11-2021

Information Extraction and Classification on Journal Papers

Lei Yu
University of Nebraska-Lincoln, yuleinku@huskers.unl.edu

INFORMATION EXTRACTION AND CLASSIFICATION ON JOURNAL PAPERS

by

Lei Yu

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Stephen D. Scott

Lincoln, Nebraska

November, 2021

INFORMATION EXTRACTION AND CLASSIFICATION ON JOURNAL PAPERS

Lei Yu, M.S.

University of Nebraska, 2021

Advisor: Stephen D. Scott

The importance of journals for diffusing the results of scientific research has increased considerably. In the digital era, Portable Document Format (PDF) became the established format of electronic journal articles. This structured form, combined with a regular and wide dissemination, spread scientific advancements easily and quickly. However, the rapidly increasing numbers of published scientific articles requires more time and effort on systematic literature reviews, searches and screens. The comprehension and extraction of useful information from the digital documents is also a challenging task, due to the complex structure of PDF.

To help a soil science team from the United States Department of Agriculture (USDA) build a queryable journal paper system, we used web crawler to download articles on soil science from the digital library. We applied named entity recognition and table analysis to extract useful information including authors, journal name and type, publish date, abstract, DOI, experiment location in papers and highlight the paper characteristics in a computer queryable format in the system. Text classification is applied on to identify the parts of interest to the users and save their search time. We used traditional machine learning techniques including logistic regression, support vector machine, decision tree, naive bayes, k-nearest neighbors, random forest, ensemble modeling, and neural networks in text classification and compare the advantages of these approaches in the end.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Prof. Stephen D. Scott, who guided me to the world of machine learning and gave me many suggestions and help to complete this thesis. My sincere thanks also goes to Prof. Vinodchandran Variyam and Prof. Ashok Samal, who are my committee members, Candiss O. Williams, Susan Andrews, Cynthia A. Cambardella, and Felipe Montes who discussed with me and provided many useful suggestions. Last but not the least, I deeply thank my mother, Xiuying Zhang, and my father, Junming Yu for supporting me spiritually throughout my study and completing my degree.

Contents

Chapter 1

Introduction

In an age of rapidly increasing numbers of published scientific articles, it is surprising that most systematic literature reviews and extraction of information from tables are still conducted by manually processing articles individually [1]. Systematic literature reviews aim to find and collect relevant information concerning a specific research question and are an essential step in virtually every area of research, e.g., for the preparation of review articles, project proposals, and experimental designs. While machine learning tools are available for literature searches and screens [2], they require a large number of manually evaluated articles for the training of the tool. They are often restricted to filtering articles by study design or choosing topics from a limited set of terms, and are generally limited to the evaluation of article titles and abstracts.

To extract information from journals automatically and easily, a soil science team from the United States Department of Agriculture (USDA) want to build a queryable journal paper data system, where users can easily identify journal papers of interest to them. To satisfy their requirements, the system needs information including authors, journal, publish date, abstract, DOI, journal type, experiment

location and key words in papers to highlight the paper characteristics. This information will help users to figure out if the paper is of interest to them and locate it quickly if needed. The important initial factor is data, which is journal papers in the system. We used web crawler with Python to download journal papers on soil science from the digital library to provide users with papers which is of interest to them. To extract useful information from journal papers and store them in data system as indexing and abstract, we applied name entity recognition to extract authors and location of experiments, table analysis to extract tables in the paper and store the them in a computer queryable form.

To make system recommend journal papers to users automatically, we built machine learning and deep learning models to identify users' interests. Text analysis is applied on the text data to figure out what parts of paper the user are interested in, and stored them in the database to save users' search time. During this part, I fed the text data including sections, paragraphs to many types of machine learning algorithms and used the trained models to classify unseen data in order to help user distinguish if the new pieces of text in journal paper is useful. I used traditional machine learning techniques including logistic regression, support vector machine, decision tree, naive bayes, k-nearest neighbors, random forest, ensemble modeling, and neural networks in text analysis and compare the advantages of these approaches in the end.

My contributions consist of five parts: Web harvesting, Text Classification, Table Analysis, Named Entity Recognition and Database System Building. Finally all of them can populate a relational database with information automatically extracted from journal papers collected from internet resources, and send users proper recommendations. The reason we used a web crawler to download papers is we need collect papers to build database and the papers are also the basis of

the following tasks. Text classification can help identify the section or paragraph in a paper that may be of interest to users based on their own search interest. Named Entity Recognition can extract author and experiment location from paper to store them in data system, and database system will make the future query more efficiently.

The contributions of this thesis are: 1. Web harvesting: We downloaded 38,444 papers with size of 29.53 GB from Digital Library at University of Nebraska Lincoln. 2. Text Classification: We built a machine learning-based system to identify the sections or paragraphs in the papers that may be of interest to users based on their own search interest. The model we built can catch all positives and 83% negatives. This means there is 1 paper of interest to users for every 2.89 suggested papers. 3. Table Analysis: After manually creating different kinds of tables in the journal papers, we used Seth et al.'s approach and found the program could process about 90% of all tables, which are well-formed tables. For the other 10% not well-formed tables, the program can not extract correct information. 4. Named Entity Recognition: We used Stanford's Named Entity Recognition to extract author and experiment location from paper and store them in data system which will make the future query more efficiently. The accuracy can reach about 83%. 5. Database System Building: We stored the journal paper related information including Title, Publication Date, Abstract, Journal, DOI and Type, authors, city and state extracted from papers by Stanford NER, the count of the occurrences of terms of interest to soil scientists, and infromation contained in the well-formed table converted by the algorithm of Seth et al. [3] in the Microsoft Access Database.

Chapter 2

Background

Machine learning [4] is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.

2.1 Machine Learning

No one classifier model works best across all scenarios, since every model is based on particular assumptions and has its own advantages and weaknesses. It is better to compare the performances of different models on specific dataset and choose the best one in order to reach the best performance.

We tried a lot of machine learning models including logistic regression, support vector machine, decision tree, naive bayes, k-nearest neighbors, ensemble methods

and neural networks on our dataset in order to select the best one fitting our task.

2.1.1 Logistic Regression

Logistic regression is a binary classification model that is very easy to implement and performs very well on linearly separable classes, which means the class of data can be separated by using linear model. It uses the sigmoid function, which is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve, to compress the input features to output ranges from 0 to 1, and maps the output to the class labels "0" or "1".

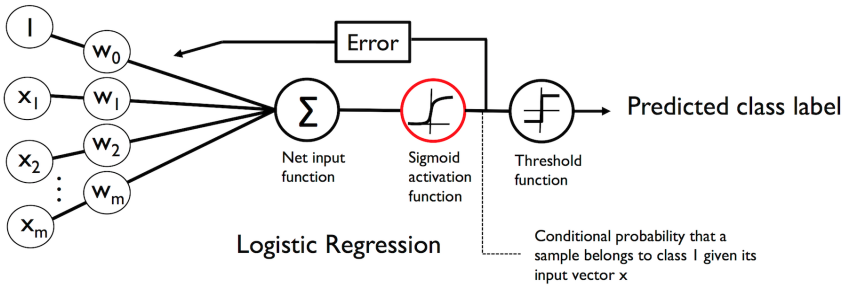


Figure 2.1: Logistic Regression Model

Let $x_0, x_1, \cdots, x_m$ in Figure 2.1 indicate sample features and $w_0, w_1, \cdots, w_m$ be weights representing the model. Z is the net input, the linear combination of weights and sample features,

$$Z = w^\top x = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m.$$

In logistic regression, the activation function is the sigmoid function.

The output of the sigmoid function is then interpreted as the probability of a particular sample belonging to class 1, $\phi(z) = p(y = 1|x; w)$, given its features x

parameterized by the weights w. The predicted probability can then simply be converted into a binary outcome via a threshold function: $\hat{y} = 1$ if $\phi(z) \geq 0.5$, and 0 otherwise. Here p stands for the probability of the positive event. The term positive event does not necessarily mean good, but refers to the event that we want to predict, for example, the probability that a patient has a certain disease, we can think of the positive event as class label $y = 1$.

The mechanism is that we define a logit function, which is simply the logarithm of the odds ratio, where the odds ratio is written as $\frac{p}{1-p}$. Formally, $logit(p) = \log \frac{p}{1-p}$, where the log refers to the natural logarithm. The logit function takes as input values in the range 0 to 1 and transforms them to values over the entire real-number range, which we can use to express a linear relationship between feature values and the log-odds:

$$logit(p(y = 1|x)) = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{m} w_i x_i = w^\top x$$

Here, $p(y = 1|x)$ is the conditional probability that a particular sample belongs to class 1 given its features x. By this way, we could get the linear relationship between sample features and probability of event given these features.

Sine we are actually interested in predicting the probability that a certain sample belongs to a particular class, we use the logistic sigmoid function, $\phi(z) = \frac{1}{1+e^{-z}}$

2.1.2 Support Vector Machine

SVM is a classification method that tries to find the hyperplane which separates classes with highest margin. The margin is defined as the minimum distance from

sample points to the hyperplane. The sample point(s) that form the margin are called support vectors and define the SVM classifier.

2.1.2.1 Linear SVM

The rationale behind having decision boundaries with large margins is that they tend to have a lower generalization error whereas models with small margins are more prone to overfitting.

Suppose the models have positive and negative hyperplanes that are parallel to the decision boundary, which can be expressed as follows:

$$w_0 + w^T x_{positive} = 1$$

$$w_0 + w^T x_{negative} = -1$$

If we subtract these two equations from each other, we could get:

$$w^T (x_{positive} - x_{negative}) = 2$$

Then we can normalize this equation by the length of the vector w, which is:

$$\|w\| = \sqrt{\sum_{j=1}^{m} w_j^2}$$

So we could arrive at the following equation:

$$\frac{w^T (x_{positive} - x_{negative})}{\|w\|} = \frac{2}{\|w\|} \tag{2.1}$$

The left side of the equation 2.1 can then be interpreted as the distance between the positive and negative hyperplanes, which is the so-called margin that we want

to maximize. By this transform, the objective function of the SVM becomes the maximization of this margin by maximizing $\frac{2}{\|w\|}$ under the constraint that the samples are classified correctly, which can be written as:

$$w_0 + w^T x_{(i)} \geq 1 \text{ if } y^{(i)} = 1 \tag{2.2}$$

$$w_0 + w^T x_{(i)} \leq -1 \text{ if } y^{(i)} = -1 \text{ for } i = 1...N, \tag{2.3}$$

where N is the number of samples in our dataset.

These two equations say that all negative samples should fall on one side of the negative hyperplane, whereas all the positive samples should fall behind the positive hyperplane. In practice, it is easier to minimize the reciprocal term $\frac{1}{2}\|w\|$.

Another concept is called soft-margin classification, which uses a slack variable ξ. The motivation for introducing the slack variable ξ is that the linear constraints need to be relaxed for nonlinearly separable data to allow the convergence of the optimization in the presence of misclassification under appropriate cost penalization. After adding the positive value's slack variable, equations 2.2 and 2.3 become:

$$w_0 + w^T x_{(i)} \geq 1 - \xi^{(i)} \text{ if } y^{(i)} = 1$$

$$w_0 + w^T x_{(i)} \leq -1 + \xi^{(i)} \text{ if } y^{(i)} = -1 \text{ for } i = 1...N$$

Here N is still the number of samples in our dataset. So the new objective function to be minimized becomes:

$$\frac{1}{2}\|w\|^2 + C(\sum_i \xi^{(i)})$$

We could control the penalty for misclassification via the variable C. A large value of C corresponds to a large error penalty, whereas a small value of C indicates less strictness about misclassification error. We can use the C parameter to control the width of the margin and therefore tune the bias-variance trade-off. This is the same as the regularization in logistic regression algorithm, and decreasing the value of C increases the bias and lowers the variance of the model.

2.1.2.2 Kernel SVM

The term ***kernel*** describes a function that calculates the dot product of the images of the samples x under the kernel function ϕ. Roughly speaking, a kernel can be understood as a similarity measure in a higher-dimensional space.

Kernel methods are algorithms that map the sample vectors of a dataset onto a higher-dimensional feature space via a kernel function $\phi(x)$. The goal is to identify and simplify general relationships between data, which is especially useful for linearly non-separable datasets.

An SVM can be easily kernelized to solve nonlinear classification problems. The basic idea behind kernel methods to deal with linearly inseparable data is to create nonlinear combinations of the original features to project them onto a higher-dimensional space via a mapping function ϕ where it becomes linearly separable.

To solve a nonlinear problem using an SVM, we could transform the training data onto a higher-dimensional feature space via a mapping function ϕ and train a linear SVM model to classify the data in this new feature space. Then we can use the same mapping function ϕ to transform new, unseen data to classify it using the linear SVM model.

However, one problem with this mapping approach is that the construction of

the new features is computationally very expensive, especially if we are dealing with high-dimensional data. This is where the so-called kernel trick comes into play. In practice all we need is to replace the dot product $x^{(i)T}x^{(j)}$ by $\phi(x^{(i)})^T\phi(x^{(j)})$. In order to save the expensive step of calculating this dot product between two points explicitly, we define a so-called kernel function:

$$\kappa(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T\phi(x^{(j)})$$

The kernel we used in the project is the Radial Basis Function (RBF) kernel or simply called the Gaussian kernel:

$$\kappa(x^{(i)}, x^{(j)}) = \exp(-\frac{\| x^{(i)} - x^{(j)} \|^2}{2\delta^2})$$

The term kernel can be interpreted as a similarity function between a pair of samples. The minus sign inverts the distance measure into a similarity score. Due to the exponential term, the resulting similarity score will fall into a range between 0 and 1, where 0 indicates very dissimilar samples, and 1 indicates exactly similar samples.

2.1.3 Decision Tree

Decision tree classifiers are attractive models if we care about interpretability. As the name decision tree suggests, we can think of this model as breaking down our data by making decision based on asking a series of questions. Using the decision algorithm, we start at the tree root and split the data on the feature that results in the largest Information Gain. In an iterative process, we can then repeat this splitting procedure at each child node until the leaves are pure, which means the samples at each node all belong to the same class. We have to be careful that the

deeper the decision tree, the more complex the decision boundaries, which can result in ***overfitting***. Overfitting is a modeling error which occurs when a function is too closely fit to a limited set of data points. Overfitting the model generally takes the form of making an overly complex model to explain idiosyncrasies in the data under study.

In order to prevent overfitting, we typically want to prune the tree by setting a limit for the maximal depth of the tree.

The objective function we use to split the nodes at the most informative features and maximize the information gain at each split is written as:

IG(D_p, f) = I(D_p) - $\sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$, where

f: feature to perform the split

D_p: dataset of the parent node

D_j: dataset of the jth child node

N_p: total number of samples at the parent node

N_j: number of samples in the jth child node

I: impurity measure

The information gain is the difference between the impurity of the parent node and the sum of the child node impurities. The lower the impurity of the child nodes, the larger the information gain.

Algorithm 1: Decision Tree

1. Start at the root node as parent node
2. Split the parent node at the feature w to minimize the sum of the child node impurities (maximize information gain)
3. Assign training samples to new child nodes
4. Stop if leave nodes are pure or early stopping criteria is satisfied, else repeat steps 1 and 2 for each new child node

Three commonly used impurity measures are Entropy, Gini impurity, and the classification error. In this project, we used Entropy as the impurity measure.

The entropy is defined as

$$I_H(t) = -\sum_{i=1}^{C} p(i \mid t) \log_2 p(i \mid t)$$

For all non-empty classes $p(i \mid t \neq 0)$, $p(i \mid t)$ is the probability of the samples that belong to class i for a particular node t; C is the number of unique class labels. The entropy is therefore zero if all samples at a node belong to the same class, and the entropy is maximal if we have an uniform class distribution.

2.1.4 Naive Bayes

Naive Bayes classifiers, a family of classifiers that are based on the Bayes' probability theorem, are known for creating simple yet well performing models, especially in the fields of document classification and disease prediction.

Naive Bayes classifiers are linear classifiers that are known for being simple yet very efficient. The probabilistic model of naive Bayes classifiers is based on Bayes' theorem, and the adjective naive comes from the assumption that the features in a dataset are mutually independent. In practice, the independence assumption is often violated, but naive Bayes classifiers still tend to perform very well under this unrealistic assumption [5]. Especially for small sample sizes, naive Bayes classifiers can outperform the more powerful alternatives [6].

Naive Bayes assumes that all attributes are conditionally independent given the

label, thereby, computing the likelihood is simplified to the product of the conditional probabilities of observing individual attributes given a particular class label. The abbreviation ”iid” stands for ”independent and identically distributed” and describes random variables that are independent from one another and are drawn from a similar probability distribution. Independence means that the probability of one observation does not affect the probability of another variable. It uses Bayes theorem to predict the probability that a given feature set belongs to a particular label. The formula is:

$$P(label \mid features) = P(label) \cdot \frac{P(features \mid label)}{P(features)}$$

The following list describes the various parameters from the previous formula:

$P(label)$: This is the prior probability of the label occurring, which is the likelihood that a random feature set will have the label. This is based on the number of training instances with the label compared to the total number of training instances. For example, if 60/100 training instances have the label, the prior probability of the label is 60%.

$P(features \mid label)$: This is the prior probability of a given feature set being classified as that label. This is based on which features have occurred with each label in the training data.

$P(features)$: This is the prior probability of a given feature set occurring. This is the likelihood of a random feature set being the same as the given feature set, and is based on the observed feature sets in the training data. For example, if the

given feature set occurs twice in 100 training instances, the prior probability is 2%.

$P(label \mid features)$: This tells us the probability that the given features should have that label. If this value is high, then we can be reasonably confident that the label is correct for the given features.

2.1.5 K-Nearest Neighbor

K-nearest neighbors algorithms find the k points that are closest to a point of interest based on their attributes using a certain distance measure like Euclidean distance. KNN does not learn a discriminative function from the training data, but memorizes the training dataset instead.

The KNN algorithm is fairly straightforward and can be summarized by the following steps:

1. Choose the number of k and a distance metric.
2. Find the k nearest neighbors of the sample that we want to classify.
3. Assign the class label by majority vote.

Based on the chosen distance metric, the KNN algorithm finds the k samples in the training dataset that are closest to the point that we want to classify. The class label of the new data point is then determined by a majority vote among its k nearest neighbors.

Advantage of KNN is that the classifier immediately adapts as we collect new training data. The disadvantages are as follows: First, the computational complex-

ity for classifying new samples grows linearly with the number of samples in the training dataset, especially for the dataset in high dimension. Second, we can not discard training samples since no training step is involved. So the storage space would become a challenge in the face of large datasets.

The good choice of value k is crucial to find a good balance between overfitting and underfitting. We also have to make sure that we choose a distance metric that is appropriate for the features in the dataset. For example, if we are using a Euclidean distance measure, it is important to standardize the data so that each feature contributes equally to the distance. In our project, we used the Minkowski distance, which is a generalization of the Euclidean and Manhattan distance and can be written as follows:

$$d(x^{(i)}, x^{(j)}) = \sqrt[p]{\sum_k |x_k^{(i)} - x_k^{(j)}|^p}.$$

It becomes the Euclidean distance if we set the parameter p=2 or the Manhattan distance at p=1.

2.1.6 Random Forest

Random forest is an ensemble classifier where multiple decision tree classifiers are combined via the bagging technique. In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Bagging (Bootstrap Aggregation) is used to reduce the variance of a decision tree. Suppose a set D of $\boldsymbol{d}$ tuples, at each iteration $\boldsymbol{i}$, a training set D_i of $\boldsymbol{d}$ tuples is sampled with replacement from D (i.e., bootstrap). Then a classifier model M_i is

learned for each training set $D < i$. Each classifier M_i returns its class prediction. Unseen/test objects are then classified by taking the majority of votes from individual decision trees.

A Random Forest can be considered an ensemble of decision trees. The idea behind a random forest is to average multiple deep decision trees that individually suffer from high variance, to build a more robust model that has a better generalization performance and is less susceptible to overfitting.

The random forest algorithm can be summarized in the following steps:

1. Draw a random bootstrap sample of size n by randomly choosing $\boldsymbol{n}$ samples from the training dataset with replacement.

2. Build a decision tree from the bootstrap sample. At each node:

a. Randomly select $\boldsymbol{d}$ features without replacement.

b. Split the node using the feature that provides the best split according to the objective function, for instance, maximizing the information gain.

3. Repeat $\boldsymbol{k}$ times the step 1 and Step 2.

4. Aggregate the prediction by each tree to assign the class label by majority vote.

Advantages to the random forest approach include:

1. We do not need to prune the random forest since the ensemble method is quite robust to noise from the individual decision tree.

2. We do not need to worry so much about choosing good hyperparameters. The only parameter that we need to care about is the number of trees in the random forest. Typically, the larger the number of trees, the better the performance of the

random forest classifier at the expense of an increased computational cost.

Other hyperparameters of the random forest classifier that can be optimized are:

1. The size n of the bootstrap sample.
2. The number of features d that is randomly chosen for each split.

The size n of the bootstrap sample is used to control the bias-variance tradeoff of the random forest. Decreasing the size of the bootstrap sample increases the diversity among the individual trees, since the probability that a particular training sample is included in the bootstrap sample is lower. Thus, shrinking the size of the bootstrap samples may increase the randomness of the random forest, and it can help to reduce the effect of overfitting. However, smaller bootstrap samples typically result in a lower overall performance of the random forest, a small gap between training and testing performance, but a low test performance overall. Conversely, increasing the size of the bootstrap sample may increase the degree of overfitting. Because the bootstrap samples, and consequently the individual decision trees, become more similar to each other, they learn to fit the original training dataset more closely.

Usually, the size of the bootstrap sample is chosen to be equal to the number of samples in the original training set, which usually provides a good bias-variance tradeoff. For the number of features d at each split, a reasonable value is $d = \sqrt{m}$, where m is the number of features in the training dataset.

In out project, we trained a random forest from 25 decision trees via the n_estimators parameter and used the entropy criterion as an impurity measure to split the nodes.

2.2 Ensemble Methods

Ensemble methods combine multiple classifiers which may differ in algorithms, input features, or input samples. Statistical analyses showed that ensemble methods yield better classification performances and are also less prone to overfitting [7]. Different methods, e.g., bagging or boosting, are used to construct the final classification decision based on weighted votes.

Ensemble methods combine different classifiers into a meta-classifier that has better generalization performance than each individual classifier alone. We will implement three approaches for creating an ensemble of classifiers in this project, including bagging and boosting. As shown in the Figure 2.2, the typical ensembling techniques are bagging and boosting. Random forest is a kind of bagging and the advantage of it is to handle overfitting and reduce variance by using independent classifiers. Gradient boosting is a kind of boosting and it can reduce bias and variance by using sequential classifiers, but the disadvantage of it is easily to trigger overfitting.

2.2.1 Majority Voting

The Majority Voting Principle simply means that we select the class label that has been predicted by the majority of classifiers, that is, received more than 50 percent of the votes.

Our goal is to build a stronger meta-classifier that balances out the individual classifiers' weakness on a particular dataset.

In majority voting, we use the training dataset to train n different weak classifiers $C_1, \ldots, C_n$. The ensemble can be built from different classification algorithms, for example, linear regression, logistic regression, decision tree, support vector

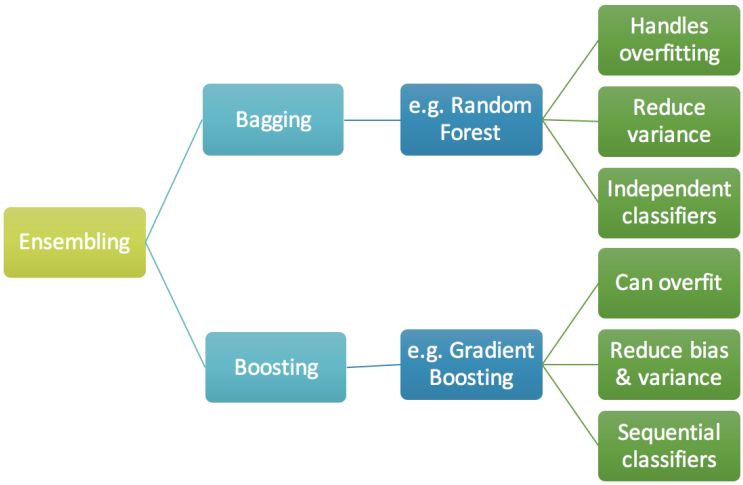


Figure 2.2: Ensemble Methods

machine, and so on. Alternatively, we could use the same base classification algorithm to fit different subsets of the training data. The latter method is also called bagging, and it is the second ensemble method we would implement.

The majority vote approach we implemented in this section is not to be confused with stacking. The stacking algorithm can be understood as a two-layer ensemble, where the first layer consists of individual classifiers that feed their predictions to the second level, where another classifier (typically logistic regression) is fit to the level-1 classifier predictions to make the final predictions. The stacking algorithm has been described in more detail by David H. Wolpert in Stacked generalization [8].

Our goal is to build a stronger meta-classifier that balances out the individual classifiers' weaknesses on a particular dataset. In more precise mathematical terms, we can write the weighted majority vote as follows:

$$\hat{y} = arg\max_i \sum_{j=1}^{m} \omega_j \chi_A(C_j(x) = i)$$

Here, ω_j is a weight associated with a base classifier, C_j, $\hat{y}$ is the predicted class label of the ensemble, χ_A is the characteristic function $[C_j(x) = i \in A]$, and $\boldsymbol{A}$ is the set of unique class labels. For equal weights, we can simplify this equation and write it as follows:

$$\hat{y} = mode\{C_1(x), C_2(x), \ldots, C_m(x)\}.$$

To better understand the concept of weighting, we will now take a look at a more concrete example. Let us assume that we have an ensemble of three base classifiers, C_j, where $j \in \{0, 1\}$, and want to predict the class label of a given sample instance, $\boldsymbol{x}$. Two out of three base classifiers predict the class label 0, and one, C_3, predicts that the sample belongs to class 1. If we weight the predictions of each base classifier equally, the majority vote would predict that the sample belongs to class 0:

$$C_1(x) : 0, C_2(x) : 0, C_3(x) : 1$$

$$\hat{y} = mode\{0, 0, 1\} = 0$$

Now, let us assign a weight of 0.6 to C_3 and weight C_1 and C_2 by a coefficient of 0.2:

$$\hat{y} = arg\max_i \sum_{j=1}^{m} \omega_j \chi_A(C_j(x) = i) = arg\max_i [0.2 \times i_0 + 0.2 \times i_0 + 0.6 \times i_1] = 1$$

More intuitively, since $3 \times 0.2 = 0.6$, we can say that the prediction made by C_3 has three times more weight than the predictions by C_1 or C_2, which we can write as follows:

$$\hat{y} = mode\{0, 0, 1, 1, 1\} = 1$$

Using the predicted class probabilities instead of the class labels for majority voting can be useful if the classifiers in our ensemble are well calibrated. The modified version of the majority vote for predicting class labels from probabilities can be written as follows:

$$\hat{y} = arg\max_i \sum_{j=1}^{m} \omega_j p_{ij}.$$

Here, p_{ij} is the predicted probability of the jth classifier for class label $\boldsymbol{i}$.

To continue with our previous example, let's assume that we have a binary classification problem with class labels $i \in \{0, 1\}$ and an ensemble of three classifiers C_j , where $j \in \{1, 2, 3\}$. Let's assume that the classifiers C_j return the following class membership probabilities for a particular sample $\boldsymbol{x}$:

$$C_1(x) : [0.9, 0, 1], C_2(x) : [0.8, 0, 2], C_3(x) : [0.4, 0, 6].$$

We can then calculate the individual class probabilities as follows:

$$p(i_0|x) = 0.2 \times 0.9 + 0.2 \times 0.8 + 0.6 \times 0.4 = 0.58$$

$$p(i_1|x) = 0.2 \times 0.1 + 0.2 \times 0.2 + 0.6 \times 0.6 = 0.42$$

$$\hat{y} = arg\max_{i}[p(i_0|x), p(i_1|x)] = 0.$$

2.2.2 Bagging

In contrast to cross-validation, bootstrapping is a random sampling with replacement. Bootstrapping is typically used for statistical estimation of bias and standard error, and a common application in machine learning is to estimate the generalization error of a predictor.

Bagging is an ensemble method for classification or regression analysis in which individual models are trained by random sampling of data in parallel, and the final decision is made by voting among individual models with equal weights or averaging for regression analysis.

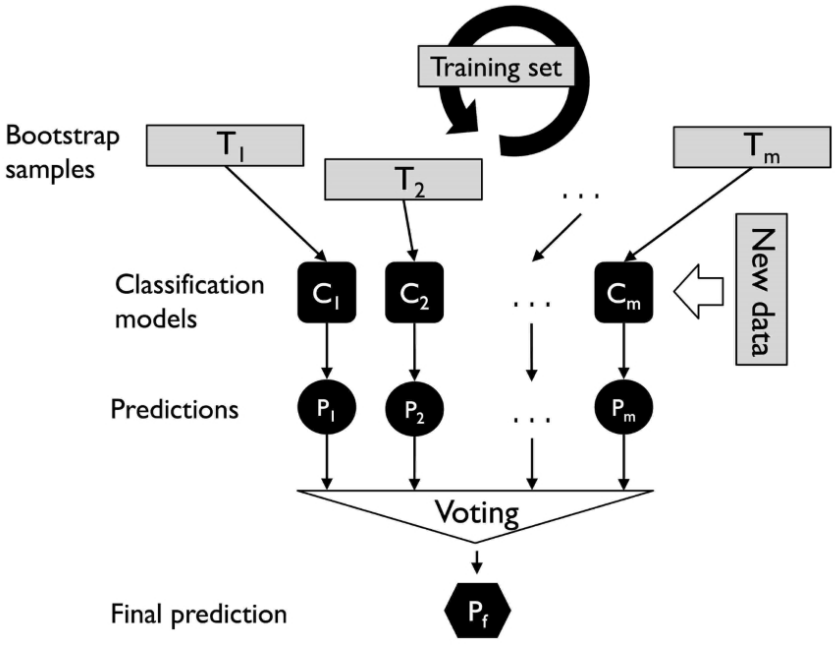


Figure 2.3: The concept of bagging

Bagging as shown in Figure 2.3 is also known as bootstrap aggregating, since

in bagging, we draw random samples with replacement from the initial training set, instead of using the same training set to fit the individual classifiers in the ensemble. In random sampling with replacement, we always return the drawn sample point to the urn so that the probabilities of drawing a particular sample point at each turn does not change, and we could draw the same sample point more than once. Each bootstrap sample is then used to fit a classifier. Once the individual classifiers are fit to the bootstrap samples, the predictions are combined using majority voting. The random forest model in section 3.1.6 we used in the project is an application of bagging technique.

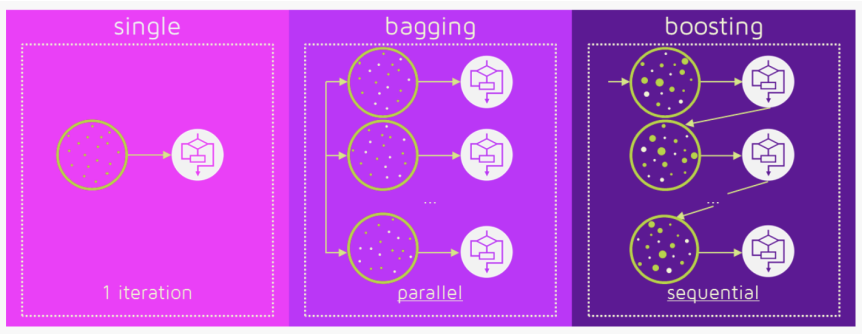


Figure 2.4: Comparison of Bagging and Boosting

2.2.3 Boosting

In boosting, the ensemble consists of weak classifiers. A weak classifier is simply a classifier that performs poorly, but performs better than random guessing. A simple example might be classifying a person as male or female based on their height. The mechanism underlying boosting is to let the weak classifiers learn from misclassified training samples in sequential order to improve the performance of the ensemble. The difference between bagging and boosting is that weak learners

are in parallel in bagging and in sequential order in boosting as shown in Figure 2.4.

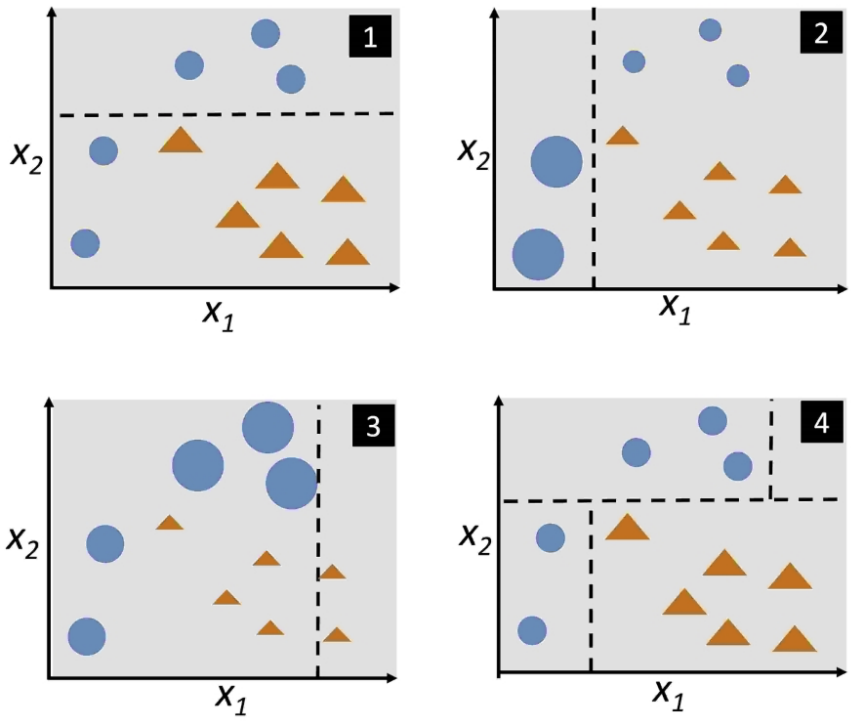


Figure 2.5: The concept of boosting

We use adaptive boosting (AdaBoost) [9], which is a popular variant of boosting. As shown in Figure 2.5, in the step 1 adaptive boosting uses the complete training set to train the weak classifiers. Then it learns from the misclassification of these weak classifiers and reweights the training samples in each iteration (in the step 1, 2 and 3) to build a strong classifier. The mechanism behind AdaBoost is as follows:

Algorithm 2: AdaBoost

a. Set the weight vector w to uniform weights, where $\sum_i w_i = 1$.

b. For j in m boosting rounds, do the following:

1. Train a weighted weak classifier: C_j = train(X, y, w).
2. Predict class labels: $\hat{y}$ = predict(C_j, X).
3. Compute weighted error rate: $\epsilon = w \cdot (\hat{y} \neq y)$.
4. Compute coefficient: $\alpha_j = 0.5 \log \frac{1-\epsilon}{\epsilon}$.
5. Update weights: $w = w \times exp(-\alpha_j \times \hat{y} \times y)$.
6. Normalize weights to sum to 1: $w = w / \sum_i w_i$.

c. Compute the final prediction: $\hat{y} = (\sum_{j=1}^{m}(\alpha_j \times predict(C_j, X)) > 0)$

2.3 Deep Learning

2.3.1 Embedding Layer

The words have been replaced by integers that indicate the absolute popularity of the word in the dataset. The sentences in each text are therefore composed of a sequence of integers. Discrete words are mapped to vectors of continuous numbers. This is useful when working with natural language problems with neural networks and deep learning models where we require numbers as input. Word embedding [10] is a technique where words are encoded as real-valued vectors in a high-dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space. Word embeddings are a technique for representing text where different words with similar meaning have a similar real-valued vector representation.

The layer takes arguments that define the mapping including the vocabulary size (the largest integer value that will be seen as an integer). The layer also

allows you to specify the dimensionality for each word vector, called the output dimension. Let's say that we are only interested in the first 2,000 most used words in the dataset. Therefore our vocabulary size will be 2,000. We can choose to use a 32-dimension vector to represent each word. Finally, we may choose to cap the maximum text length at 150 words, truncating text longer than that and padding text shorter than that with 0 values. We would then use the Keras utility to truncate or pad the dataset to a length of 150 for each observation using the sequence.pad_sequences() function. The output of this first layer would be a matrix with the size 32×150 for a given review training or test pattern in integer format.

2.3.2 1D-CNN

Convolutional Neural Networks [11] apply a filter to an input to create a feature map that summarizes the presence of detected features in the input. The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Using a filter smaller than the input is intentional as it allows the same filter to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom. This systematic application of the same filter across an input is a powerful idea. If the filter is designed to detect a specific type of feature in the input, then the application of that filter systematically across the entire input allows the filter an opportunity to discover that feature anywhere. This capability is commonly referred to as translation invariance, e.g.

the general interest in whether the feature is present rather than where it was present. Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

A standard model for document classification [12] is to use an Embedding layer as input, followed by a one-dimensional convolutional neural network, pooling layer, and then a prediction output layer. The **kernel size** in the convolutional layer defines the number of words to consider as the convolution is passed across the input transcript, providing a grouping parameter. After the Embedding input layer, we insert a Conv1D layer. This convolutional layer has 32 feature maps and reads embedded word representations of 100 vector elements of the word embedding at a time. The convolutional layer is followed by a 1D max pooling layer with a length and stride of 2 that halves the size of the feature maps from the convolutional layer. The rest of the network is the same as the neural network. We can see our convolutional layer preserves the dimensionality of our Embedding input layer of 32-dimensional input with a maximum of 150 words. The pooling layer compresses this representation by halving it.

2.3.3 Multi Channel CNN

A multi-channel convolutional neural network [13] for text classification involves using multiple versions of the standard model with different sized kernels. This allows the text to be processed at different resolutions or different n-grams (groups of words) at a time, whilst the model learns how to best integrate these interpretations. We defined a model with three input channels for processing 3-grams, 5-grams, and 7-grams of text. Each channel is comprised of the following elements:
1. Input layer that defines the length of input sequences.

2. Embedding layer set to the size of the vocabulary and 100-dimensional real-valued representations.
3. One-dimensional convolutional layer with 32 filters and a kernel size set to the number of words to read at once.
4. Max Pooling layer to consolidate the output from the convolutional layer.
5. Flatten layer to reduce the three-dimensional output to two dimensional for concatenation.
6. The output from the three channels are concatenated into a single vector and processed by a dense layer and an output layer.

2.3.4 LSTM

Convolutional neural networks excel at learning the spatial structure in input data. The text data in our project does have a one-dimensional spatial structure in the sequence of words. These learned spatial features may then be learned as sequences by an LSTM layer. Sequence classification [14] is a predictive modeling problem where we have some sequence of inputs over space or time and the task is to predict a category for the sequence. What makes this problem difficult is that the sequences can vary in length, consist of a very large vocabulary of input symbols and may require the model to learn the long-term context or dependencies between symbols in the input sequence.

In our LSTM model, the first layer is the embedded layer that uses 32 length vectors to represent each word. The next layer is the LSTM layer with 100 memory units. Finally, because this is a classification problem we use a dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes in the problem. Because it is a binary classification problem,

log loss is used as the loss function (binary_crossentropy in Keras). The efficient ADAM optimization algorithm is used.

2.3.5 Character-Level CNNs

All of the models mentioned above were based on words. But there has also been research in applying CNNs directly to characters.

The basic idea of using character-level CNN is to transform the data from a sequence of letters into possible categories. The reason we use letters instead of words since the text files are converted from PDF format and words are often misspelled or written differently so looking at character level correlations might work better. In english, all words are formed by 26 (or 52 if including both upper and lower case character, or even more if including special characters). Having the character embedding, every single word's vector can be formed even it is out-of-vocabulary words (optional). On the other hand, word embedding can only handle those seen words. Another benefit is that it good fits for misspelling words, emoticons, new words (e.g. in 2018, Oxford English Dictionary introduced new word which is boba tea. Before that we do not have any pre-trained word embedding for that). It handles infrequent words better than word2vec embedding as later one suffers from lack of enough training opportunity for those rare words. Third reason is that as there are only small amount of vector, it reduces model complexity and improving the performance (in terms of speed).

2.4 Performance Evaluation Metrics

Performance metrics we used to evaluate the classification algorithm are based on the following concepts and formula.

2.4.1 Confusion Matrix

The confusion matrix as shown in Figure 2.6, is used as a way to represent the performance of a classifier and is sometimes also called ”error matrix”. This square matrix consists of columns and rows that list the number of instances as absolute or relative ”actual class” vs. ”predicted class” ratios.

	Predicted condition	
Total population	Predicted Condition positive	Predicted Condition negative
condition positive	**True positive**	**False Negative** (Type II error)
condition negative	**False Positive** (Type I error)	**True negative**

Figure 2.6: Confusion Matrix

True Negatives (TN) is that case was negative and predicted negative. True Positives (TP) is that case was positive and predicted positive. False Negatives (FN) is that case was positive but predicted negative. False Positives (FP) is that case was negative but predicted positive.

2.4.2 True and False Positive Rates

As shown in Figure 2.6, the True Positive Rate (TPR) and False Positive Rate (FPR) are performance metrics that are especially useful for imbalanced class problems. For example, in spam classification we are of course primarily interested in the detection and filtering out of spam. However, it is also important to

decrease the number of messages that were incorrectly classified as spam (False Positives), since missing an important message is worse than ending up with a few spam messages in e-mail inbox. In contrast to the False Positive Rate (FPR), the True Positive Rate (TPR) provides useful information about the fraction of positive (or relevant) samples that were correctly identified out of the total pool of Positives.

$$\text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{FN+TP}}$$

$$\text{False Positive Rate (FPR)} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP+TN}}$$

2.4.3 Accuracy, Precision, Recall and F measure

As shown in Figure 2.6, accuracy is defined as the fraction of correct classifications out of the total number of samples; it resembles one way to assess the performance of a predictor and is often used synonymous to specificity/precision although it is calculated differently. Accuracy is calculated as (TP+TN)/(P+N), where TP=True Positives, TN=True Negatives, P=Positives, N=Negatives.

Precision (synonymous to specificity) and recall (synonymous to sensitivity) are two measures to assess performance of a classifier if class label distributions are skewed. Precision is defined as the ratio of number of relevant items out of total retrieved items, whereas recall is the fraction of relevant items which are retrieved.

Accuracy is the proportion of true results (both true positives and true nega-

tives) among the total number of cases examined:

$$\text{Accuracy (ACC)} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}.$$

Precision is the probability that a (randomly selected) retrieved document is relevant:

$$\text{Precision (PRE)} = \frac{\text{TP}}{\text{TP+FP}}.$$

Recall is the probability that a (randomly selected) relevant document is retrieved in a search:

$$\text{Recall (REC)} = \text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{FN+TP}}.$$

F measure is a measure that combines precision and recall is the harmonic mean of precision and recall, the traditional F-measure or balanced F-score:

$$\text{F measure } (F_1) = 2 \cdot \frac{\text{PRE} \cdot \text{REC}}{\text{PRE+REC}}.$$

2.4.4 Receiver Operator Characteristics (ROC)

Receiver Operator Characteristics (ROC) curves as shown in Figure 2.7 are useful tools to select classification models based on their performance with respect to True Positive and the False Positive rates.

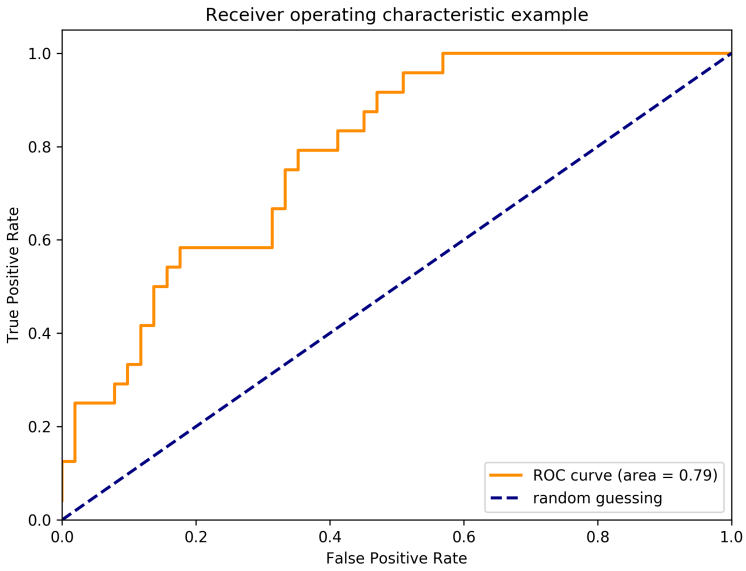


Figure 2.7: Example of a Receiver Operating Characteristic. This plot was created using the Python scikit-learn machine learning library.

The diagonal of a ROC graph can be interpreted as random guessing and classification models that fall below the diagonal are considered as worse than random guessing. A perfect classifier would fall into the top left corner of the graph with a True Positive Rate of 1 and a False Positive Rate of 0. Based on the ROC curve, the so-called Area Under the Curve (AUC) can be calculated to characterize the performance of a classification model. The bigger AUC value, the better classification model.

2.5 Input Data Representation

Since we can't feed text to machine learning algorithms directly, the text data need to be represented in the form of numerical feature vectors. The bag of words model would help us to complete this task.

2.5.1 Bag-of-Words model

Bag of words is a model that is used to construct sparse feature vectors for text classification tasks. The bag of words is an unordered set of all words that occur in all documents that are part of the training set. Every word is then associated with a count of how often it occurs whereas the positional information is ignored. Sometimes, the bag of words is also called ”dictionary” or ”vocabulary” based on the training data. The mechanism of bag-of-words model is that the model creates a vocabulary of unique words from the entire set of documents, and then constructs a feature vector for each file. The feature vector contains the counts of words appearing in the specific file. Usually we would get sparse feature vectors since the unique words in each file is usually only a small subset of all words in the whole vocabulary.

Suppose the original files contains the following words:
File1 = 'Statistical analysis of the data'
File2 = 'The exact distance from the edge of the plot varied slightly to avoid wheel tracks'
File3 = 'These soil samples were used for obtaining aggregates and conducting aggregate size analyses'
File4 = 'The duplicated cores were obtained for three depths mentioned above'
File5 = 'Aggregate size distribution and stability'

The vocabulary maps the unique words to integer indices of feature vectors.
vocabulary = [('above', 0), ('aggregate', 1), ('aggregates', 2), ('analyses', 3), ('analysis', 4), ('and', 5), ('avoid', 6), ('conducting', 7), ('cores', 8), ('data', 9), ('depths',

10), ('distance', 11), ('distribution', 12), ('duplicated', 13), ('edge', 14), ('exact', 15), ('for', 16), ('from', 17), ('mentioned', 18), ('obtained', 19), ('obtaining', 20), ('of', 21), ('plot', 22), ('samples', 23), ('size', 24), ('slightly', 25), ('soil', 26), ('stability', 27), ('statistical', 28), ('the', 29), ('these', 30), ('three', 31), ('to', 32), ('tracks', 33), ('used', 34), ('varied', 35), ('were', 36), ('wheel', 37)]

Then we could get sparse feature vectors for each file as follows:

File1 = [0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0]

File2 = [0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 3 0 0 1 1 0 1 0 1]

File3 = [0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 1 0 1 0]

File4 = [1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0]

File5 = [0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0],

where each index position in the feature vectors corresponds to the integer values in the vocabulary map.

The model above is a 1-gram or unigram model since each token in the vocabulary is only a single word. However, for different tasks, we could choose the size of n-gram model. Like if we choose n=2, then we could get the following 2-gram vocabulary map via 2-gram or bigram model.

2-gram vocabulary = [('aggregate size', 0), ('aggregates and', 1), ('analysis of', 2), ('and conducting', 3), ('and stability', 4), ('avoid wheel', 5), ('conducting aggregate', 6), ('cores were', 7), ('depths mentioned', 8), ('distance from', 9), ('distribution and', 10), ('duplicated cores', 11), ('edge of', 12), ('exact distance', 13), ('for obtaining', 14), ('for three', 15), ('from the', 16), ('mentioned above', 17), ('obtained for', 18), ('obtaining aggregates', 19), ('of the', 20), ('plot varied', 21), ('samples were', 22), ('size analyses', 23), ('size distribution', 24), ('slightly to', 25), ('soil samples', 26),

('statistical analysis', 27), ('the data', 28), ('the duplicated', 29), ('the edge', 30), ('the exact', 31), ('the plot', 32), ('these soil', 33), ('three depths', 34), ('to avoid', 35), ('used for', 36), ('varied slightly', 37), ('were obtained', 38), ('were used', 39), ('wheel tracks', 40)]

2.5.2 Term Frequency-Inverse Document Frequency (TF-IDF)

A bag-of-words model which we used in traditional machine learning is a way of extracting features from text so the text input can be used with machine learning algorithms like logistic regression [15], support vector machine, random tree, etc. Each text is converted into a vector representation. The number of items in the vector representing a text corresponds to the number of words in the vocabulary. The larger the vocabulary, the longer the vector representation. Words in a text are scored and the scores are placed in the corresponding location in the representation. However, the frequently appearing words typically do not contain much useful information for text classification if these words occur across multiple texts from both or all classes. This is why we need to use the term frequency-inverse document frequency technique to downweight these types of words in the feature vectors. The TF-IDF is defined as the product of the term frequency and the inverse document frequency.

However, the frequently appearing words typically do not contain much useful information for documents classification if these words occur across multiple documents from both or all classes. This is why we need to use term frequency-inverse document frequency technique to downweight these types of words in the feature vectors. The TF-IDF is defined as the product of the term frequency and the inverse document frequency:

$$\text{TF-IDF}(t,d) = \text{TF (t,d)} \times \{\text{IDF}(t,d) + 1\},$$

where the TF(t, d) is the term frequency, and the inverse document frequency IDF(t, d) is calculated as:

$$\text{IDF}(t,d) = \log \frac{1 + n_d}{1 + \text{DF}(d,t)},$$

where n_d is the total number of documents, and DF(d, t) is the number of documents d that contain the term t. Adding the constant 1 to the denominator is optional and serves the purpose of assigning a non-zero value to terms that occur in all training samples; the log function is used to ensure that low document frequencies are not given too much weight.

After this TF-IDF transformation, we normalize the feature vectors by L2-Norm formula:

$$v_{\text{norm}} = \frac{v}{|v|} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}}$$

After TF-IDF transformation and L2 normalization, we could get the feature vectors as shown in the following. Then we could see the 'the' weights decrease from 3 to 0.51 since it appears in File1, File2, and File4 and this indicates 'the' does not contain much information for document classification.

File1 = [0. 0. 0. 0. 0.49 0. 0. 0. 0. 0.49 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.4 0. 0. 0. 0. 0. 0. 0.49 0.33 0. 0. 0. 0. 0. 0. 0. 0.]

File2 = [0. 0. 0. 0. 0. 0. 0.25 0. 0. 0. 0. 0.25 0. 0. 0.25 0.25 0. 0.25 0. 0. 0. 0.2 0.25 0. 0. 0.25 0. 0. 0. 0.51 0. 0. 0.25 0.25 0. 0.25 0. 0.25]

File3 = [0. 0.24 0.3 0.3 0. 0.24 0. 0.3 0. 0. 0. 0. 0. 0. 0. 0. 0.24 0. 0. 0. 0.3 0. 0. 0.3

0.24 0. 0.3 0. 0. 0. 0.3 0. 0. 0. 0.3 0. 0.24 0.]

File4 = [0.34 0. 0. 0. 0. 0. 0. 0. 0.34 0. 0.34 0. 0. 0.34 0. 0. 0.27 0. 0.34 0.34 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.23 0. 0.34 0. 0. 0. 0. 0.27 0.]

File5 = [0. 0.41 0. 0. 0. 0.41 0. 0. 0. 0. 0. 0. 0.5 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.41 0. 0. 0.5 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

2.6 Train/Test Data Split and k-fold cross-validation

K-fold cross-validation is a resampling technique without replacement, where each sample only occur exactly once in the folds. This yields a lower variance estimate of the model performance. In k-fold cross-validation the data is split into k subsets, then a prediction/classification model is trained k times, each time holding one subset as the test set, training the model parameters using the remaining k-1 subsets. Finally, cross-validation error is evaluated as the average error out of all k training models.

Here we use 10-fold cross-validation, where we randomly split the training data into 10 folds without replacement. Nine folds are used for the training, and 1 fold is used for performance evaluation. This procedure is repeated 10 times so that we obtain 10 models and performance estimates and average the individual model estimates. As shown in Figure 2.8, E_1, E_2, E_3, ..., E_{10} are performance estimates via 10-fold cross-validation. E is the average of the individual model estimates.

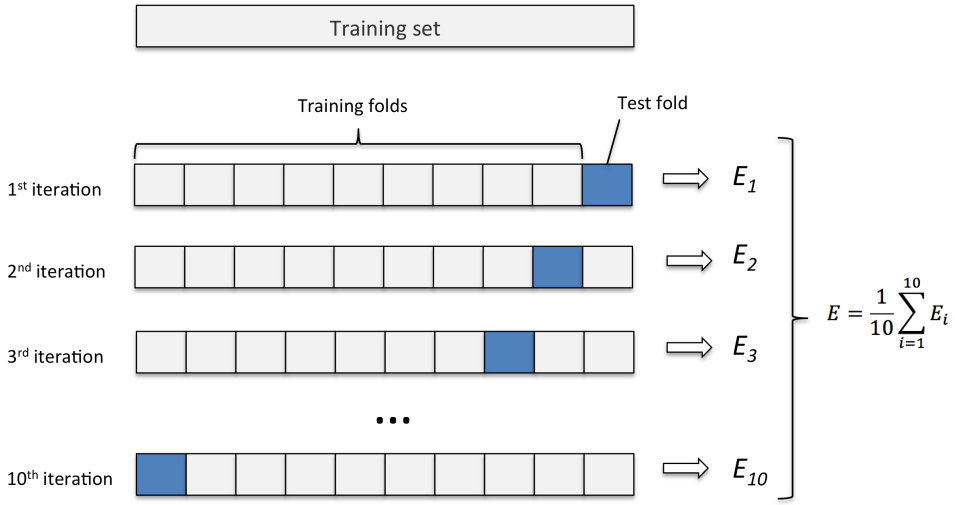


Figure 2.8: 10-fold cross-validation

Chapter 3

Related Work

3.1 Deep Learning-Based Methods

An interesting use case of CNNs in NLP can be found in [16] and [17], coming out of Microsoft Research. These papers describe how to learn semantically meaningful representations of sentences that can be used for Information Retrieval. The example given in the papers includes recommending potentially interesting documents to users based on what they are currently reading. The sentence representations are trained based on search engine log data.

Santos and Zadrozny [18] learn character-level embeddings, joins them with pre-trained word embeddings, and uses a CNN for Part of Speech tagging. Xiang and LeCun [19] [20] explore the use of CNNs to learn directly from characters, without the need for any pre-trained embeddings. Notably, the authors use a relatively deep network with a total of 9 layers, and apply it to Sentiment Analysis and Text Categorization tasks. Results show that learning directly from character-level input works very well on large datasets (millions of examples), but underperforms simpler models on smaller datasets (hundreds of thousands of examples). Kim

[21] explores to application of character-level convolutions to Language Modeling, using the output of the character-level CNN as the input to an LSTM at each time step. The same model is applied to various languages.

Xiang and Yann [22] introduced character CNN. They found that character includes key signal to improve model performance. In the paper, a list of character are defined 70 characters which including 26 English letters, 10 digits, 33 special characters and new line character.

3.2 Named Entity Recognition

Stanford NER [23] is a Java implementation of a Named Entity Recognizer. Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Included with the download are good named entity recognizers for English, particularly for the 3 classes (PERSON, ORGANIZATION, LOCATION), and they also make available on various other models for different languages and circumstances, including models trained on just the CoNLL 2003 English training data.

Stanford NER is also known as CRFClassifier. The software provides a general implementation of (arbitrary order) linear chain Conditional Random Field (CRF) sequence models. That is, by training your own models on labeled data, you can actually use this code to build sequence models for NER or any other task. The original CRF code is by Jenny Finkel. The feature extractors are by Dan Klein, Christopher Manning, and Jenny Finkel. Much of the documentation and usability is due to Anna Rafferty. More recent code development has been done by various

Stanford NLP Group members.

Stanford NER is available for download, licensed under the GNU General Public License (v2 or later). Source is included.

3.3 Table Analysis

The purpose of table analysis is to covert the content of human-readable tables to a query-table store of machine-manipulable assertions. Tables provide a convenient and succinct way to communicate data of interest to human readers. Tables are not, however, inherently amenable to machine-based search and query. A source table may be any file representation that allows rendering (printing or displaying) the essential characteristics of a source table in a form suitable for a human reader, where layout, rulings and typesetting are often used to reveal the intrinsic relationship between headers and content cells.

In Seth et al.'s algorithm [3], critical cells (CC1, CC2, CC3, CC4) delineate regions. In a WFT every critical cell must appear in the grid. As shown in Figure 3.1, CC1 and CC2 demarcate the StubHeader and CC3 and CC4 demarcate the Data region. Furthermore, in combination with one another, these critical cells also demarcate both the ColHeader and RowHeader regions. Letting row r_i and column c_i be the coordinates of critical cell CCi, a WFT satisfies the following constraints: $r_1 \leq r_2 < r_3 \leq r_4$ and $c_1 \leq c_2 < c_3 \leq c_4$. These constraints guarantee that the ColHeader and RowHeader regions properly align with the Data region and that the Data region is not degenerate. A single row or column of data is acceptable, provided both row and column headers exist.

They do not deal here with concatenated (composite) tables, nested tables (tables whose data cells may themselves be tables), tables containing graphic data,

or “egregious” tables (those not laid out on a grid with headers above and to the left).

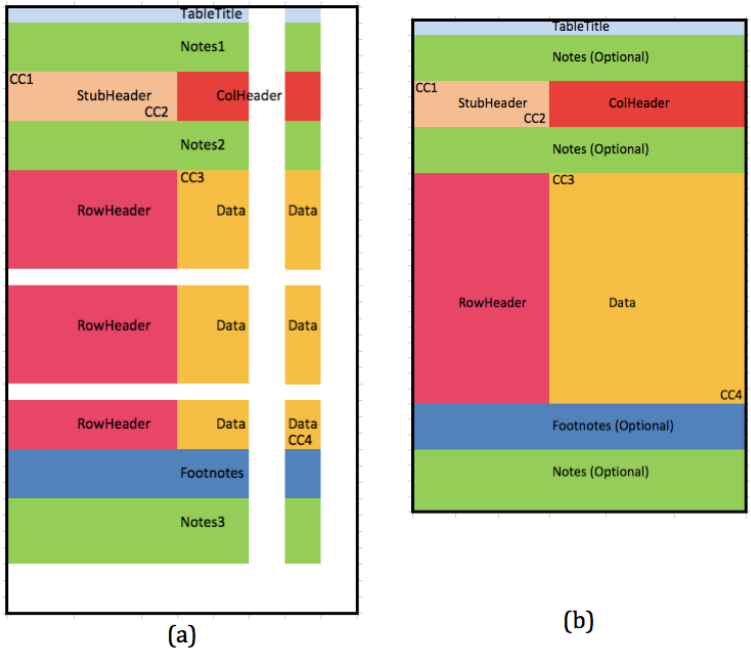


Figure 3.1: Visual WFT model: (a) with and (b) without blank rows and columns. *CC*1 and *CC*2 demarcate the StubHeader and *CC*3 and *CC*4 demarcate the Data region.

Their program could transform well-formed tables to a new canonical table format via: segmenting table regions by algorithmic data cell indexing, factoring header paths into categories by algorithmic header analysis, and generating queryable canonical relational tables.

In a well-formed table (WFT), every data cell is uniquely indexed by its row and column header paths, which are respectively left of and above the data region. A hierarchical (row or column) header may index one or more categories. A single-category header path consists of the root-to-leaf path of the corresponding category tree. A multi-category header path consists of concatenated category paths.

Chapter 4

Methods

The system overview shown in Figure 4.1 gives a outline of the contributions of this thesis. My work consists of five parts: Web harvesting, Text Classification, Table Analysis, Named Entity Recognition and Database System Build.

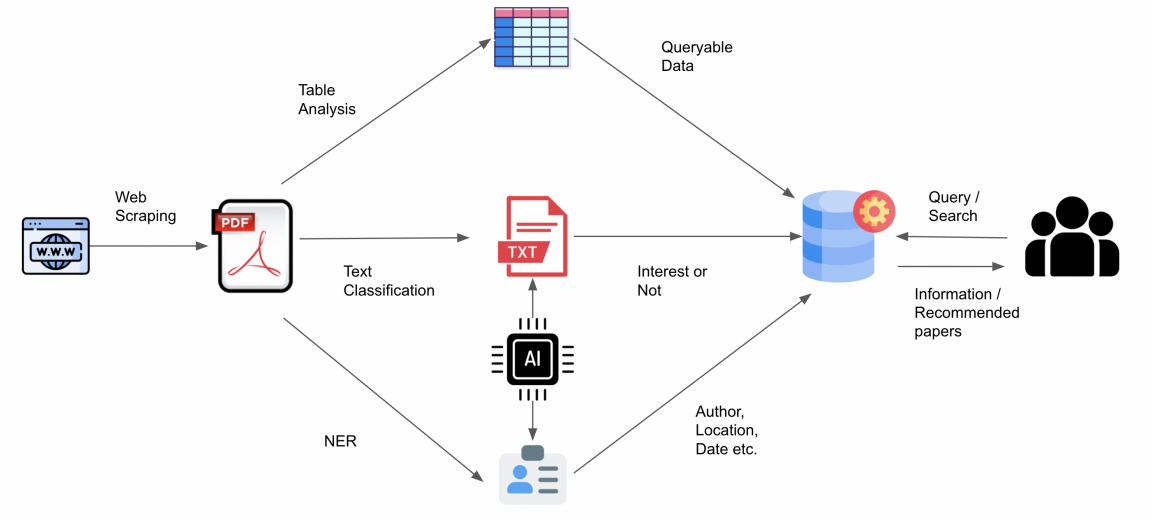


Figure 4.1: System Overview

First, we built a web crawler to download soil science journal papers which are the basis to build entire machine learning based system. Then I used tools

to convert downloaded journal papers from PDF to TXT format, which is vital for following stages. Second, text classification can help identify the section or paragraph in a paper that may be of interest to users based on their own search interest. Third, Named Entity Recognition can extract author and experiment location from paper to store them in data system. In the forth step, another important information which is of interest to users is table. I did not find any good method to extract table from PDF format directly. It is also not valid for TXT format since the table structure will lose after format conversion. I followed Prof. Seth's well formed table requirement and create tables in CSV format manually and used his program to store the table in a queryable machine readable format. Finally, I populate a relational database with information automatically extracted from journal papers collected from internet resources. The users can query and search key words or highlight some section or paragraph in the paper, then the system will provide the relevant information or recommend a paper which is of interest to users. This system will make future queries more efficient.

Chapter 5

Data Statistical Description

We have in total 207 papers and divide each paper into sections to apply text classification on them. The total number of files is 1690. The total number of journals is 207. And the average files per journal is 8.2. The labels in the table 5.1 and 5.2 indicate if users are interested in the content.

Table 5.1: Section text data statistical description

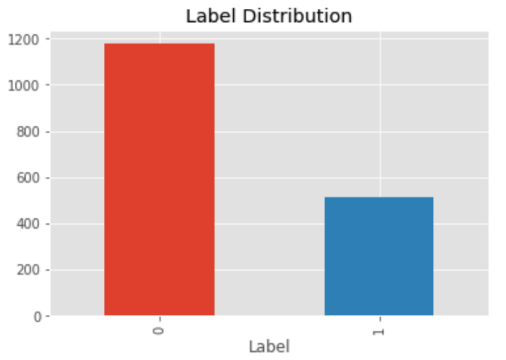
		File Name	N_chs_cleanText				N_words_cleanText			
		count	min	max	sum	mean	min	max	sum	mean
Journal	Label									
001	no interest	5	304	17013	33326	6665	42	2648	5047	1009
	interest	2	908	5252	6160	3080	116	810	926	463
002	no interest	3	1500	7808	12343	4114	212	1035	1677	559
	interest	4	358	8843	15664	3916	49	1422	2471	617
003	no interest	4	269	6096	13492	3373	33	835	1901	475
	interest	3	1026	13346	21576	7192	137	2052	3311	1103
004	no interest	4	1946	11662	27082	6770	279	1694	3912	978
	interest	3	881	12224	18897	6299	130	1874	2876	958
005	no interest	6	445	11785	30407	5067	67	1767	4532	755
	interest	3	1068	9210	17231	5743	144	1454	2754	918

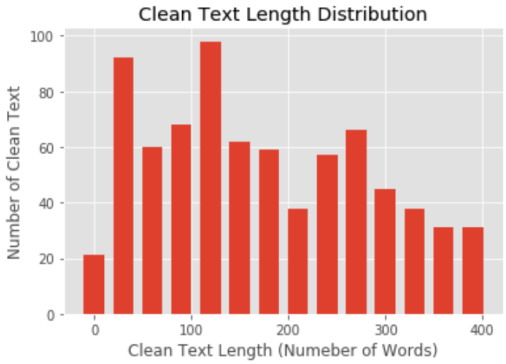
Table 5.2: Paragraph text data statistical description

		File Name	N_chs_cleanText				N_words_cleanText			
		count	min	max	sum	mean	min	max	sum	mean
Journal	Label									
001	no interest	24	257	8532	33948	1414	39	1262	5152	214
	interest	7	107	1345	5478	782	17	206	821	117
002	no interest	23	263	7810	22388	973	39	1035	3292	143
	interest	7	337	1709	5562	794	49	250	856	122
003	no interest	23	271	6098	29950	1302	33	835	4450	193
	interest	5	487	1733	5067	1013	71	275	762	152
004	no interest	32	136	9099	35797	1118	20	1318	5241	163
	interest	10	572	1735	10089	1008	87	283	1547	154
005	no interest	19	298	8025	33275	1751	40	1161	4982	262
	interest	14	380	1797	14307	1021	58	294	2304	164

To test if I can use text classification on less information, I divide each paper into paragraphs which contain less information than sections. The total number of files is 7543. The total number of journals is 207. And the average files per journal is 36.4. The labels in the table 5.2 indicate if users are interested in the content, where 0 means not interest and 1 means interest.

Table 5.1 and Table 5.2 show the text description in our raw data, and they only show the top 5 files in section and paragraph text, due to limit space. Journal column indicates the journal index whose range is from 001 to 207. Label denotes if the content is relevant with researchers' interest, where 0 means no interest, while 1 means interest. File Name count column shows how many files in each label, for example, 5 means there are 5 files in Journal 001 which the researchers are interested in. Let still use the first row in Table 5.1 as an example. Number of characters in the clean text is 304 in minimum, 17013 in maximum, sum is 33326, and 6665 on average. Number of words in the clean text is 42 in minimum, 2648 in maximum, sum is 5047, and 1009 on average. Figure 5.1 and Figure 5.2 indicates





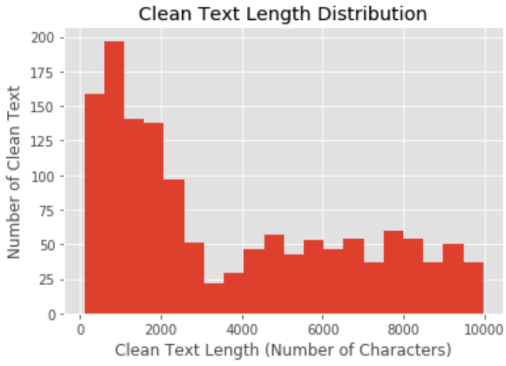
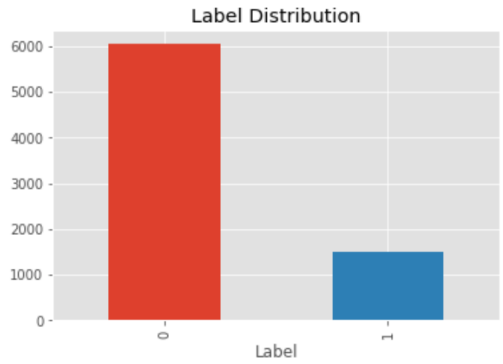
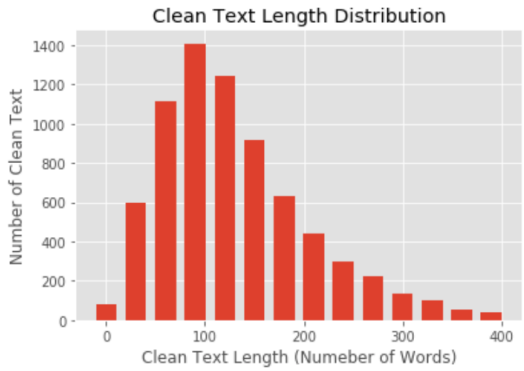


Figure 5.1: Distribution of Section Data





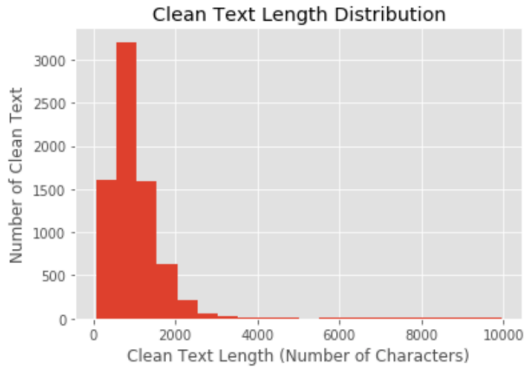


Figure 5.2: Distribution of Paragraph Data

the distribution of labels, distribution of number of words, and distribution of number of characters in section and paragraph data.

These two tables and distribution Figure 5.1 and Figure 5.2 provide a outline to show us what the data looks like.

Chapter 6

Web Scraping

In the first pass of paper downloading, we downloaded 3,657 papers with a combined size of 4.15 GB. These papers were collected from on-line sources of journals using the following query terms: “soil quality” and “conservation management”. In the second pass of paper downloading, we downloaded 34,787 with a combined size of 25.38 GB. These papers were collected from on-line sources of journals using the following query terms: “Soil Quality”, “Soil Management”, “Dynamic Soil Properties” and “Soil Health”. In both searches, we filtered out items that were not journal articles. The total papers I downloaded numbered 38,444 and the total size was 29.53 GB. The details about numbers of papers and related sources are listed in the Table 6.1.

6.1 Procedure of Web Scraping

The first time, the library I chose to download papers is ACSEE Digital Library through UNL library. I filtered out Meeting Session, Book Chapter, and other resources, and keep files from Journal Article. The second time, I download

papers from more resources by filtering words like Soil Quality, Soil Management, Dynamic Soil Properties and Soil Health as shown in Figure 6.1 and Figure 6.2.

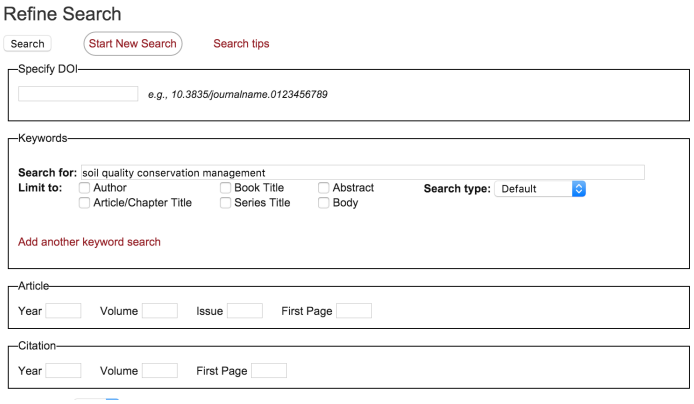


Figure 6.1: Restrict to some terms: soil quality, conservation management.

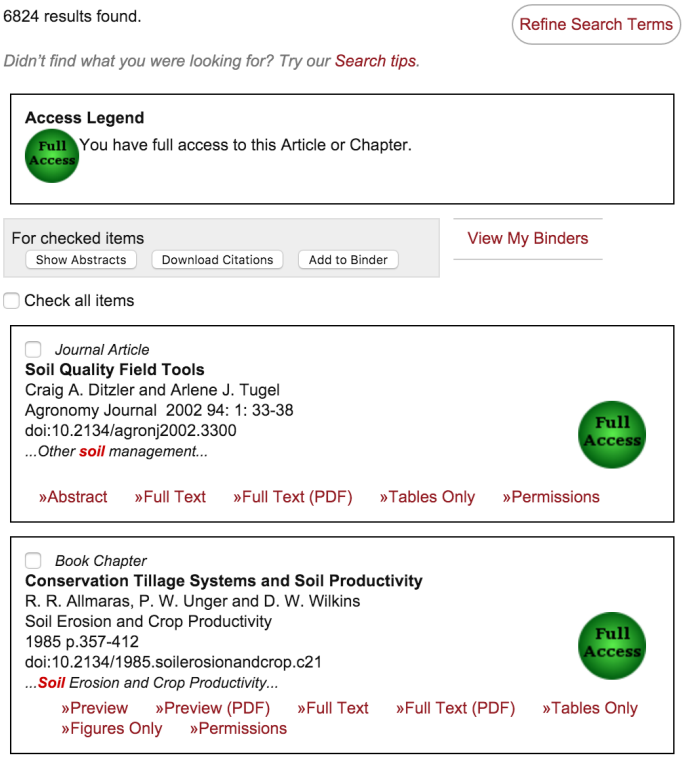


Figure 6.2: Search results with filter words.

During web scraping, the code scanned and extracted link to papers in each

web page, and went to next page automatically until no more page. Then it stored all papers links and finally downloaded them all.

6.2 Summary of Downloaded Papers

Table 6.1: Downloaded Journal Papers and associated recourse

No. papers	Journal Name	Resource
9919	Soil Science of America	https://dl.sciencesocieties.org/publications/sssaj
5425	Journal of Environmental Quality	https://dl.sciencesocieties.org/publications/jeq
8788	Agronomy	https://dl.sciencesocieties.org/publications/aj
4188	Crop Science	https://dl.sciencesocieties.org/publications/cs
43	Soil and Tillage Research	http://www.journals.elsevier.com/soil-and-tillage-research
25	Agricultural Water Management	http://www.journals.elsevier.com/agricultural-water-management
101	Agriculture Ecosystems & Management	http://www.journals.elsevier.com/agriculture-ecosystems-and-environment
112	Journal of Environmental Management	http://www.journals.elsevier.com/journal-of-environmental-management
42	Applied Soil Ecology	http://www.journals.elsevier.com/applied-soil-ecology
107	Forest Ecology and Management	http://www.journals.elsevier.com/forest-ecology-and-management
47	Soil Biology and Biochemistry	http://www.journals.elsevier.com/soil-biology-and-biochemistry
23	Catena	http://www.journals.elsevier.com/catena
74	Ecological Indicators	http://www.journals.elsevier.com/ecological-indicators/
74	Geoderma	http://www.journals.elsevier.com/geoderma
34	Soil Use and Management	http://onlinelibrary.wiley.com/journal/10.1111/(ISSN)1475-2743
34	Ecological Applications	http://esajournals.onlinelibrary.wiley.com/hub/journal/10.1002/(ISSN)1939-5582/
36	Plant and soil	http://link.springer.com/journal/11104
196	Environmental Monitoring & Assessment	http://link.springer.com/journal/10661
3205	Journal of Soil and Water Conservation	http://www.jswconline.org/
2314	Soil Research	http://www.publish.csiro.au/nid/84.htm

Chapter 7

Text Analysis via Machine/Deep Learning

Text classification is a way to categorize documents or pieces of text. By examining the word usage in a piece of text, classifiers can decide what class label to assign to it. A binary classifier decides between two labels, such as positive review or negative review, desirable or not desirable information. The text can either be one label or another, but not both. The purpose of text classification in this project is to classify the unknown journal paper or pieces of text in it as desirable information or not by training on already highlighted desirable documents, in order to save the users' new paper seeking time and save the desirable information in queryable database.

We need to first convert PDF to text format, since the text can not be read directly from PDF format. There are many conversion tools and a lot of variance in output quality. Converting PDF to text is one of the most common features for standard PDF converting tool. However, there could be great difference in output

quality. In our daily documents processing, PDF that with multi-column text is somehow inevitable. Unfortunately, many PDF Text converters handle single column text well but fail miserably when presented with a typical multiple-column layout by interlacing the multiple columns. For these journal papers, we need to clean the text, since after conversion from PDF format the text would get scrambled, with pieces of left column being mixed with the right one. Some papers have three columns, making the problem more serious. Another common problem is that the position of splitting is not fixed. Part of content in the first paragraph may be split to the second, or even third paragraph. These would make cleaning text tough.

In this section, we delve into text analysis and use machine learning algorithms to classify documents or pieces of text (sentence, paragraph, section) based on the attitude or emotions of the end user, like interested in them or not. The details of machine learning algorithms and performance evaluation metrics we used here are in section 3. For the section classification problem, it consists of 1690 files that are labeled as 1177 positive and 513 negative, where positive means that the user is interested in that text and negative means that the user is not interested in that text. And for the paragraph classification task, it consists of 7543 files (6045 positives and 1498 negatives). I spent a lot of time on the labeling process since the documents are labeled manually. The positves and negatives are placed on different folders. After we got these files, we preprocess them into a useable format for machine learning algorithms, and extract meaningful information from them to feed to models. Then we use these models to predict whether the user is interested in the text or not.

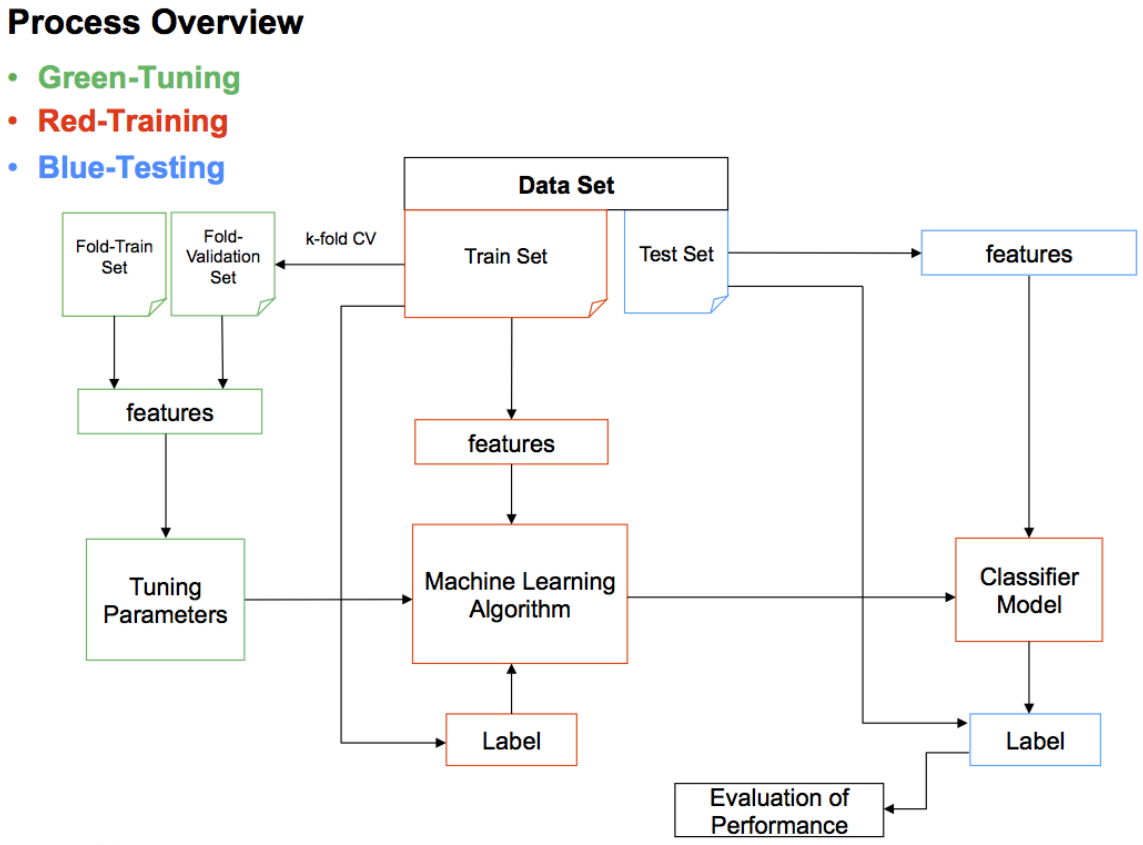


Figure 7.1: Machine Learning System Design

7.1 Data Preprocessing

To handle text data easier, we read the text data into a pandas DataFrame object and it gives more structured data and better visualization.

7.1.1 Clean Data

We first clean numbers, punctuation marks, and other non letter characters in the text data, since they do not contain much useful semantic information in our project.

7.1.2 Tokenization

Tokenization is the process of breaking down a text corpus into individual elements that serve as input for various natural language processing algorithms. Usually, tokenization is accompanied by other optional processing steps, such as the removal of stop words and punctuation characters, stemming or lemmatizing, and the construction of n-grams.

7.1.2.1 Stop Words

We remove the stop words, since they are pretty common in all kinds of texts and do not contain much useful information for document classification. NLTK library [24] has a set of 127 English stop words. And we could use it to remove stop words in the text.

7.1.2.2 Lowercase

Then we convert the text into lowercase characters, since the semantic information does not depend on whether the word is at the start of the sentence or not. Another reason is our model does not distinguish the letter case difference, since unigram bag-of-words model does not concern the order of the words.

7.1.2.3 Stemming and Lemmatization

Stemming describes the process of transforming a word into its root form. The original stemming algorithm was developed my Martin F. Porter in 1979 and is hence known as Porter stemmer [25]. Stemming can create non-real words, such as "thu" in the example above. In contrast to stemming, lemmatization aims to obtain

the canonical (grammatically correct) forms of the words, the so-called lemmas. Lemmatization is computationally more difficult and expensive than stemming.

7.1.2.4 N-Grams

In the n-gram model [26], a token can be defined as a sequence of n items. The simplest case is the so-called unigram (1-gram) where each token consists of exactly one word, letter, or symbol. Choosing the optimal number n depends on the language as well as the particular application. For example, Andelka Zecevic found in his study that n-grams with $3 \leq n \leq 7$ were the best choice to determine authorship of Serbian text documents [27]. In a different study, the n-grams of size $4 \leq n \leq 8$ yielded the highest accuracy in authorship determination of English text books [28] and Kanaris and others report that n-grams of size 3 and 4 yield good performances in anti-spam filtering of e-mail messages [29]. In our work, we chose range 1 to 3 as n-gram grid search search to balance train time and performance due to compute resource limit.

7.2 Fine Tuning Hyperparameters

In machine learning, we have two types of parameters: One are the parameters that the machine learning algorithm learned from the training data like the weights in the logistic regression, neural network, which we would get in the training step. The other are tuning parameters, which are called hyperparameters, like the regularization parameter in the logistic regression, the maximum depth of a decision tree and number of estimators in the random forest.

Now we need to tune the hyperparameters in our machine learning models.

We use a grid search to find the optimal set of parameters by finding the optimal combination of hyperparameters values for model using stratified 10-fold cross-validation. The reason why we use stratified 10-fold cross-validation instead of the standard 10-fold cross-validation is that our dataset has unequal class proportions. In the stratified 10-fold cross-validation, the class proportions are preserved in each fold to ensure that each fold is representative of the class proportions in the training dataset, and this would yield better bias and variance estimates on this type of dataset.

The approach of grid search is a brute force exhaustive search paradigm where we specify a list of values for different hyperparameters, and the computer evaluates the model performance for each combination of those to obtain the optimal combination of values. Here, we use 10-fold cross-validation for tuning hyperparameters, since it would help to find the optimal hyperparameter values that yields a satisfying generalization performance.

7.2.1 Logistic Regression

- ngram range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min\_n <= n <= max\_n$ will be used.
- stop words: A list of words which will be removed from the resulting tokens.
- penalty: Used to specify the norm used in the penalization.
- C: Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

```
param_grid = [{'vect__ngram_range': [(1, 2)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer, tokenizer_porter],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [0.1, 1.0, 10.0, 100.0]},
              {'vect__ngram_range': [(1, 2)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer, tokenizer_porter],
               'vect__use_idf':[False],
               'vect__norm':[None],
               'clf__penalty': ['l1', 'l2'],
               'clf__C': [0.1, 1.0, 10.0, 100.0]}]
```

Figure 7.2: Logistic Regression Parameters Grid Search Code

7.2.2 SVM

```
param_grid = [{'vect__ngram_range': [(1, 2)],
               'vect__stop_words': [stop, None],
               'vect__tokenizer': [tokenizer, tokenizer_porter],
               'clf__kernel': ['linear', 'rbf'],
               'clf__C': param_range}
              ]
```

Figure 7.3: Support Vector Machine Parameters Grid Search Code

- ngram range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min\_n <= n <= max\_n$ will be used.
- stop words: A list of words which will be removed from the resulting tokens.
- kernel: Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used.

- C: Penalty parameter C of the error term.

7.2.3 Decision Tree

```
param_grid = {'vect__ngram_range': [(1, 2)],
              'vect__stop_words': [stop, None],
              'vect__tokenizer': [tokenizer, tokenizer_porter],
              'clf__max_depth': np.arange(1, 30, 2)
             }
```

Figure 7.4: Decision Tree Parameters Grid Search Code

- ngram range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min\_n <= n <= max\_n$ will be used.
- stop words: A list of words which will be removed from the resulting tokens.
- maximum depth: The maximum depth of the tree.

7.2.4 Naive Bayes

```
param_grid = {'vect__ngram_range': [(1, 3)],
              'vect__stop_words': [stop, None],
              'vect__tokenizer': [tokenizer, tokenizer_porter],
              "clf__alpha": np.arange(0.1, 3, 0.1),
              "clf__fit_prior": [True, False],
             }
```

Figure 7.5: Naive Bayes Parameters Grid Search Code

- ngram range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min\_n <= n <= max\_n$ will be used.
- stop words: A list of words which will be removed from the resulting tokens.
- alpha: Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).
- fit_prior: Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

7.2.5 K Nearest Neighbors

```
param_grid = {'vect__ngram_range': [(1, 2)],
              'vect__stop_words': [stop, None],
              'vect__tokenizer': [tokenizer, tokenizer_porter],
              "clf__leaf_size": np.arange(10, 20, 5),
              "clf__p": [1, 2],
              "clf__metric": ['minkowski']}
```

Figure 7.6: K Nearest Neighbors Parameters Grid Search Code

- ngram range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min\_n <= n <= max\_n$ will be used.
- stop words: A list of words which will be removed from the resulting tokens.
- leaf_size: Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

- p: Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

- metric: the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of the DistanceMetric class for a list of available metrics.

7.2.6 Random Forest

```
param_grid = {'vect__ngram_range': [(1, 2)],
              'vect__stop_words': [stop, None],
              'vect__tokenizer': [tokenizer, tokenizer_porter],
              "clf__n_estimators": np.arange(10, 150, 50),
              "clf__max_depth": np.arange(1, 20, 8),
              "clf__min_samples_split": np.arange(10,100,50),
              "clf__min_samples_leaf": np.arange(10,100,50),
              "clf__max_leaf_nodes": np.arange(10,30,10),
              'clf__class_weight': [{0:1, 1:1}, {0:1, 1:2}, {0:1, 1:3}],
              "clf__bootstrap": [True, False],
              "clf__criterion": ["gini", "entropy"]}
```

Figure 7.7: Random Forest Parameters Grid Search Code

- ngram range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min\_n <= n <= max\_n$ will be used.

- stop words: A list of words which will be removed from the resulting tokens.

- n_estimators: The number of trees in the forest.

- maximum depth: The maximum depth of the tree.

- minimum samples split: The minimum number of samples required to split an internal node.
- minimum samples leaf: The minimum number of samples required to be at a leaf node.
- maximum leaf nodes: Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity.
- class weight: Weights associated with classes in the form class_label: weight.
- bootstrap: Whether bootstrap samples are used when building trees.
- criterion: The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

7.2.7 Adaptive Boosting

```
param_grid = {'vect__ngram_range': [(1, 2)],
              'vect__stop_words': [stop, None],
              'vect__tokenizer': [tokenizer, tokenizer_porter],
              "clf__n_estimators": np.arange(10, 150, 20),
              "clf__learning_rate": np.arange(0.1, 2, 0.1)}
```

Figure 7.8: Adaptive Boosting Parameters Grid Search Code

- ngram range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min\_n <= n <= max\_n$ will be used.
- stop words: A list of words which will be removed from the resulting tokens.
- n_estimators: The number of trees in the forest.

- learning_rate: Learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.

7.3 Fitting Machine Learning Models

After we have found satisfactory hyper parameter values, we could retrain the model on the complete training set and obtain a final performance estimate by using an independent testing set, since fitting a model to the complete training dataset after 10-fold cross-validation would usually result in a more accurate and robust model.

Table 7.1: Section text data (F is F-Measure and Time is Train Time)

	TPR	TNR	Accuracy	Precision	Recall	F	Time(min)
LR	73.4	99.3	91.5	97.9	73.4	83.9	30.2
LR FT	76.6	97.3	91.0	92.5	76.6	83.8	-
SVM	0.0	100.0	69.7	0.0	0.0	0.0	51.5
SVM FT	76.6	96.9	90.8	91.6	76.6	83.4	-
DT	82.8	91.2	88.7	80.3	82.8	81.5	20.1
DT FT	71.9	98.3	90.3	94.8	71.9	81.8	-
NB	29.7	99.7	78.5	97.4	29.7	45.5	123.3
NB FT	70.3	98.0	89.6	93.8	70.3	80.4	-
KNN	75.0	94.2	88.4	85.0	75.0	79.7	85
KNN FT	71.9	95.3	88.2	86.8	71.9	78.6	-
RF	74.2	96.9	90.1	91.3	74.2	81.9	1456.4
RF FT	65.6	99.7	89.4	98.8	65.6	78.9	-
AdaBoost	78.9	94.6	89.8	86.3	78.9	82.4	290.5
AdaBoost FT	72.7	98.6	90.8	95.9	72.7	82.7	-
Majority Voting	75.0	99.0	91.7	97.0	75.0	84.6	-

Table 7.2: Paragraph text data (F is F-Measure and Time is Train Time)

	TPR	TNR	Accuracy	Precision	Recall	F	Time (min)
LR	59.5	96.7	89.3	81.7	59.5	68.8	28.6
LR FT	66.7	95.6	89.9	79.1	66.7	72.4	-
SVM	0.0	100.0	80.1	0.0	0.0	0.0	104.7
SVM FT	69.9	95.8	90.6	80.4	69.9	74.8	-
DT	59.7	90.0	84.0	59.7	59.7	59.7	27.5
DT FT	55.5	95.4	87.5	75.1	55.5	63.8	-
NB	14.7	99.3	82.4	83.3	14.7	24.9	111
NB FT	47.2	97.2	87.2	80.5	47.2	59.5	-
KNN	60.5	95.2	88.3	75.9	60.5	67.4	1556.6
KNN FT	67.7	95.4	89.9	78.6	67.7	72.8	-
RF	44.5	96.8	86.4	77.3	44.5	56.5	1527.4
RF FT	53.3	97.1	88.4	82.0	53.3	64.6	-
AdaBoost	56.3	94.3	86.7	71.0	56.3	62.8	332.4
AdaBoost FT	53.9	97.4	88.7	83.5	53.9	65.5	-
Majority Voting	63.5	96.8	90.1	82.9	63.5	71.9	-

7.4 Performance Evaluation

Table 7.1 and Table 7.2 denote the TPR (True Positive Rate), TNR (True Negative Rate), Accuracy, Precision, Recall, F-Measure and tuning hyper parameter time for LR (Logistic Regression), SVM (Support Vector Machine), DT (Decision Tree), Naive Bayes (NB), K Nearest Neighbors (KNN), Random Forest (RF), Adaboosting (AdaBoost) and Majority Voting models in Section and Paragraph Text Data respectively, where FT stands for fine tunning. The chapter 7 is a task to identify if the users are interested in a particular section in journal papers, while the Paragraph Text Classification is a task to see if the users are interested in a particular paragraph in papers. All except time are measured by percentage. We can see the Majority Voting could take the advantages of the other models and give us the relative best results in all metrics. Since Majority Voting combines all other fine tuning models, we do not need to tune hyper parameters again.

Figure 7.9: Radar Chart A of Models for Section text data

Figure 7.9 to Figure 7.12 show Radar Charts (the charts look like radar signal) for comparing default parameters and fine tuning hyper parameters of LR (Logistic Regression), SVM (Support Vector Machine), DT (Decision Tree), Naive Bayes (NB), K Nearest Neighbors (KNN), Random Forest (RF), Adaboosting (AdaBoost) and Majority Voting models in Section and Paragraph Text Data respectively, where

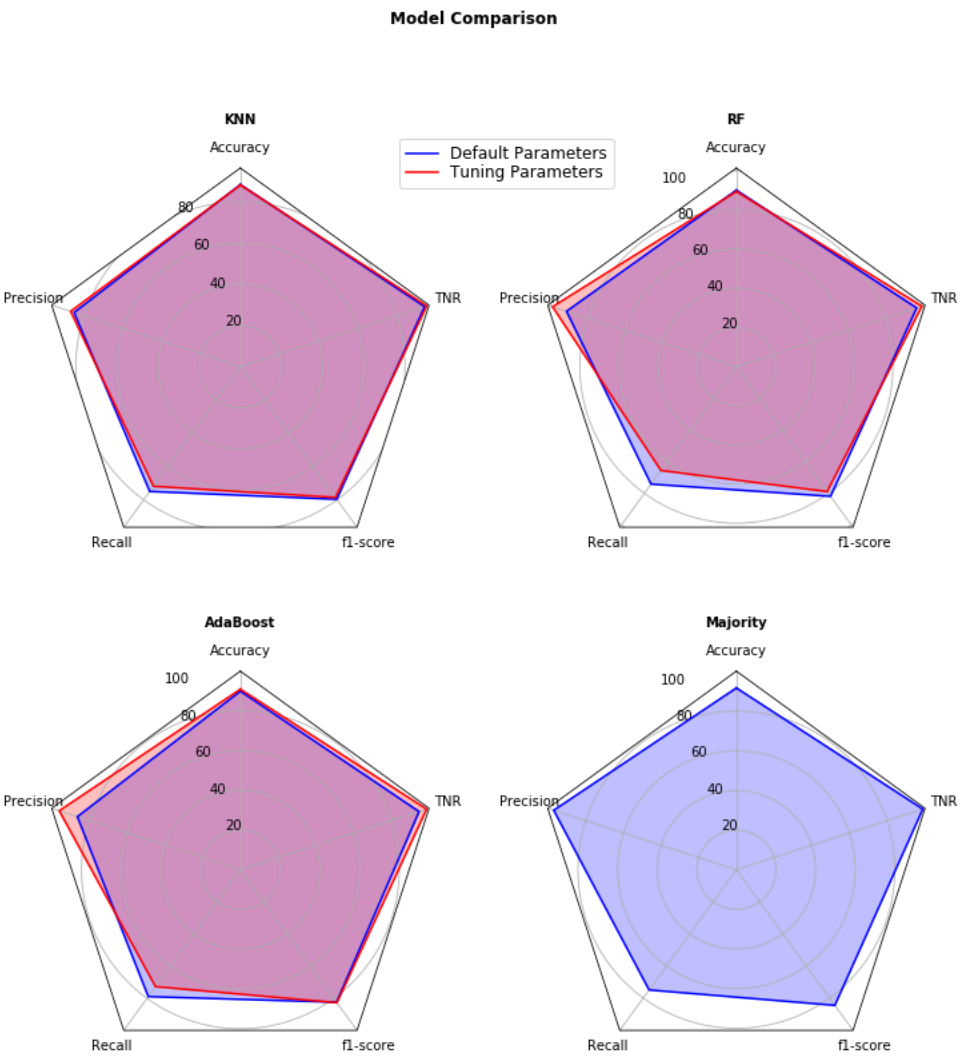


Figure 7.10: Radar Chart B of Models for Section text data

FT stands for fine tunning, the Blue color indicates the default hyper parameters, and the Red color indicates the fine tunning of hyper parameters. Here we choose TNR (True Negative Rate), Accuracy, Precision, Recall, and F-Measure, since TPR (True Positive Rate) is equal to Recall. We could see from these figures that the performance of the model after fine tunning is much better than just using default

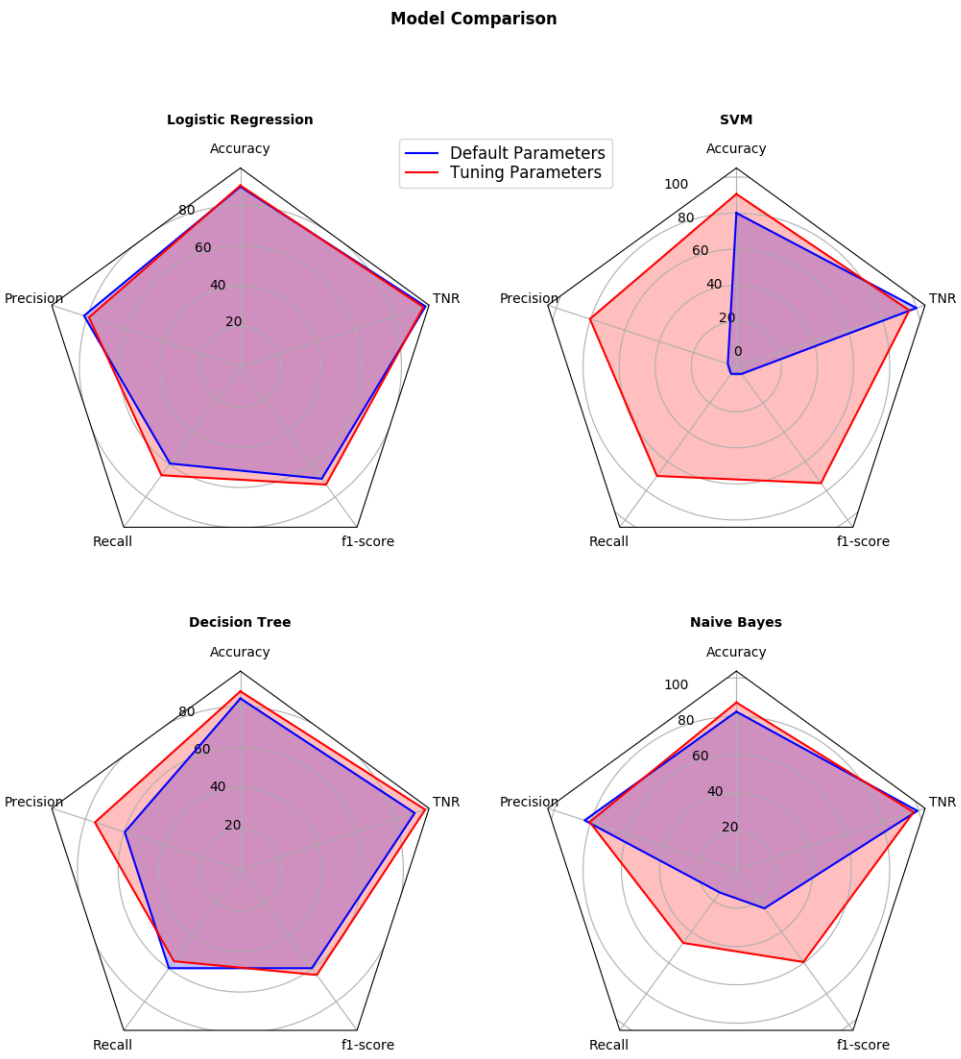


Figure 7.11: Radar Chart A of Models for Paragraph text data

hyper parameters, and the Majority Voting could combine and take the advantages of the other models and give us the relative best results in all metrics. The Radar Charts in Figure 7.9 to Figure 7.12 give a good and clear comparison between the default and fine tuned parameters.

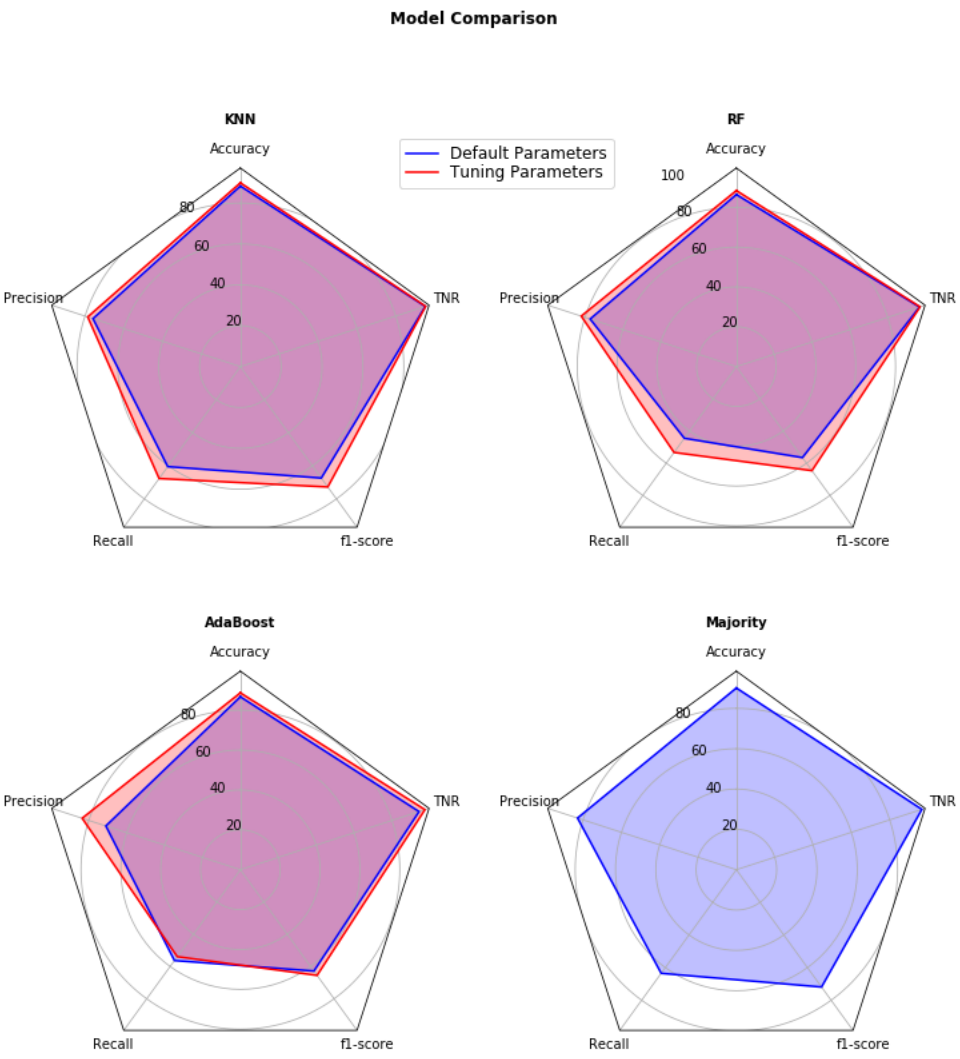


Figure 7.12: Radar Chart B of Models for Paragraph text data

7.5 Conclusion and Discussion

The purpose of Text Classification is that we feed model with an unseen journal paper, and this classifier model could identify if the new paper or part of it is what users are interested in and store the content in the database if needed. We do two experiments there, one for section text classification, and the other paragraph

text classification. Section text classification indicates the classifier models would consider each section in the journal paper as a block and classify it as desirable or not. There are 1690 files in section text classification. Paragraph text classification denotes that the classifier models would consider each paragraph in the journal paper as a block and classify it as desirable or not. There are 7543 files in paragraph text classification.

For section text classification, we have total 1690 files, and 513 of them are content of interest, and the other 1177 are not needed. 30% of them are what users are interested in. By our classifier model, we could classify about 92% of them correctly. For paragraph text classification, we have total 7543 files, and 1498 of them are desired, and the other 6045 are not needed. 20% of them are what users are interested in. By our classifier model, we could classify almost 90% of them correctly. According to our experiment results like F-measure in Table 7.1 and Table 7.2, the less content, the more difficulty to classify correctly. This makes sense since the less information we have, the more difficulty for us to make correct decision.

Table 7.3: Neural Networks Performance Comparison

	TPR	FNP	TP	FP	TP/A	M Size	T Size
Text 1D-CNN	100	94.5	103	222	3.15	36.3 MB	1.2 MB
Text Multi-CNN	100	94.5	103	222	3.15	109.0 MB	1.2 MB
Text LSTM	100	83	103	195	2.89	40.0 MB	1.2 MB
Text Character-D-CNN	100	97	103	228	3.21	1.0 MB	0.0 MB

7.6 Neural Networks

In the Table 7.3, TPR is True Positives Percentage. FNP is False Negatives Percentage. TP is True Positives. FP is False Positives. TP/A is True Positives out of all predicted Positives Percentage. M Size is Model Size. T Size is Tokenizer Size. We can see LSTM achieved the best performance and it shows us that users will get 1 paper which they are interested in given every 2.89 recommendations.

Figure 7.13 shows the accuracy and loss during training and testing and we can see the gap between train and test makes sense and does not trigger overfitting. To build robust model, we need to catch all true positives and reduce false positives. Figure 7.14 shows how we search the optimum cutoff to achieve this goal. The top two plots are for Percentage, while the bottom two are for Counts. They give us a clear tracking during the search. We search twice, the first search window is 0 to 1 which are probabilities of class 1 (interest). Then we narrow the search window and get a preciser cutoff, since a tiny cutoff change can change the model performance a lot as shown in the Figure 7.14.

Figure 7.15 shows we catch all positives and 195 (83%) negatives. This means every 2.89 suggested papers, users can get 1 which they are interested. 2.89 is calculated on (103+195)/103. Because the data is imbalanced, accuracy is not a good metric for model evaluation. Our goal is to make sure all true positives can be identified since we hope the model does not miss any section or paragraph of interest to users, while reduce the false positives since they are sections or paragraphs users are not interested in. To achieve this purpose, we built a custom metric which can catch all true positives and reduce false positives as many as it could.

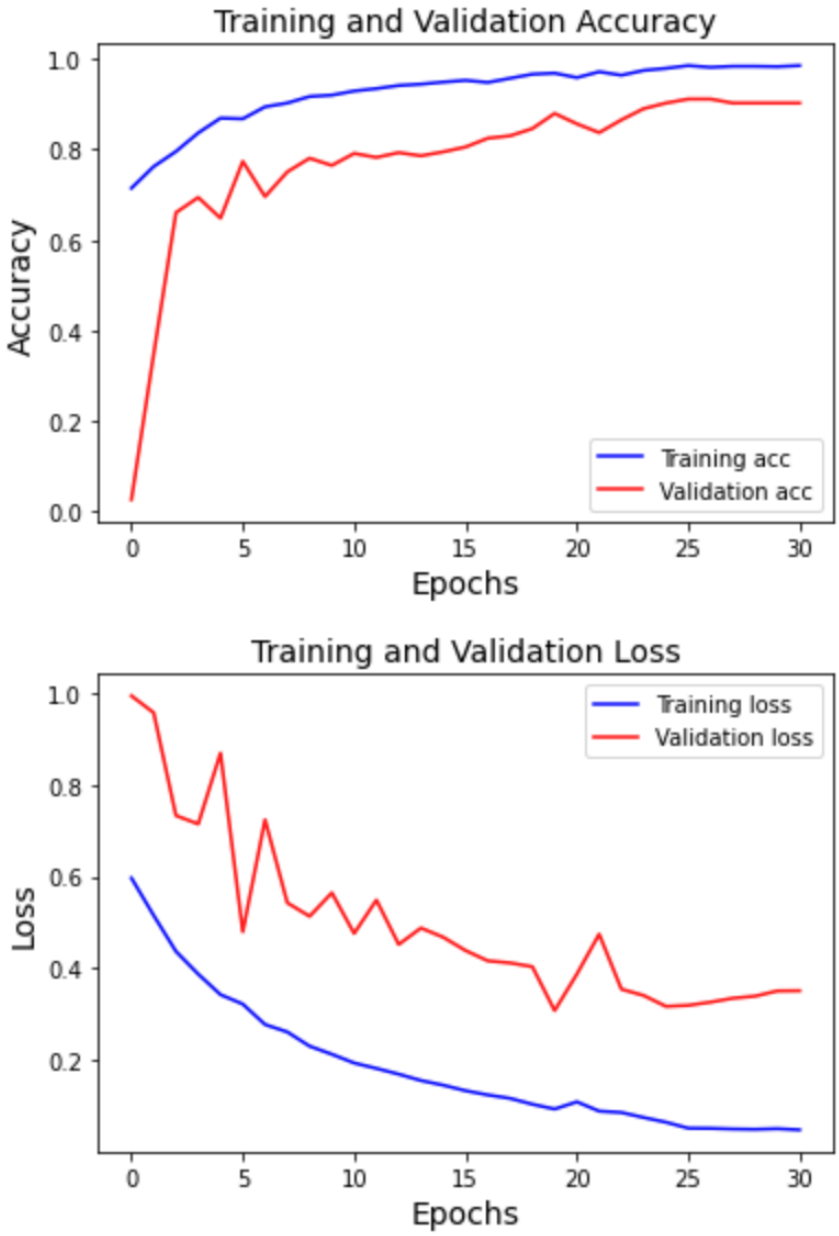


Figure 7.13: LSTM Train/Test Accuracy and Loss

```
1st Round Search:
low: 0, high: 1.0
Find Max Threshold 0.0 to make False Negative = 0
2nd Round Search:
low: 0.0, high: 0.001
val:  8.016032064128256e-05
Find Max Threshold 8.016032064128256e-05 to make False Negative = 0
```

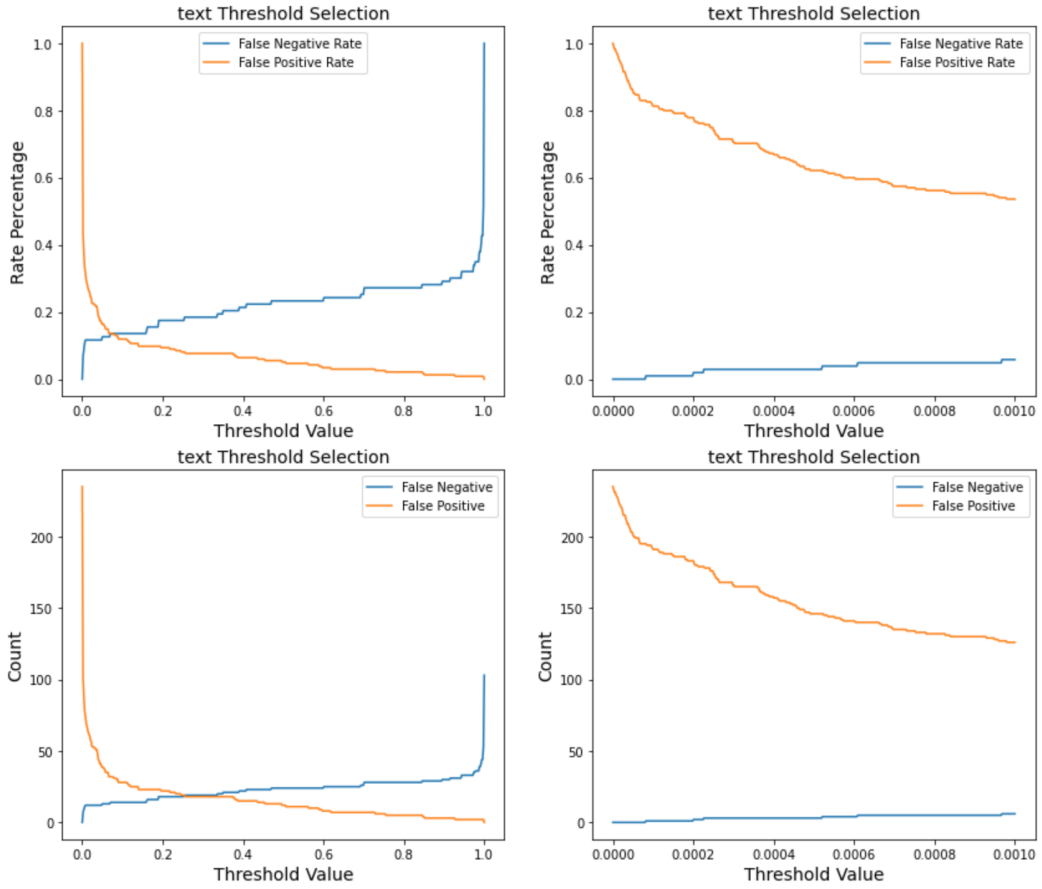


Figure 7.14: Cutoff Search

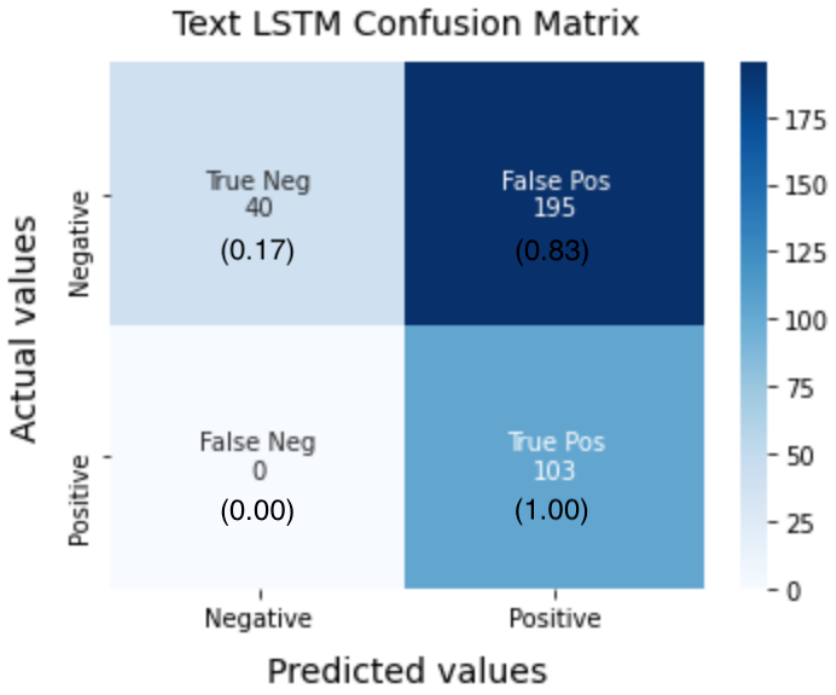


Figure 7.15: LSTM Confusion Matrix

Chapter 8

Named Entity Recognition

Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, locations, geo-political entities, date, percent etc. As shown in Figure 8.1, the purpose of NER is to identify all named entities. We used Stanford NER [23], which is a Java implementation of a Named Entity Recognizer, to identify persons and locations contained in the journal.

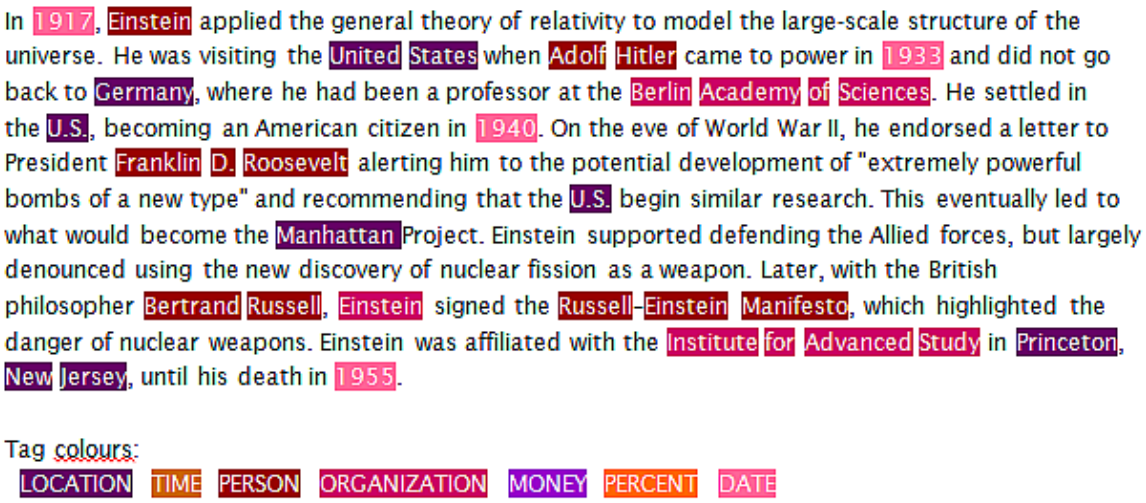


Figure 8.1: Named Entities in journal paper

8.1 Purpose

The reason why we want to extract named entities from journal papers is that we want to select papers by the locations contained in the paper. For example, to be able to query studies that were performed in Lancaster County, Nebraska, so one could view journal papers and results associated with that location as shown in Figure 8.2. Named Entity Recognition can help us make it a reality. It would extract information about references to PERSONs, ORGANIZATIONs, and LOCATIONs from journal papers. The LOCATION will include all locations mentioned in the paper. We also want to extract information about the authors and organizations, and this information is contained in PERSONs, ORGANIZATIONs. As shown in the Figure 8.2, user can select Lincoln as Location and the system will list all papers whose experiments take place at Lincoln, Nebraska.

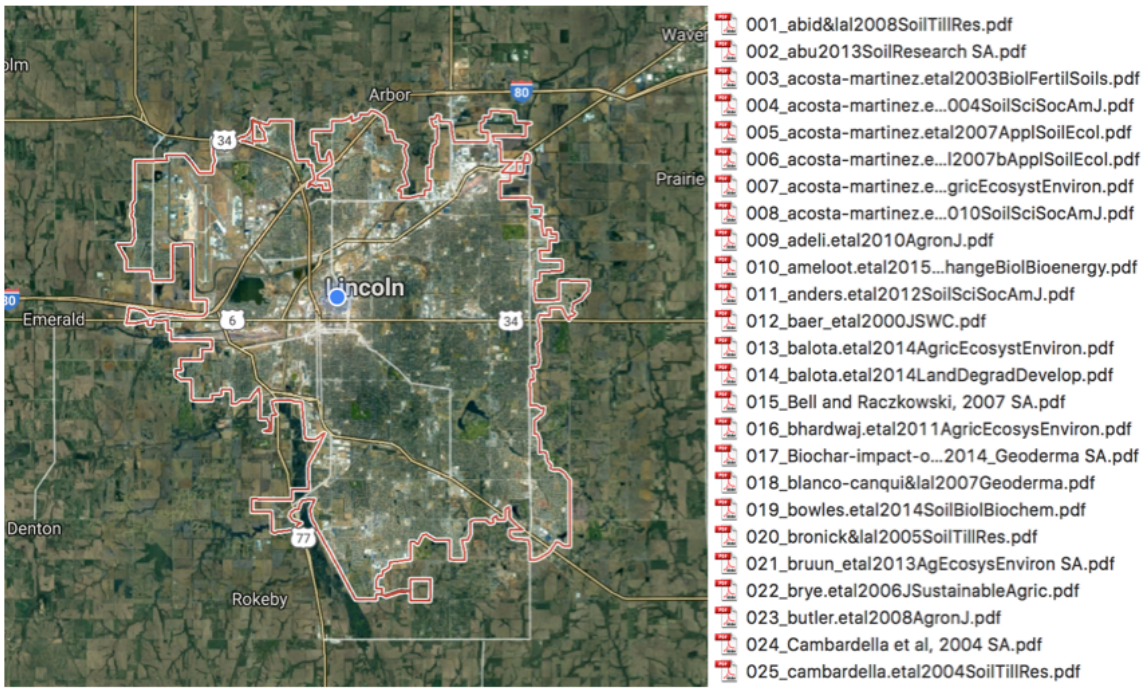


Figure 8.2: Location of Interest and related list of journal papers

8.2 Procedure

While PDFs provide an easily readable presentation of data, they are extremely difficult to work with in data analysis. We used a Pdfminer called "pdf2txt.py" that extracts text contents from a PDF file, and NLTK (the Natural Language Tool Kit) [24] serves as one of Python's leading platforms to analyze natural language data. In Named Entity Recognition, we ignore the references section since it contains too many persons and organizations which are not related to the desired information. We also used Stanford's Named Entity Recognizer [23], often called Stanford NER, to extract PERSONs, LOCATIONs, and ORGANIZAIONs from journal papers and store them in JSON format. NLTK contains an interface to Stanford NER, so all codes are written in Python.

8.3 Result

LOCATION
Coffey Road
Columbus
OH
USA
Multan
Pakistan
Ohio
Columbus
Ohio
Multan
Pakistan
Hanau
Germany
Miami
south Brazil
Islamabad


```
{
   "docName": "p1.txt",
   "ensemble_Location": [
      "Multan",
      "USA",
      "Miami",
      "Islamabad",
      "Ohio",
      "Pakistan",
      "Columbus",
      "Germany",
      "Coffey Road",
      "Hanau"
   ]
}
```


Figure 8.3: Comparison between ground truth and Stanford NER output

On the left side of the Figure 8.3, the column is from hand labeled location file. Data on the right side is the output of the Stanford's Named Entity Recognizer. Among location list on the left side of the Figure 8.3, Stanford's Named Entity

Recognizer can identify 10 items including Multan, USA, Miami, Islamabad, Ohio, Pakistan, Columbus, Germany, Coffey Road and Hanau. The missed items are OH and south Brazil, where OH is abbreviation of Ohio. Stanford's Named Entity Recognizer can identify 10 out of 12, which is about 83%. Based on the precision, recall and F-score formulas, we can get that precision is 1, recall is 0.83 and F-score is about 0.91. In Manning et al.'s paper [23], they claim that F-scores are 0.87 and 0.92 for the CoNLL and CMU Seminar Announcements respectively, where the CoNLL is named entity recognition task and the CMU Seminar Announcements is information extraction task for NER approach evaluation. The NER performance is in between these two scores for our project and it makes sense. It is better if we could use more samples to test the performance of Stanford's Named Entity Recognizer and verify the approach works well in our project. However, it will cost a lot of labeling time and effort and we do not have enough time or labor to do that.

Chapter 9

Table Analysis

Tables contain data of interest of readers. The purpose of table analysis is to convert the content of human-readable tables in the journal paper to query-table in database. We created 1006 tables for 207 journal papers in Comma-Separated Values (CSV) format manually. Then we ran the algorithm of Seth et al. [3] to extract data from CSV and store them in a machine-readable format. There are 1006 tables in the 207 papers. Each table would cost 20-30 minutes, we test 50% of them, so the total time spent on preparing and checking these 500 tables is about 280 hours.

9.1 Program Output

The program can output two kinds of tables. One is a classification table. This table is in a five-column format, with a row entry (after the header row) for each cell of its source table. The first column is a unique cell identifier with the file name of the CSV table and the cell coordinates. The second and third rows give the numerical cell coordinates separately for ease of handling. The fourth column

is the content of the cell in the original table, and the last column is its assigned class.

The other table is a category table which is a relational table where each row comprises the indexing header paths and the corresponding indexed data value. Therefore the number of rows in the category table equals the number of data cells in the original table (plus one for the relational table's field names in a header row). The number of columns is one for the Cell_ID, plus one for DATA, plus the sum of the heights of the category trees (which, usually, equals the sum of the column width of the row header and row height of the column header). In the category table, Cell_ID is a key field and each cell label in the original header paths becomes a key field value in the composite key comprising all the category fields.

9.1.1 Well-Formed Table result

Table 3
Soil enzyme activities following the application of different cover crops at the Kinston and Goldsboro sites.

	Exoglucanase	β-Glucosidase	β-Glucosaminidase	Peroxidase
	μmol h^{-1} g^{-1} soil			
Kinston site				
Austrian winter pea	0.240 a	0.854 *ab*	0.378 ab	
Hairy vetch	0.222 a	0.849 *ab*	0.335 bc	
Crimson clover	0.237 a	0.900 *a*	0.399 a	
No cover crop	0.223 a	0.807 *b*	0.309 c	
Goldsboro site				
Austrian winter pea	0.226 a	1.379 a	0.403 a	0.581 a
Hairy vetch	0.218 ab	1.332 a	0.400 a	0.640 a
Crimson clover	0.226 a	1.402 a	0.412 a	0.632 a
No cover crop	0.202 b	1.191 b	0.366 b	0.622 a

Different letters in a column indicate significant differences at $P < 0.05$. Different italic letters are also used to indicate significant differences at $P < 0.1$.

Figure 9.1: Table 3 in journal paper No.105

ACCESS_105_liang

ID	Field1	Field2	Field3	Field4	Field5
1	Cell_ID	RowCat_1.1	RowCat_1.2	ColCat_1.1	DATA
2	105_liang.etal2014EurJSoilBiol_Table_3_R3_C2	Soil enzyme ac	BLANC	Exoglucanase	mmolh 1g
3	105_liang.etal2014EurJSoilBiol_Table_3_R3_C3	Soil enzyme ac	BLANC	b-Glucosidase	
4	105_liang.etal2014EurJSoilBiol_Table_3_R3_C4	Soil enzyme ac	BLANC	b-Glucosamini	
5	105_liang.etal2014EurJSoilBiol_Table_3_R3_C5	Soil enzyme ac	BLANC	Peroxidase	
6	105_liang.etal2014EurJSoilBiol_Table_3_R4_C2	Kinston site	ditto	Exoglucanase	
7	105_liang.etal2014EurJSoilBiol_Table_3_R4_C3	Kinston site	ditto	b-Glucosidase	
8	105_liang.etal2014EurJSoilBiol_Table_3_R4_C4	Kinston site	ditto	b-Glucosamini	
9	105_liang.etal2014EurJSoilBiol_Table_3_R4_C5	Kinston site	ditto	Peroxidase	
10	105_liang.etal2014EurJSoilBiol_Table_3_R5_C2	Kinston site	Austrian winte	Exoglucanase	0.240a
11	105_liang.etal2014EurJSoilBiol_Table_3_R5_C3	Kinston site	Austrian winte	b-Glucosidase	0.854ab
12	105_liang.etal2014EurJSoilBiol_Table_3_R5_C4	Kinston site	Austrian winte	b-Glucosamini	0.378ab
13	105_liang.etal2014EurJSoilBiol_Table_3_R5_C5	Kinston site	Austrian winte	Peroxidase	
14	105_liang.etal2014EurJSoilBiol_Table_3_R6_C2	Kinston site	pea	Exoglucanase	
15	105_liang.etal2014EurJSoilBiol_Table_3_R6_C3	Kinston site	pea	b-Glucosidase	
16	105_liang.etal2014EurJSoilBiol_Table_3_R6_C4	Kinston site	pea	b-Glucosamini	
17	105_liang.etal2014EurJSoilBiol_Table_3_R6_C5	Kinston site	pea	Peroxidase	
18	105_liang.etal2014EurJSoilBiol_Table_3_R7_C2	Kinston site	Hairy vetch	Exoglucanase	0.222a
19	105_liang.etal2014EurJSoilBiol_Table_3_R7_C3	Kinston site	Hairy vetch	b-Glucosidase	0.849ab
20	105_liang.etal2014EurJSoilBiol_Table_3_R7_C4	Kinston site	Hairy vetch	b-Glucosamini	0.335bc
21	105_liang.etal2014EurJSoilBiol_Table_3_R7_C5	Kinston site	Hairy vetch	Peroxidase	
22	105_liang.etal2014EurJSoilBiol_Table_3_R8_C2	Kinston site	Crimson clover	Exoglucanase	0.237a
23	105_liang.etal2014EurJSoilBiol_Table_3_R8_C3	Kinston site	Crimson clover	b-Glucosidase	0.900a
24	105_liang.etal2014EurJSoilBiol_Table_3_R8_C4	Kinston site	Crimson clover	b-Glucosamini	0.399a
25	105_liang.etal2014EurJSoilBiol_Table_3_R8_C5	Kinston site	Crimson clover	Peroxidase	

Figure 9.2: Category Table (already imported to Microsoft Access Database) for Table 3 in Journal Paper No. 105

As shown in Figure 9.2, we could see the program could extract the information from Well-Formed Table in Figure 9.1 and store it in relational database (Microsoft Access). Then users could use SQL to query the information they are interested in. And the classification table also make sense.

9.1.2 NOT Well Formed Table result

There are also some NOT Well Formed Tables as shown in Figure 9.3 where the program does not work well.

We could see in the classification table as shown in Figure 9.4, the program fails to classify the data at the bottom half table correctly, and classify the data as note. So the category table only contains the data from top half of the original table, but miss the data in bottom half.

Table 2
Tillage and drainage effects on C:N ratio

Tillage systems	0–10 cm[a]		10–20 cm[a]		20–30 cm[a]	
	UN[b]	D[b]	UN[b]	D[b]	UN[b]	D[b]
NT	10.41	10.26	10.92	10.32	10.49	10.23
T	10.74	10.98	10.24	10.28	10.50	10.37
Least significant difference (LSD)						
Tillage	0.41 (*)		0.95 (ns)		1.07 (ns)	
Drainage	0.41 (ns)		0.95 (ns)		1.07 (ns)	
T × D	0.41 (ns)		0.95 (ns)		1.07 (ns)	

NT: no-till; T: tillage (chisel plough); T × D: interactive effect of tillage and drainage; UD: un-drained; D: drained; symbol (*) and/or (ns) is *P*-value; ns: not significant; *significant at $P \leq 0.05$.

[a] Depth.

[b] Drainage type.

Figure 9.3: Table 2 in Journal Paper No. 001

9.1.3 Discussion

We have a total of 500 tables in 122 journal papers. 99 of them are not well-formed tables, which means we could not or maybe just could partly extract data from the table and store them in relational database. After manually creating different kinds of tables in the journal papers, we found the program could process about 90% of all tables, which are well-formed tables. For the other 10% not well-formed tables, the program can not extract correct information.

001_abid&la	7	1	Tillage	note
001_abid&la	7	2	0.41 (*)	note
001_abid&la	7	3		note
001_abid&la	7	4	0.95 (ns)	note
001_abid&la	7	5		note
001_abid&la	7	6	1.07 (ns)	note
001_abid&la	7	7		note
001_abid&la	8	1	Drainage	note
001_abid&la	8	2	0.41 (ns)	note
001_abid&la	8	3		note
001_abid&la	8	4	0.95 (ns)	note
001_abid&la	8	5		note
001_abid&la	8	6	1.07 (ns)	note
001_abid&la	8	7		note
001_abid&la	9	1	T D	note
001_abid&la	9	2	0.41 (ns)	note
001_abid&la	9	3		note
001_abid&la	9	4	0.95 (ns)	note
001_abid&la	9	5		note
001_abid&la	9	6	1.07 (ns)	note
001_abid&la	9	7		note

Figure 9.4: Classification Table for Table 2 in Journal Paper No. 001

Chapter 10

Database System

We used Stanford NER to extract first five authors, city and state from papers, wrote code to extract Title, Publication Date, Abstract, Journal, DOI and Type from papers, and count if keywords defined by soil scientist appear in the paper. Then we used the algorithm of Seth et al. [3] to convert the well-formed table (manually created by myself) to tables. Finally, we inserted all information mentioned above into Microsoft Access Database.

10.1 Database Summary

We stored the journal paper information in the Microsoft Access Database instead of personal designed database system. The details of the system is shown in the Figures 10.1 to 10.3. Table 10.1 indicates and describes columns stored in database shown in Figure 10.1. And Table 10.2 indicates and describes columns stored in database shown in Figures 10.2 and 10.3, where keywords are defined by soil scientist and they are used to count the occurrences of terms that soil scientists are interested in, including Conservation, No Tillage, Ridge Tillage, Mulch Tillage,

Strip Tillage, Reduced Till etc. in Keyword 1 list and Germanium, Gold, Hafnium, Hassium etc. in Keyword 2 list.

Journal

ID	Title	Author1	Author2	Author3	Author4	Author5	Date/Year	City	State	Abstract	Journal	DOI	Type
1	The Amount ar	Amos Feigin	Georgia Shear	Daniel H. Kohl	Barry Commo		1974	Decatur	Georgia	Abstract This p	sssaj	doi:10.2136/sss	Journal
2	Relationship b	B. L. Tillman	S. A. Harrison	J. S. Russin	C. A. Clark		1996	Baton Rouge	Louisiana	Abstract Disea	cropsci	doi:10.2135/cr	Journal
3	Crop Effects or	S. B. Milligan	K. A. Gravois	K. P. Bischoff	F. A. Martin		1990			Abstract Estim	cropsci	doi:10.2135/cr	Journal
4	Increased Soyb	Larry E. Williar	Donald A. Phil				1983	Baton Rouge	Louisiana	Abstract A mu	cropsci	doi:10.2135/cr	Journal
5	Chemical Effec	Catherine The	Kenneth M. H	J. D. Rhoades	Garrison Spos		1990	Riverside	Ohio	Abstract Repre	jeq	doi:10.2134/je	Journal
6	Tall Fescue Res	J. L. Moyer	D. W. Sweene				1990			Abstract Sever	sssaj	doi:10.2136/sss	Journal
7	Chlortetracycli	A. R. Batchelde					1981	Fort Collins	Colorado	Abstract Cattle	jeq	doi:10.2134/je	Journal
8	Nitrite Fixatior	K. A. Thorn *a	M. A. Mikitab				2000	Joliet	Illinois	Studies have s	sssaj	doi:10.2136/sss	Journal
9	Combining Abi	J. B. Holland	M. M. Goodm				1995	Ames	Iowa	Abstract To su	cropsci	doi:10.2135/cr	Journal
10	Earthworm Effe	J. E. Zachmann	D. R. Linden				1989			Abstract Earth	sssaj	doi:10.2136/sss	Journal
11	Simulation of F	Y. Ouyang *a	D. Shindeb	L. Q. Mab			2004			Knowledge of	jeq	doi:10.2134/je	Journal
12	Studying Orgar	Charisma Latta	Justin Birdwel	Jim J. Wangc	Robert L. Cool		2007	Baton Rouge	Louisiana	This work sho	jeq	doi:10.2134/je	Journal
13	Evaluation of F	Philip J. Bauer	Cynthia C. Gre				1996	Norfolk	Nebraska	Abstract Reduc	cropsci	doi:10.2135/cr	Journal
14	Modeling Varia	Günter Langer	Jirka Šimůnek				2004	Riverside	Ohio	Constructed w	vzj	doi:10.2136/vz	Journal
15	A Kinetic Meth	J. W. Odom	P. F. Low2				1983	Cerritos	California	Abstract A me	sssaj	doi:10.2136/sss	Journal
16	Influence of Da	L. J. Chena	L. Xingb	L. J. Han *a			2009	Boston	Massachusetts	With increasin	jeq	doi:10.2134/je	Journal
17	Genetic Analys	Shree P. Singh	P. N. Drolsom				1977			Abstract Four (	cropsci	doi:10.2135/cr	Journal
18	Pore-Water Ex	M. Oostrom a	M.J. Truexa	T.W. Wietsma	G.D. Tartakov		2014	Richland	Washington	Vacuum-induc	vzj	doi:10.2136/vz	Journal
19	Effects of Lime	E.M. Thaysen a	S. Jessenb	D. Postmac	R. Jakobsenc	D. Jacquesd	2014	Portland	Oregon	In this work we	vzj	doi:10.2136/vz	Journal
20	Broadening the	Shelagh P. Kel	Brian V. Ford-	Joana Magos E	José M. Iriond	Nigel Maxted	2016	Birmingham	Alabama	A broad definit	cropsci	doi:10.2135/cr	Journal
21	Hybrid Quality	Greg W. Roth*					1994	Rock Springs	Wyoming	Differences an	jpa	doi:10.2134/jp	Journal
22	Selection for D	M. Bänziger	G. O. Edmead	H. R. Lafitte			1999			Abstract It is n	cropsci	doi:10.2135/cr	Journal
23	Temporal Resp	Alan J. Sexstor	Timothy B. Pa	James M. Tiec			1985	East Lansing	Michigan	Abstract The re	sssaj	doi:10.2136/sss	Journal
24	Influence of Fa	H. V. Welch Jr.	A. Wallace	R. T. Mueller2			1954	Los Angeles	California	Abstract The in	sssaj	doi:10.2136/sss	Journal
25	Nitrogen Dynai	Paul M. Mayer	Peter M. Grof	Elise A. Strizac	Sujay S. Kaush		2009	Baltimore	Maryland	Few studies ha	jeq	doi:10.2134/je	Journal
26	Bromacil in Lak	Edwin A. Hebt	Willis B. Whe				1978	Lakeland	Florida	Abstract The o	jeq	doi:10.2134/je	Journal
27	A Self-Adjustir	Ralph A. Leona	Philip F. Low2				1962	Knoxville	Tennessee	Abstract This p	sssaj	doi:10.2136/sss	Journal
28	Effect of Black	Hasan K. Qash	D. D. Evans2				1967	Tucson	Arizona	Abstract The u	sssaj	doi:10.2136/sss	Journal
29	Selenate and S	Chunming Su *	Donald L. Suar				2000	Long Island Cit	New York	We studied se	sssaj	doi:10.2136/sss	Journal
30	Use of Acetyle	O. Q. Hendrick					1988	Ontario	California	Abstract Both (	sssaj	doi:10.2136/sss	Journal
31	Hydraulic Prop	C. H. M. van Ba	K. J. Brust	G. B. Stirk2			1968	Adelanto	California	Abstract Soil w	sssaj	doi:10.2136/sss	Journal
32	Effects of Nitrc	D. H. Van Lear					1980	Athens	Georgia	Abstract Chan	sssaj	doi:10.2136/sss	Journal
33	Heavy Metal Cc	W. Nelson Bey	Rufus L. Chan	Bernard M. Mc			1982	Beltsville	Maryland	Abstract Metal	jeq	doi:10.2134/je	Journal
34	Persistence of	A. J. VandenBy	B. D. Kayb				2004	Ottawa	Illinois	There is abunc	sssaj	doi:10.2136/sss	Journal
35	Late-Season M	R. D. Horrocks'	Mojtaba Zaifn				1997	Spanish Fork	Utah	Yield and stan	jpa	doi:10.2134/jp	Journal
36	Effect of Gypsu	A. K. Sharma	J. B. Fehrenba	B. A. Jones Jr.			1974	Newton	Kansas	Abstract Gypsu	sssaj	doi:10.2136/sss	Journal
37	Quantitative D	R. S. Loomis	W. A. William	W. G. Duncan	A. Dovrat	F. Nunez A.2	1968	Davis	California	Abstract Leaf c	cropsci	doi:10.2135/cr	Journal
38	MIAB10: A Trit	J. J. Maxwella	J. H. Lyerlyb	G. Srnicc	R. Parksd	C. Cowgerd	2010			Powdery mild	cropsci	doi:10.2135/cr	Journal
39	Depth of Surfa	A. N. Sharpley					1985	Stillwater	Oklahoma	Abstract The e	sssaj	doi:10.2136/sss	Journal
40	Humic Substan	Y. Inbar	Y. Chen	Y. Hadar			1990	Philadelphia	Pennsylvania	Abstract Humi	sssaj	doi:10.2136/sss	Journal

Record: 15 of 40845 | No Filter | Search

Figure 10.1: Journal in Database

Keyword_1

ID	Title	Conservatio	No Tillage	Ridge Tillage	Mulch Tillag	Strip Tillage	Reduced Till	Deep Rippin	Conventiona	Chisel tillage	Aggregate St
1	The Amount ar	0	0	0	0	0	0	0	0	0	0
2	Relationship b	0	0	0	0	0	0	0	0	0	0
3	Crop Effects or	0	0	0	0	0	0	0	0	0	0
4	Increased Soyb	0	0	0	0	0	0	0	0	0	0
5	Chemical Effec	0	0	0	0	0	0	0	0	0	0
7	Chlortetracycli	0	0	0	0	0	0	0	0	0	0
8	Nitrite Fixatior	0	0	0	0	0	0	0	0	0	0
9	Combining Abi	0	0	0	0	0	0	0	0	0	0
10	Earthworm Effe	0	0	0	0	0	0	0	0	0	0
11	Simulation of P	0	0	0	0	0	0	0	0	0	0
12	Studying Orgar	0	0	0	0	0	0	0	0	0	0
13	Evaluation of F	1	0	0	0	0	0	0	1	0	0
14	Modeling Varia	0	0	0	0	0	0	0	0	0	1
15	A Kinetic Meth	0	0	0	0	0	0	0	0	0	0
16	Influence of Da	0	0	0	0	0	0	0	0	0	0
17	Genetic Analys	0	0	0	0	0	0	0	0	0	0
18	Pore-Water Ext	0	0	0	0	0	0	0	0	0	1
19	Effects of Lime	0	0	0	0	0	1	0	0	0	0
20	Broadening the	0	0	0	0	0	0	0	0	0	0
21	Hybrid Quality	0	0	0	0	0	0	0	0	0	0
22	Selection for D	0	0	0	0	0	0	0	0	0	0
23	Temporal Resp	0	0	0	0	0	0	0	0	0	0
24	Influence of Fa	0	0	0	0	0	0	0	0	0	0
25	Nitrogen Dynar	0	0	0	0	0	0	0	0	0	0
26	Bromacil in Lak	0	0	0	0	0	0	0	0	0	0
27	A Self-Adjustin	0	0	0	0	0	0	0	0	0	0
28	Effect of Black	0	0	0	0	0	0	0	0	0	0
29	Selenate and S	0	0	0	0	0	0	0	0	0	0
30	Use of Acetyler	0	0	0	0	0	0	0	0	0	0
31	Hydraulic Prop	0	0	0	0	0	0	0	0	0	0
32	Effects of Nitrc	0	0	0	0	0	0	0	0	0	0
33	Heavy Metal Co	0	0	0	0	0	0	0	0	0	0
34	Persistence of	1	0	0	0	0	0	0	1	0	1
35	Late-Season M	0	0	0	0	0	0	0	0	0	0
36	Effect of Gypsu	0	0	0	0	0	0	0	0	0	0
37	Quantitative D	0	0	0	0	0	0	0	0	0	0
38	MlAB10: A Triti	0	0	0	0	0	0	0	0	0	0
39	Depth of Surfa	0	0	0	0	0	0	0	0	0	1
40	Humic Substan	0	0	0	0	0	0	0	0	0	0
41	In Situ Monitor	0	0	0	0	0	0	0	0	0	1

Record: 14 of 38926 No Filter Search

Figure 10.2: Keyword 1 in Database

Keyword_2

ID	Title	Germanium	Gold	Hafnium	Hassium	Holmium	Indium	Iridium	Iron	Lanthanum	Lawrencium
1	The Amount ar	0	0	0	0	0	0	0	0	0	0
2	Relationship b	0	0	0	0	0	0	0	0	0	0
3	Crop Effects or	0	0	0	0	0	0	0	0	0	0
4	Increased Soyb	0	0	0	0	0	0	0	0	0	0
5	Chemical Effec	0	0	0	0	0	0	0	0	0	0
7	Chlortetracycli	0	0	0	0	0	0	0	0	0	0
8	Nitrite Fixatior	0	1	0	0	0	0	0	0	0	0
9	Combining Abi	0	0	0	0	0	0	0	0	0	0
10	Earthworm Effe	0	0	0	0	0	0	0	0	0	0
11	Simulation of P	0	0	0	0	0	0	0	0	0	0
12	Studying Orgar	0	1	0	0	0	0	0	0	0	0
13	Evaluation of F	0	0	0	0	0	0	0	0	0	0
14	Modeling Varia	0	1	0	0	0	0	0	0	0	0
15	A Kinetic Meth	0	0	0	0	0	0	0	0	0	0
16	Influence of Da	0	0	0	0	0	0	0	0	0	0
17	Genetic Analys	0	0	0	0	0	0	0	0	0	0
18	Pore-Water Ext	0	0	0	0	0	0	0	0	0	0
19	Effects of Lime	0	0	0	0	0	0	0	0	0	0
20	Broadening the	0	0	0	0	0	0	0	0	0	0
21	Hybrid Quality	0	0	0	0	0	0	0	0	0	0
22	Selection for D	0	0	0	0	0	0	0	0	0	0
23	Temporal Resp	0	0	0	0	0	0	0	0	0	0
24	Influence of Fa	0	0	0	0	0	0	0	0	0	0
25	Nitrogen Dynai	0	1	0	0	0	0	0	0	0	0
26	Bromacil In Lak	0	0	0	0	0	0	0	0	0	0
27	A Self-Adjustir	0	0	0	0	0	0	0	0	0	0
28	Effect of Black	0	0	0	0	0	0	0	0	0	0
29	Selenate and S	0	1	0	0	0	0	0	1	0	0
30	Use of Acetyle	0	0	0	0	0	0	0	0	0	0
31	Hydraulic Prop	0	0	0	0	0	0	0	0	0	0
32	Effects of Nitrc	0	0	0	0	0	0	0	0	0	0
33	Heavy Metal Co	0	0	0	0	0	0	0	0	0	0
34	Persistence of	0	0	0	0	0	0	0	0	0	0
35	Late-Season M	0	0	0	0	0	0	0	0	0	0
36	Effect of Gypsu	0	0	0	0	0	0	0	0	0	0
37	Quantitative D	0	0	0	0	0	0	0	0	0	0
38	MIAB10: A Triti	0	0	0	0	0	0	0	0	0	0
39	Depth of Surfa	0	0	0	0	0	0	0	0	0	0
40	Humic Substan	0	1	0	0	0	0	0	0	0	0
41	In Situ Monitor	0	0	0	0	0	0	0	0	0	0

Record: 17 of 38926 No Filter Search

Figure 10.3: Keyword 2 in Database

Table 10.1: Journal in Database

Field Name	Description
ID	Unique identifier assigned to each journal paper
Title	Title of journal paper
Journal	Journal Name
Year	Publication Year
Author1	First author of the journal paper
Author2	Second author of the journal paper
Author3	Third author of the journal paper
Author4	Forth author of the journal paper
Author5	Fifth author of the journal paper
State	Experiment State in U.S.
City	Experiment City in U.S.
Abstract	Abstract of the journal paper
DOI	
Type	Journal or Book

Table 10.2: Keyword 1 and Keyword 2 in Database

Field Name	Description
ID	Unique identifier assigned to each journal paper
Title	Title of journal paper
Variable Name	Variables count, where 1 indicates variable contained in the journal paper, and 0 indicates variable not contained in the journal paper

Chapter 11

Conclusion

My work consists of five parts: Web harvesting, Text Classification, Table Analysis, Named Entity Recognition and Database System Build. We built a web crawler to download soil science journal papers and machine learning based system. After converting downloaded journal papers from PDF to TXT format, I used text classification to identify the section or paragraph in a paper that may be of interest to users based on their own search interest. Then I used Named Entity Recognition to extract author and experiment location from paper to store them in data system. I followed Seth et al.'s well formed table requirement and create tables in CSV format manually and used his program to store the table in a queryable machine readable format. Finally, I populated a relational database with information automatically extracted from journal papers collected from internet resources. The users can query and search key words or highlight some section or paragraph in the paper, then the system will provide the relevant information or recommend a paper which is of interest to users. This system will make future queries more efficient.

Chapter 12

Future Work

The future work is designed to perform comprehensive literature reviews for scientists at any stage in a user-friendly way [1]. It will permit the user to filter and search thousands of scientific articles using a simple user interface. In the interface, the user can then quickly browse the sentences or paragraphs with detected keywords or search history, open the full-text article, when required, and convert tables conveniently from PDF files to Excel sheets, and store in the queryable database. The potential work we can do to allow the user-friendly, efficient, and automated extraction of meta-data from full-text articles, which can aid in summarizing the existing literature on any topic of interest are as follows.

12.1 Text Extraction

PDF (Portable Document Format) files are a type of files developed by Adobe in order to enable the creation of various forms of content. Particularly, it allows a consistent safety regarding the change in its content. A PDF file can host different types of data: text, images, media, etc. It is a tag-structured file which makes it

easy to parse it just like an HTML page. However, working with PDFs is difficult due to the extreme flexibility given by the PDF format. The main problem is that PDF was never really designed as a data input format, but rather, it was designed as an output format giving fine grained control over the resulting document. At its core, the PDF format consists of a stream of instructions describing how to draw on a page. In particular, text data isn't stored as paragraphs, or even words, but as characters which are painted at certain locations on the page. As a result, most of the content semantics are lost when a text or word document is converted to PDF - all the implied text structure is converted into an almost amorphous soup of characters floating on pages. Our mission was particularly difficult as we had to process PDF documents coming from a variety of sources, with wildly different styling, typesetting and presentation choices. Sometimes PDFs include extra spaces between letters in a word. Reconstructing the original text is a difficult problem to solve generally. In the future work, we could try to apply OCR techniques and grouping or clustering algorithm which compares letter sizes, positions and alignments in order to determine what is a word/paragraph [30].

12.2 Table Extraction

Table extraction from PDF and image documents is a ubiquitous task in the real-world. Perfect extraction quality is difficult to achieve with one single out-of-box model due to the wide variety of table styles, and the lack of training data representing this variety. Meanwhile, building customized models from scratch can be difficult due to the expensive nature of annotating table data.

In addition to the high-througput evaluation and categorization of scientific articles, the conversion of tables from PDF files into processable file formats such

as comma- or tab-separated values files, i.e., *.csv or *.tsv, is often a tedious but integral part of literature reviews. While many tools allow the fast and fairly accurate conversion of PDF files into Excel files, they often require a paid subscription for the processing of more than a few files (e.g., https://smallpdf.com/, https://www.adobe.com/acrobat/online/pdf-to-excel.html, and https://pdftables.com/), are limited in input file numbers and therefore do not allow high-throughput table extraction (e.g., https://pdftoxls.com/, https://docs.zone/pdf-to-excel, and https://www.pdftoexcel.com/), lose table headings and footnotes (e.g., tabulizer R package, and https://pdftoxls.com/) or require manual selection of tables in a file and adjustment of the table format information (e.g., pdf_text and pdf_data from pdftools R package which only extract the PDF file text or positional information of words (e.g., Microsoft Excel).

For our project, we can try to use computer vision and machine learning techniques to locate tables in PDF's with a better recall than existing approach. The potentail approach consisted of three main steps: 1. Use OCR contour analysis to identify the characters in the image. 2. Run k-means on the locations of the characters to generate bounding boxes that might contain a table. 3.Identify the bounding boxes that contain tables using a CNN.

12.3 Algorithm

Deep learning neural networks are nonlinear methods. They offer increased flexibility and can scale in proportion to the amount of training data available. A downside of this flexibility is that they learn via a stochastic training algorithm which means that they are sensitive to the specifics of the training data and may find a different set of weights each time they are trained, which in turn produce

different predictions. Generally, this is referred to as neural networks having a high variance and it can be frustrating when trying to develop a final model to use for making predictions. A successful approach to reducing the variance of neural network models is to train multiple models instead of a single model and to combine the predictions from these models. This is called ensemble learning and not only reduces the variance of predictions but also can improve prediction performance that are better than any single model. Techniques for ensemble learning can be grouped by the element that is varied, such as training data, the model, and how predictions are combined.

12.4 Software Engineering

We also plan to build a simple user interface and set up a process allowing the extraction of tables and text easier instead of automating PDF data extraction, fetching data, building model, recommendation and search, and creating machine-readable data separately. That will also enhance user experience even without any technical background.

Bibliography

[1] Erik Stricker and Michael Scheurer. "PDF Data Extractor (PDE) - A Free Web Application and R Package Allowing the Extraction of Tables from Portable Document Format (PDF) Files and High-Throughput Keyword Searches of Full-Text Articles". *bioRxiv*, 2021. 1, 12

[2] EMarshall IJ and Wallace BC. "Toward systematic review automation: a practical guide to using machine learning tools in research synthesis". *Syst Rev.*, 2019. 1

[3] George Nagy, David W. Embley, Mukkai Krishnamoorthy, and Sharad Seth. "Clustering header categories extracted from web tables". *Proceedings of SPIE - The International Society for Optical Engineering*, 2015. 1, 3.3, 9, 10

[4] Tom Mitchell. "Machine Learning". *McGraw Hill*, 1997. 2

[5] Rish. "An empirical study of the naive bayes classifier". In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, pages 41–46, 2001. 2.1.4

[6] P. Domingos and M. Pazzani. "On the optimality of the simple bayesian classifier under zero-one loss". *Machine learning*, 29(2–3):103–130, 1997. 2.1.4

[7] Zainab Ghadiri Modarres, Mahmood Shabankhah, and Ali Kamandi. "Making AdaBoost Less Prone to Overfitting On Noisy Datasets". *EasyChair Preprint no. 2742*, 2020. 2.2

[8] David H. Wolpert. "Stacked Generalization". *Neural Networks*, 5(2):241–259, 1992. 2.2.1

[9] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". *Journal of Computer and System Sciences*, 55:119–139, 1997. 2.2.3

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient Estimation of Word Representations in Vector Space". *ICLR (Workshop Poster)*, 2013. 2.3.1

[11] Jason Brownlee. How Do Convolutional Layers Work in Deep Learning Neural Networks? https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/, 2019. [Online]. 2.3.2

[12] Jason Brownlee. Deep Convolutional Neural Network for Sentiment Analysis (Text Classification). https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/, 2017. [Online]. 2.3.2

[13] Jason Brownlee. How to Develop a Multichannel CNN Model for Text Classification. https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/, 2018. [Online]. 2.3.3

[14] Jason Brownlee. Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras. https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/, 2016. [Online]. 2.3.4

[15] Scikit-Learn. Working With Text Data. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html. [Online]. 2.5.2

[16] Gao J., Pantel P., and Deng L. "Modeling Interestingness with Deep Neural Networks". *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics*, 2014. 3.1

[17] Shen Y., He X., and Mesnil G. "A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval". *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management – CIKM '14*, page 101–110, 2014. 3.1

[18] Santos C. and Zadrozny B. "Learning Character-level Representations for Part-of-Speech Tagging". *Proceedings of the 31st International Conference on Machine Learning, ICML-14*, page 1818–1826, 2014. 3.1

[19] Zhang X., Zhao J., and LeCun Y. "Character-level Convolutional Networks for Text Classification". *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, page 1–9, 2015. 3.1

[20] Zhang X. and LeCun Y. "Text Understanding from Scratch". *arXiv E-Prints*, 2015. 3.1

[21] Kim Y., Jernite Y., and Rush A. M. "Character-Aware Neural Language Models". *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016. 3.1

[22] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level Convolutional Networks for Text Classification". *Proceedings of the 28th International Conference on Neural Information Processing Systems*, 2015. 3.1

[23] Jenny Finkel, Trond Grenager, and Christopher Manning. "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling". *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics*, pages 363–370, 2005. 3.2, 8, 8.2, 8.3

[24] Edward Loper, Ewan Klein, and Steven Bird. "Natural Language Processing with Python". *O'Reilly Media Inc*, 2009. 7.1.2.1, 8.2

[25] M. F. Porter. "An algorithm for suffix stripping". *Program: electronic library and information systems*, 14(3):130–137, 1980. 7.1.2.3

[26] Sebastian Raschka. "Naive Bayes and Text Classification: Introduction and Theory". http://sebastianraschka.com/Articles/2014_naive_bayes_1.html, October 2014. 7.1.2.4

[27] A. Zevcevic. "N-gram based text classification according to authorship". In *Student Research Workshop*, pages 145–149, 2011. 7.1.2.4

[28] V. Keˇselj, F. Peng, N. Cercone, and C. Thomas. "N-gram-based author profiles for authorship attribution". In *Proceedings of the conference pacific association for computational linguistics*, volume 3, pages 255–264. PACLING, 2003. 7.1.2.4

[29] I. Kanaris, K. Kanaris, I. Houvardas, and E. Stamatatos. "Words versus character n-grams for anti-spam filtering". *International Journal on Artificial Intelligence Tools*, 16(06):1047–1067, 2013. 7.1.2.4

[30] Bogdan. What's so hard about PDF text extraction? https://https://filingdb.com/b/pdf-text-extraction, 2020. [Online]. 12.1