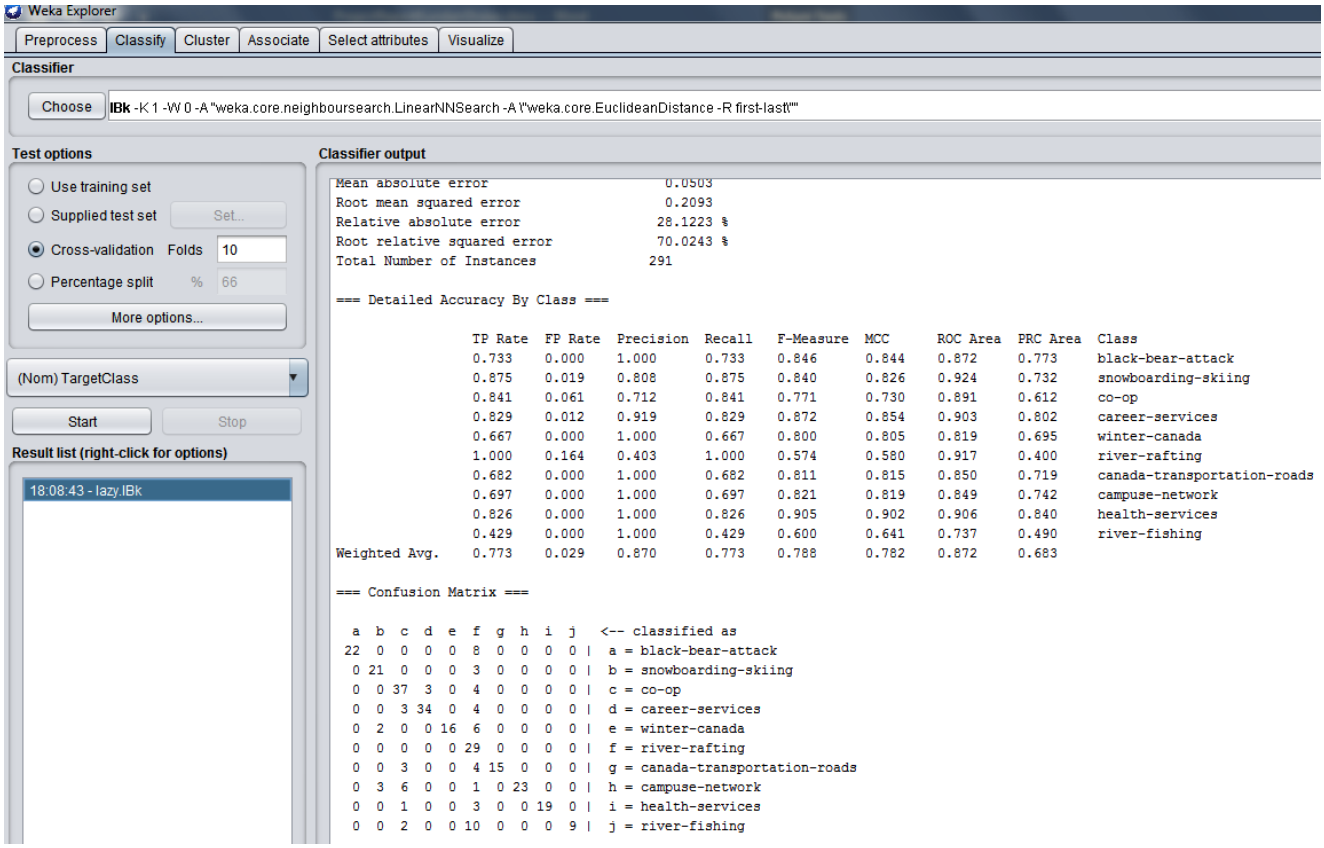

Fig 3.8: Screenshot of KNN classification using Weka

Classification Using Our Implemented Standard KNN

Below is the screenshot of the sample execution of our implemented version KNN. In following execution, the decision function used majority voting and 10 nearest neighbors were used for classification decision.

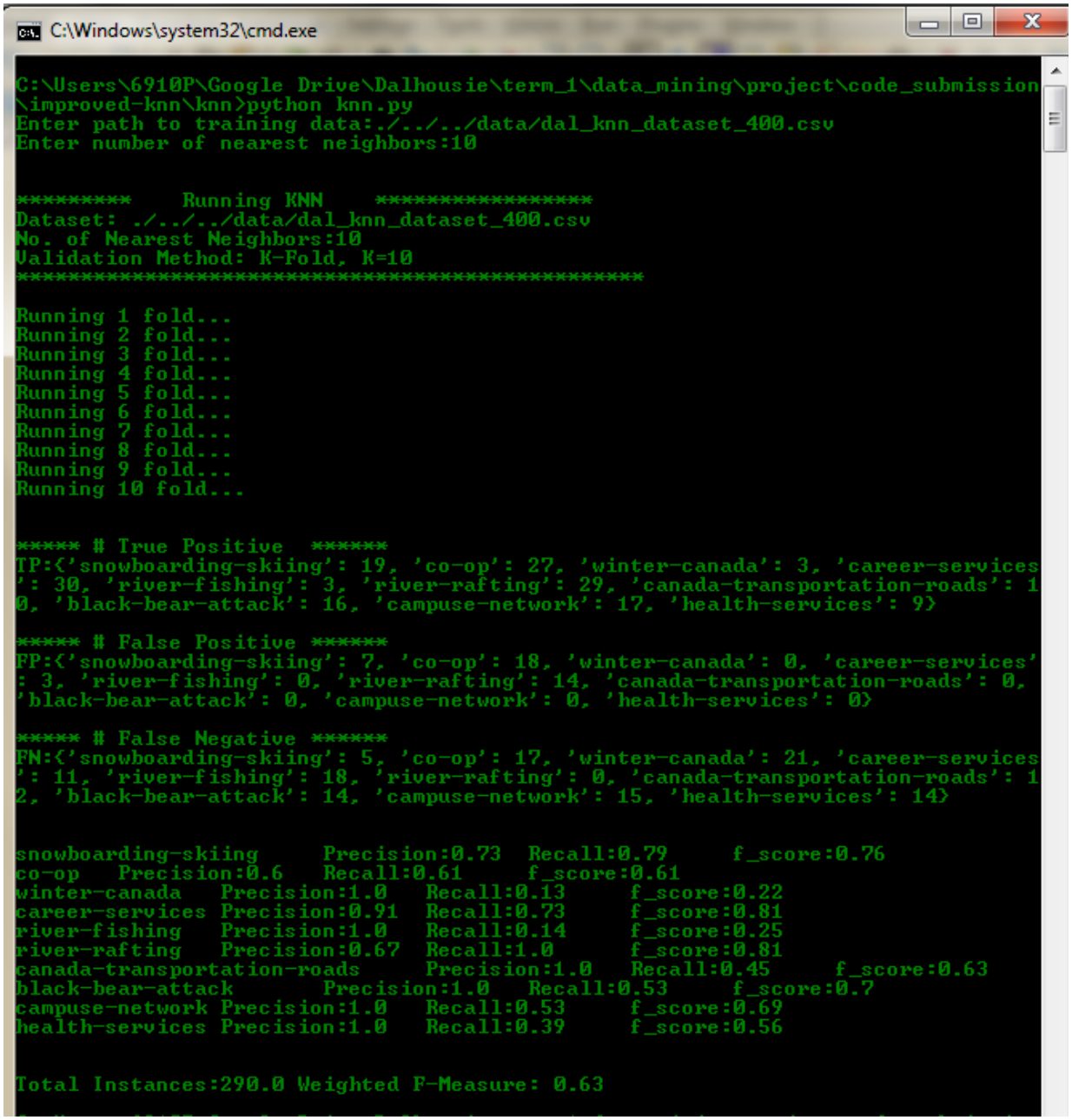

Fig 3.9: Standard KNN execution

Improved Density Based KNN

Below is the screenshot of density based improved KNN. For this execution, we used dataset with 400 features and 25 nearest neighbor were used in density based, cosine-similarity weighted decision function. K-fold cross validation was used to calculated weighted F-Measure.

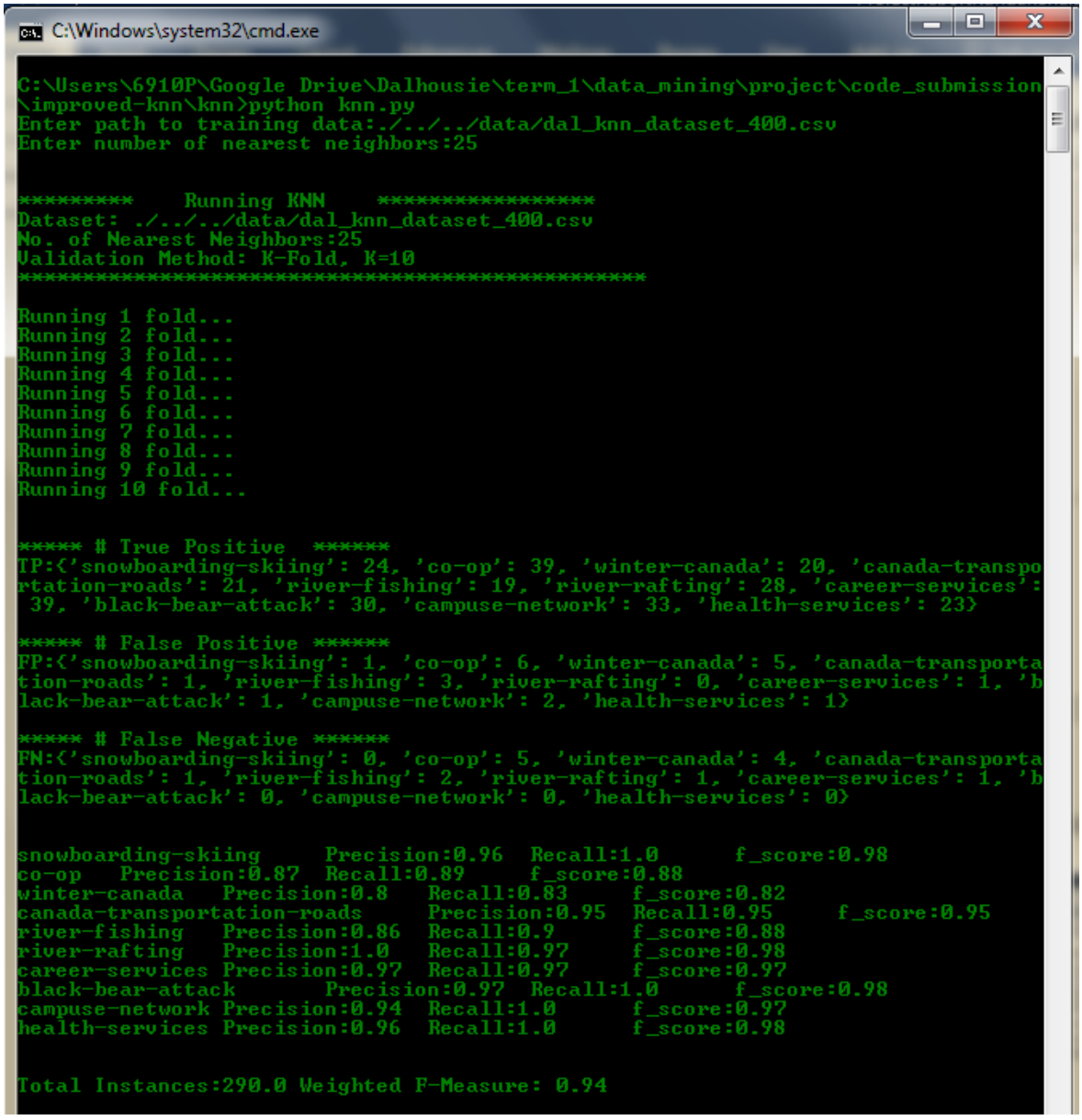

Fig 3.10: Improved Density Based KNN

Chapter 4: Evaluation

4.1 Comparison of our KNN against Weka

We implemented KNN in order enhance it with recommendations from referenced research papers. Below is the evaluation of the classification performance measured by F-Score. We compare our version of KNN against Weka. F-Measure has been calculated for dataset having varied count of features selected by filtering top N features given by their TF-IDF score.

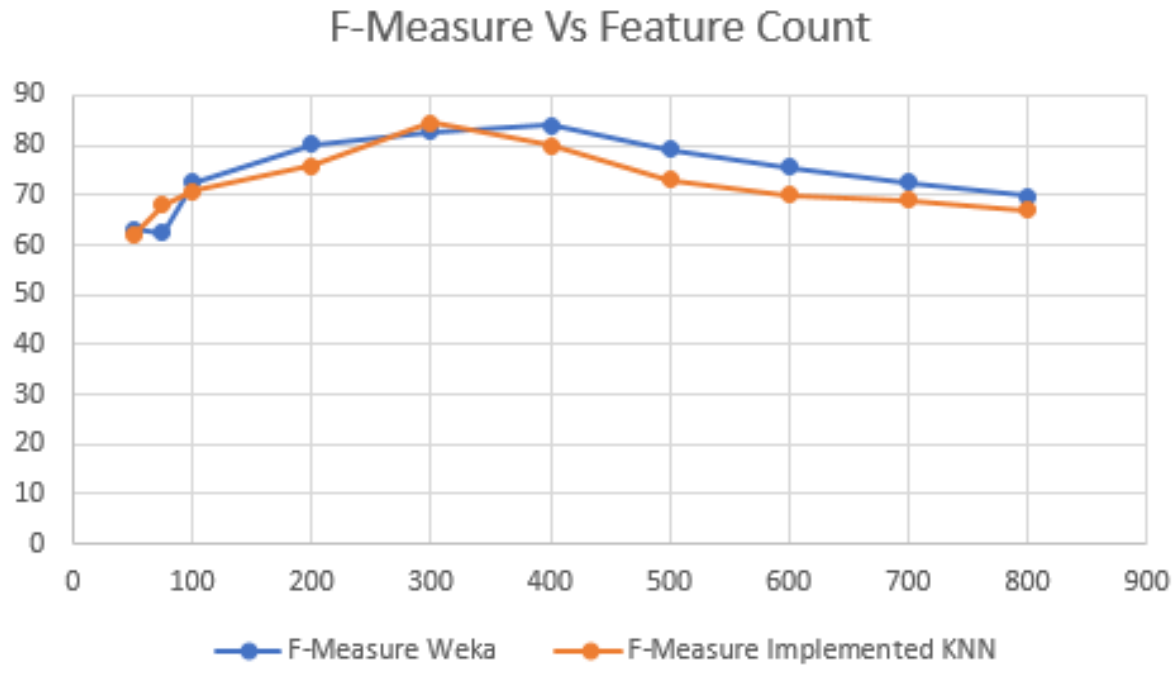

Fig 4.1: Comparison of KNN against Weka

For above experiment the value of k for KNN was set to 10. As we can see from the above plot, our implemented KNN was able to achieve performance like that offered by Weka. The best performance for classification was achieved by Weka when a dataset having 400 features was used with 10 nearest neighbor. Our implemented version of KNN achieved best performance for dataset having 300 features and using 10 nearest neighbor. As we can see, Weka performed marginally better than our implementation of KNN algorithm. We have used our version of KNN for further experimentation.

4.2 Comparison of TF-IDF against TF-IDF-CI for classification

As our next objective we evaluate, TF-IDF-CI against TF-IDF for its impact on classification performance. To test this we used two datasets, one where the features were selected by filtering top N words scored by TF-IDF scheme and the other where the features were selected by filtering top N words scored by TF-IDF-CI method. The chart below shows the classification performance measured by F-Score against dataset with different number of features.

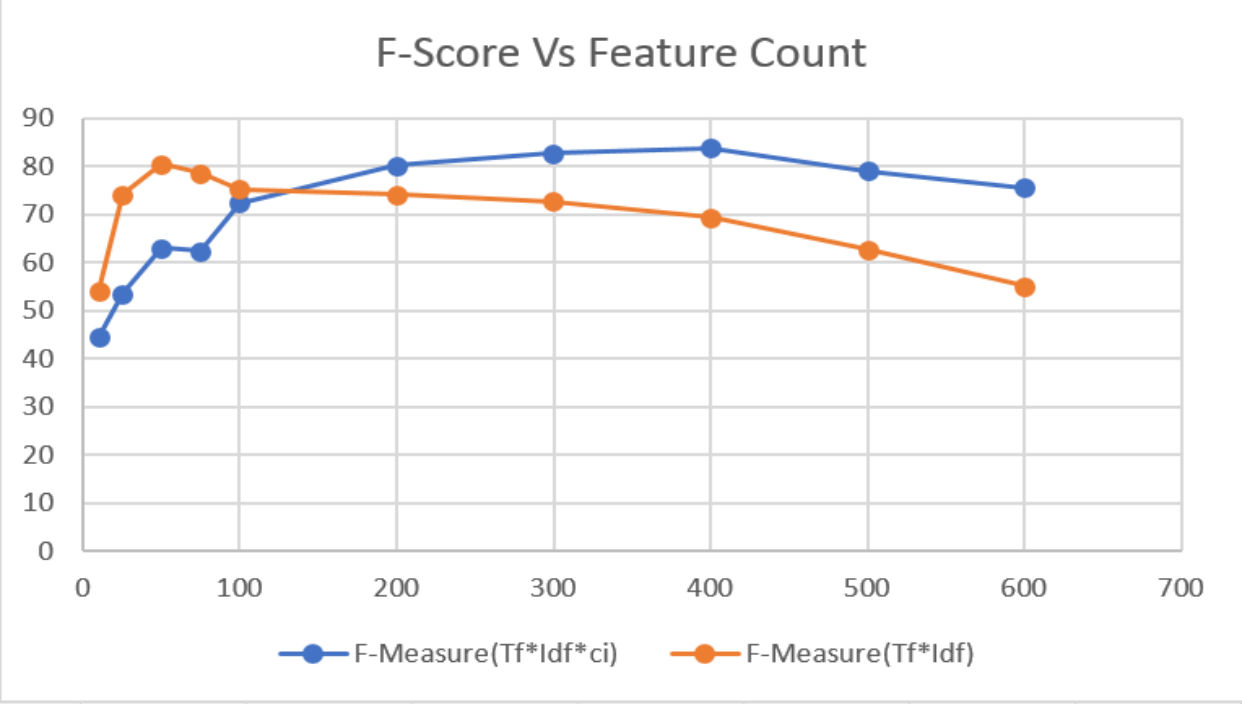

Fig 4.2: Comparison of TF-IDF against TF-IDF-CI for their impact on classification performance

As we can see from the above plot that the classification performance significantly degrades as the number of features increase if TF-IDF is used. However, if TF-IDF-CI is used, increase in features does not significantly impact the classification performance. Also, we can see that better performance result can be achieved with TF-IDF-CI than TF-IDF. This validates the claim stated in [3] regarding classification performance improvement by use of TF-IDF-CI.

4.3 Evaluation of density based decision function

Lastly, we evaluate the density based decision function in KNN for its impact on performance. Below is the plot of F-Measure vs Feature count for cosine similarity based KNN and density based KNN algorithm. 25 nearest neighbors were used for this experiment and 10-fold cross validation was used as test strategy.

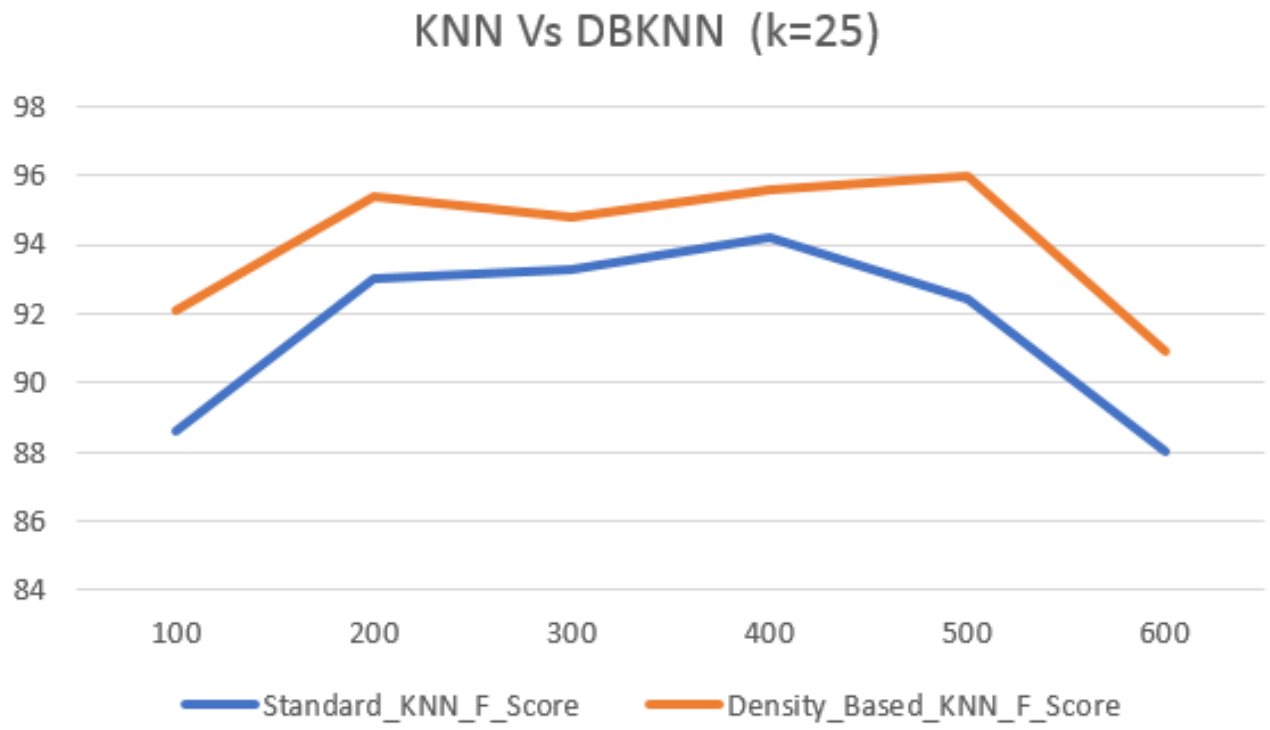

Fig 4.3: Comparison of KNN and DBKNN

As we can see from the above plot, DBKNN performed marginally better than the KNN. Using KNN we achieved a maximum F-score of 0.94 however when 400 features were used in the dataset. For the same number of features DBKNN achieved marginally higher F-Score (slightly above 0.95). DBKNN achieved a maximum F-score of 0.96 when 500 feature were used in the dataset. Based on the above plot we can conclude that DBKNN performed marginally better than KNN. This validates the claim stated in [1] regarding KNN classification performance improvement using density based KNN.

Chapter 5: Conclusion and Future Work

In this project, we used KNN algorithm to classify web documents. Standard implementation of KNN was found to perform well, by achieving approximately 0.8 F-Score. We further implemented two recommendations to improve the performance of KNN algorithm to classify web documents. The first recommendation focused on improving classification performance by improving feature selection using TF-IDF-CI algorithm to score weight of features or words. Through our implementation, we found this approach to significantly improve the classification performance. The second recommendation stated use of density based decision function in KNN to mitigate adverse effect of uneven distribution of data in training set on classification performance. In our work, we found that the density based decision function improves the performance of KNN algorithm but only by a small margin.

For future work, we can test the classifier on different datasets and benchmark its performance against large number of documents.

References

[1]"An improved KNN text classification algorithm based on density - IEEE Xplore Document",Ieeexplore.ieee.org, 2017. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6045043.[Accessed: 11- Apr- 2017].

[2]"6. Learning to Classify Text", Nltk.org, 2017. [Online]. Available: http://www.nltk.org/book/ch06.html. [Accessed: 11- Apr- 2017].

[3]"Improvement and Application of TF•IDF Method Based on Text Classification - IEEE Xplore Document",Ieeexplore.ieee.org, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/5566113/. [Accessed: 11-Apr- 2017].

[4]"Web Mining Data - UW-CAN-DATASET", Pami.uwaterloo.ca, 2017. [Online]. Available:

http://pami.uwaterloo.ca/~hammouda/webdata/. [Accessed: 07- Apr- 2017].

[5]"Tf–idf", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Tf%E2%80%93idf. [Accessed: 11- Apr- 2017].

[6]"Vector space model", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Vector_space_model. [Accessed: 11- Apr- 2017].

[7]"k nearest neighbor", Nlp.stanford.edu, 2017. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/k-nearest-neighbor-1.html. [Accessed: 11- Apr- 2017].

[8]"Cosine similarity", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Cosine_similarity. [Accessed: 11- Apr- 2017].

[9]C. Perone, "Machine Learning :: Cosine Similarity for Vector Space Models (Part III) | Terra Incognita", Blog.christianperone.com, 2017. [Online]. Available: http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii. [Accessed: 11- Apr- 2017].

[10]"F1 score", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/F1_score. [Accessed: 11- Apr- 2017].

[11]"Cross-validation (statistics)", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Cross-validation_(statistics). [Accessed: 11- Apr- 2017].

[12]"CS231n Convolutional Neural Networks for Visual Recognition", Cs231n.github.io, 2017. [Online]. Available: http://cs231n.github.io/classification/. [Accessed: 11- Apr- 2017]

[13]Chen T, Xie Y Q. Review of Feature Reduction method in Text Categorization [J]. Journal of Information, 2005,24 (6): 690-684.

[14]Salton G, Wong A, Yang C S. A vector space model for information retrieval[J].Communications of the ACM, 1975,18(11):613-620.

[15]Chen Z G, He B L, Sun Y H, Zheng X S. Research and Implementation of text classification system based on vector space model[J]. Journal of Chinese Information, 2005,19 (1): 36 - 41.

[16]"Weka 3 - Data Mining with Open Source Machine Learning Software in Java", Cs.waikato.ac.nz, 2017. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/. [Accessed: 11- Apr- 2017].

[17]J. Vanschoren, "OpenML", OpenML: exploring machine learning better, together., 2017. [Online]. Available: https://www.openml.org/a/estimation-procedures/1. [Accessed: 11- Apr- 2017].