INFORMATION EXTRACTION FROM 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

October, 2017

ABSTRACT

In this project, we used web crawler with Python to download journal papers on soil science from the digital library. Then applied name entity recognition, table analysis, text analysis on the text data, and extracted useful information from the journal paper and stored them in the personal designed database. The name entity recognition technique is used to extract authors, experiment location information. And the table analysis is used to store the tables from the journal paper in a computer queryable form. In text analysis part, we 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 or not.

Don't use "in this thesis" or "in this project"

First, you need to motivate your work: why are you using a web crawler to download papers, applying NER, table analysis, etc.? Then explain why each part of your thesis is necessary, e.g., why you need to harvest the journal papers, why you need to do NER, and so on. Then, describe the basic ML approaches you used for each part. Finally, summarize your experimental results, including how they compare to the state of the art.

ACKNOWLEDGMENTS

Which one(s)? State the number(s).

The thesis is funded by United States Department of Agriculture (USDA) grants.

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 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 would like to thank my family for supporting me spiritually throughout my study and completing my degree.

Contents

Chapter 1

Introduction

The part of the project consists of five parts: Web harvesting, Text Classification, Table Analysis, Named Entity Recognition and Database Design. The purpose of this project is to populate a relational database with data automatically extracted from journal papers collected from internet resources.

Most time on this project was spent on cleaning data and checking results.

First, you need to motivate your work: why are you using a web crawler to download papers, applying NER, table analysis, etc.? Then explain why each part of your thesis is necessary, e.g., why you need to harvest the journal papers, why you need to do NER, and so on. Then, describe the basic ML approaches you used for each part. Finally, summarize your experimental results, including how they compare to the state of the art. Included in all of this should be a list of your specific contributions.

Chapter 2

Prior Work

Chapter 3

System Overview

Since this chapter is a "System Overview", you should open with a figure overviewing the system, and explain each system module in detail. Figure 3.1 is more on how you trained and tested an ML component, which you did multiple times, correct? Perhaps that should go later, in an ML background section or in the experimental section.

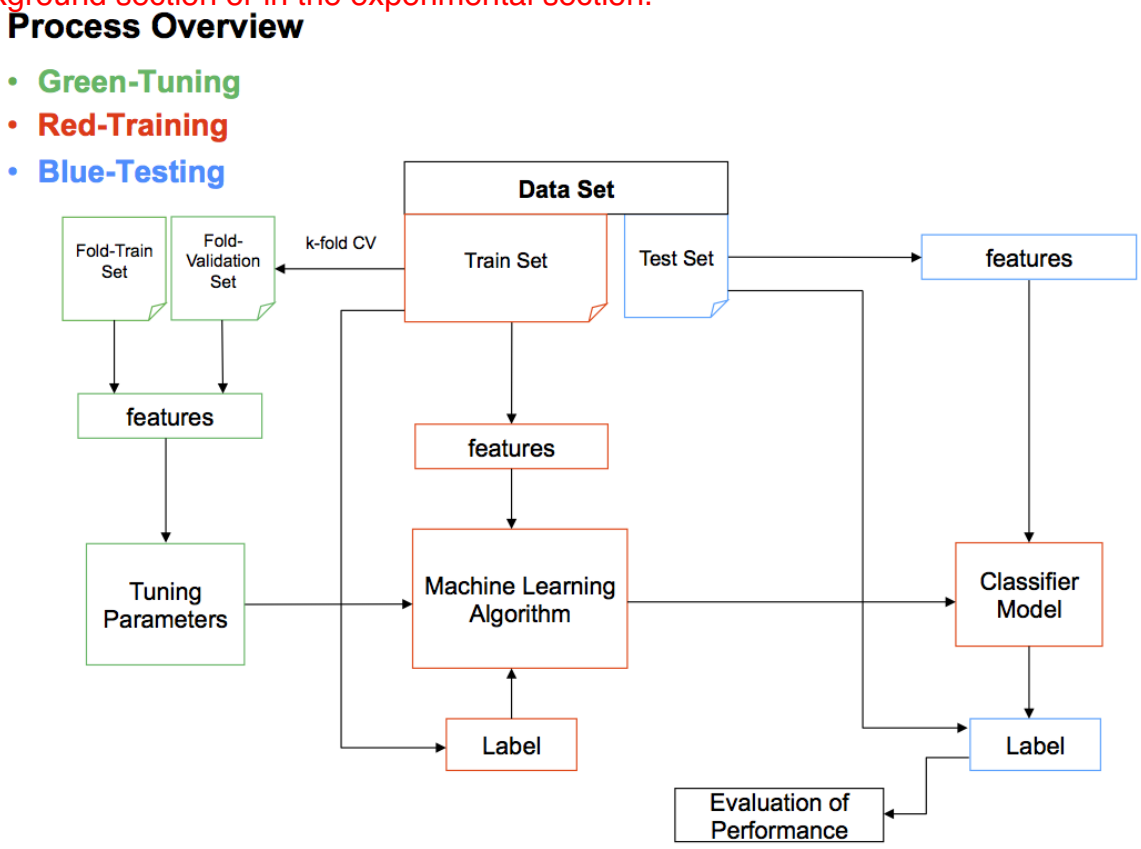


Figure 3.1: Machine Learning System Design

Chapter 4

First, give background on: (1) what data you collected; (2) why you collected it; (3) how you collected it; and then, (4) what the statistics were.

Data Statistical Description

Use complete sentences

The file information about section text data:

The total number of file is: 1690

The total number of journal is: 207

That's an average of 8.2 file per journal

Put table captions above tables

		File Name	N_chs_cleanText				N_words_cleanText			
		count	min	max	sum	mean	min	max	sum	mean
Journal	Label									
001	0	5	304	17013	33326	6665	42	2648	5047	1009
	1	2	908	5252	6160	3080	116	810	926	463
002	0	3	1500	7808	12343	4114	212	1035	1677	559
	1	4	358	8843	15664	3916	49	1422	2471	617
003	0	4	269	6096	13492	3373	33	835	1901	475
	1	3	1026	13346	21576	7192	137	2052	3311	1103
004	0	4	1946	11662	27082	6770	279	1694	3912	978
	1	3	881	12224	18897	6299	130	1874	2876	958
005	0	6	445	11785	30407	5067	67	1767	4532	755
	1	3	1068	9210	17231	5743	144	1454	2754	918

Table 4.1: 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	0	24	257	8532	33948	1414	39	1262	5152	214
	1	7	107	1345	5478	782	17	206	821	117
002	0	23	263	7810	22388	973	39	1035	3292	143
	1	7	337	1709	5562	794	49	250	856	122
003	0	23	271	6098	29950	1302	33	835	4450	193
	1	5	487	1733	5067	1013	71	275	762	152
004	0	32	136	9099	35797	1118	20	1318	5241	163
	1	10	572	1735	10089	1008	87	283	1547	154
005	0	19	298	8025	33275	1751	40	1161	4982	262
	1	14	380	1797	14307	1021	58	294	2304	164

Table 4.2: Paragraph text data statistical description

There's enough room here to write out "interest" and "no interest"

The file information about paragraph text data:

The total number of file is: 7543

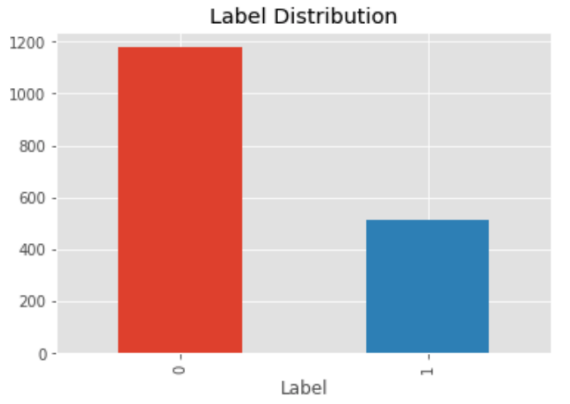
The total number of journal is: 207

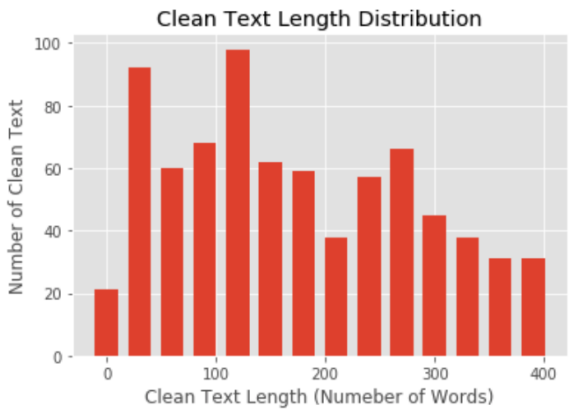
That's an average of 36.4 file per journal

The Table 4.1 and Table 4.2 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 4.1 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 4.1 and Figure 4.2 indicates

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 4.1 and Figure 4.2 provide a outline to show us what the data looks like.





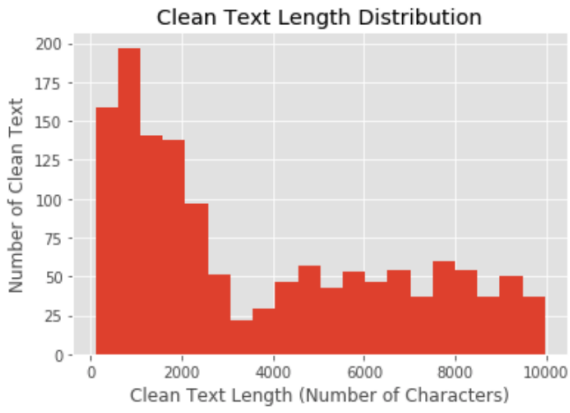
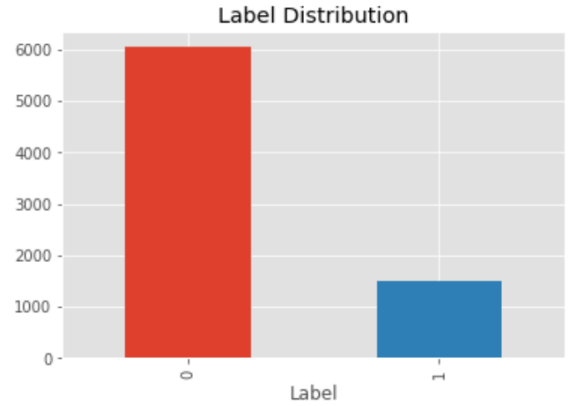
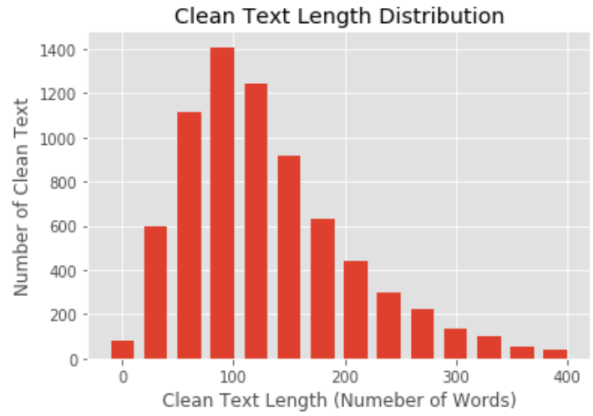


Figure 4.1: Distribution of Section Data





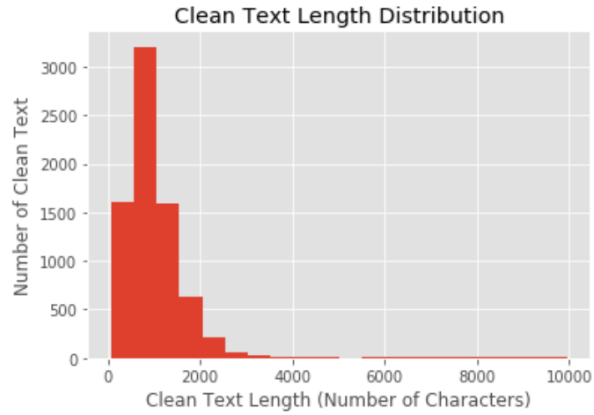


Figure 4.2: Distribution of Paragraph Data

This chapter should be earlier

Chapter 5

Machine Learning Background

First, give a basic definition of what ML is, example applications, etc.

5.1 Machine Learning Algorithms

Define these terms.

No one classifier model works best across all scenarios, since every algorithm 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 optimal result.

Optimal in what way? How is performance measured?

In this project, we tried logistic regression, support vector machine, decision tree, naive bayes, k-nearest neighbors, and some ensemble methods on our dataset in order to select the best one fitting our task ~~purpose~~.

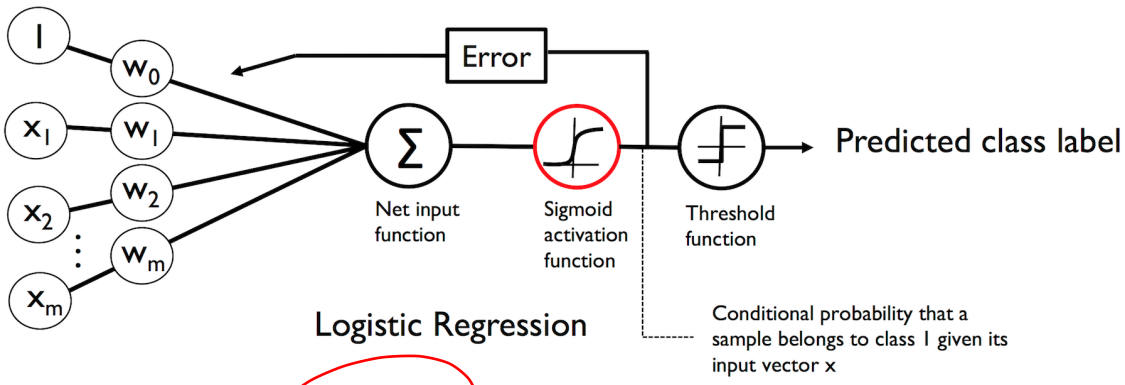
5.1.1 Logistic Regression

Logistic regression is a binary classification model that is very easy to implement ~~but~~ and performs very well on linearly separable classes. It uses sigmoid function to compress the input features to output ranges from 0 to 1, and map~~ped~~s the output to the class labels "0" or "1".

Define

the

Define



You don't refer to this figure in the text

Figure 5.1: Logistic Regression Model

Let $x_0, x_1, \cdots, x_m$ indicate sample features and $w_0, w_1, \cdots, w_m$ ~~are relevant~~ be representing the model weights.

Z is the net input, the linear combination of weights and sample features,

Be consistent with capitalization.

$$z = w^T x = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m.$$

In ~~the~~ 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 o 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.

Be consistent with italics

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 $\frac{p}{1-p}$. $logit(p) = \log \frac{p}{1-p}$, where the log refers to the natural logarithm, ~~as it is the common convention in computer science~~. 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:

$$logit(p(y = 1|x)) = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^{m} w_ix_i = w^Tx$$

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, ~~which is the inverse form of the logit function~~. ~~It is also called~~ we use the logistic sigmoid function: $\phi(z) = \frac{1}{1+e^{-z}}$

5.1.2 Support Vector Machine

As shown in the Figure 5.2, 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 ~~eventually~~ define establish the SVM ~~model~~ classifier.

5.1.2.1 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

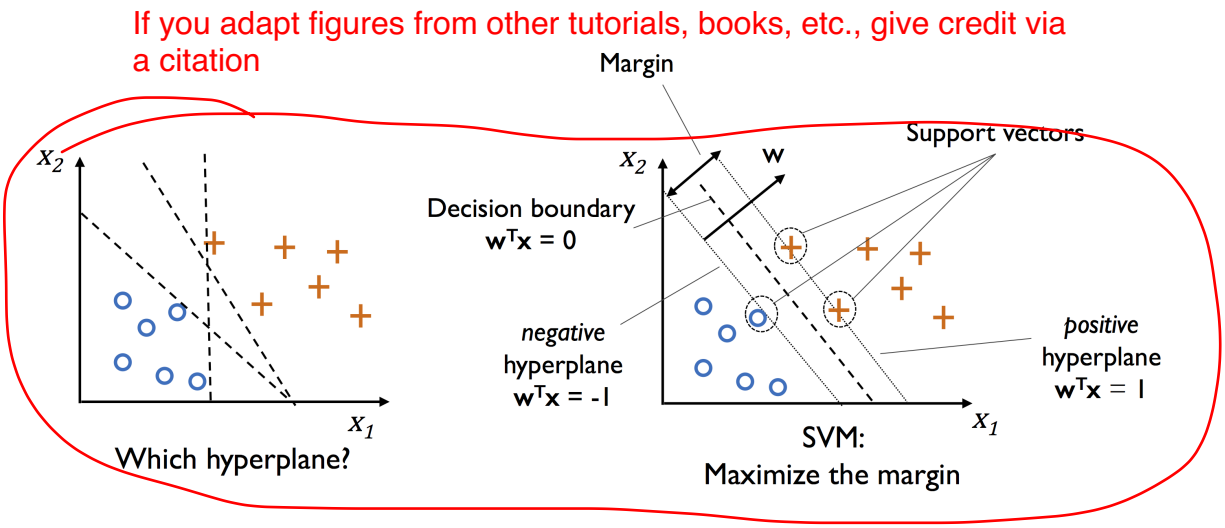


Figure 5.2: Hyperplane and margin in SVM

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:

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

Give the equation a number and refer to it by number

The left side of the preceding equation 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:

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

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

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 ~~is relevant to the~~ 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 values slack variable, the previous equations become:

Refer to equation numbers.

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

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

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. Large value of C correspond to large error penalty, whereas small value of C indicates less strict about misclassification error. We can use the C parameter to control the width of the margin and therefore tune the bias-variance trade-off, as shown in ~~the~~ Figure 5.3. 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.

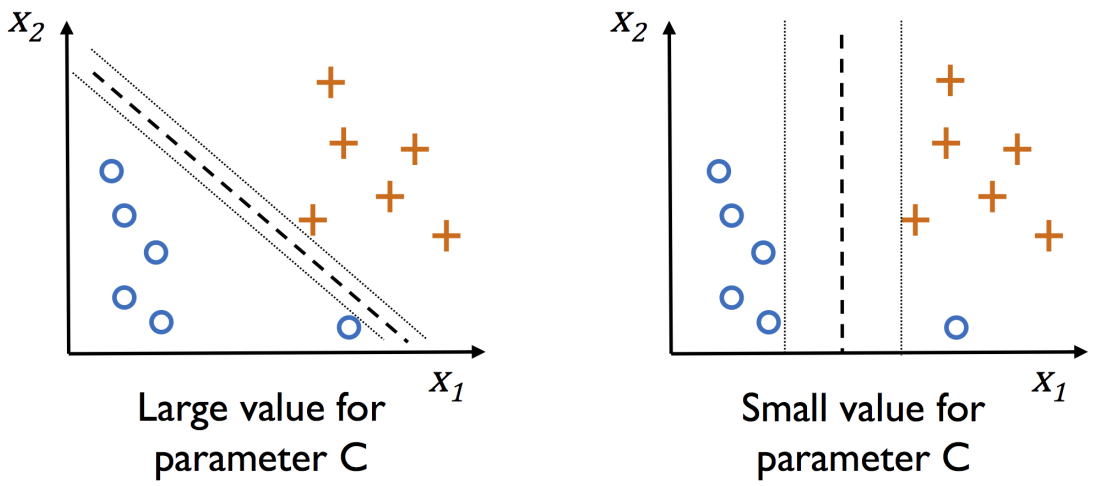


Figure 5.3: The impact of C ~~in~~ on the margin

5.1.2.2 Kernel SVM

Italicize or boldface terms when you introduce them

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

a

understood as a similarity measure in higher-dimensional space.

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

Since the explicit computation of the kernel is increasingly computationally expensive for large sample sizes and high numbers of dimensions, the kernel trick uses approximations to calculate the kernel implicitly. The most popular kernels used for the kernel trick are Gaussian Radius Basis Function (RBF) kernels, sigmoidal kernels, and polynomial kernels.

What kinds of approximations?

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.

5.1.3 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 result

in overfitting. You haven't defined this

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:

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

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: Set this apart from the text as "Algorithm 1" with a caption, and refer to it in the main text

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

~~The~~ 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 $p(i \mid t \neq 0)$, ~~where~~ $p(i \mid t)$ 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 0 if all samples at a node belong to the same class, and the entropy is maximal if we have an uniform class distribution.

Italicize

Zero?

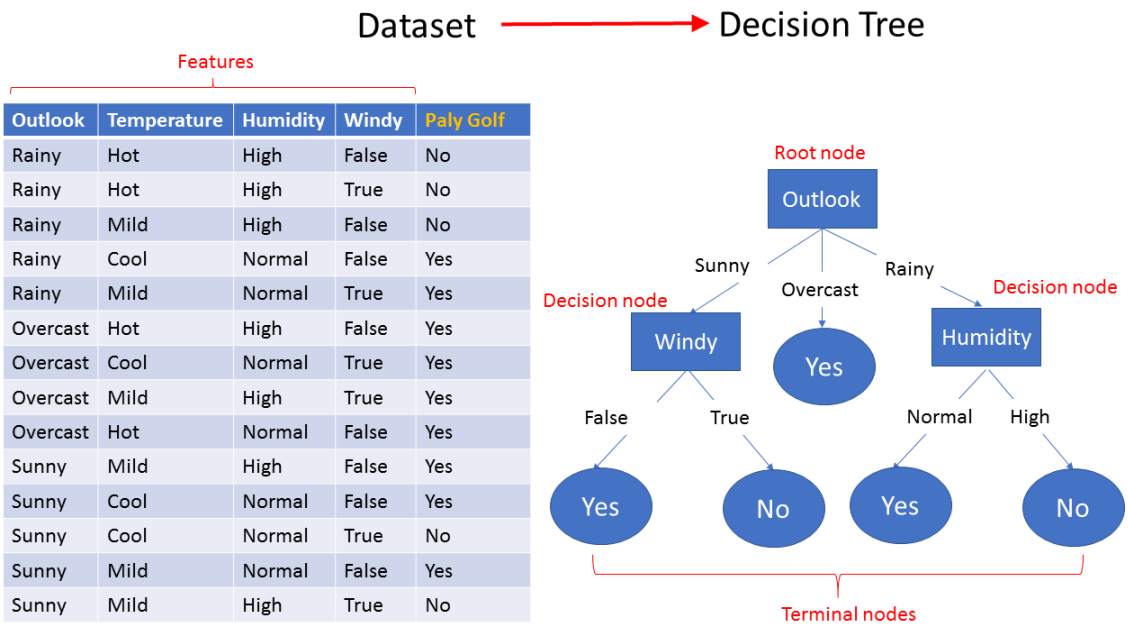


Figure 5.4: Decision Tree Construction

Indicate that this example is adapted from Mitchell, and cite his book

The decision tree image is shown in Figure 5.4. Looking at the decision tree construction figure, we can nicely trace back the splits that the decision tree determined from our training dataset.

5.1.4 Naive Bayes

Naive Bayes classifiers, a family of classifiers that are based on the ~~popular~~ 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 [1]. Especially for small sample sizes, naive Bayes classifiers can outperform the more powerful alternatives [2].

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 ~~the~~ 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.

5.1.5 K-Nearest Neighbors

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. While the disadvantage is: First, the computational complexity 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 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.

5.1.6 Random Forest

Random forest is an ensemble classifier where multiple decision tree classifiers are combined via the bagging technique. Unseen/test objects are then classified by taking the majority of votes from individual decision trees.

Random Forest can be considered ~~as~~ 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 n samples from the training dataset with replacement.

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

a. Randomly select 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 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:

~~Advantage~~:

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

5.2 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. Different methods, e.g., bagging or boosting, are used to construct the final classification decision based on weighted votes.

Ensemble methods ~~are to~~ 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.

5.2.1 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

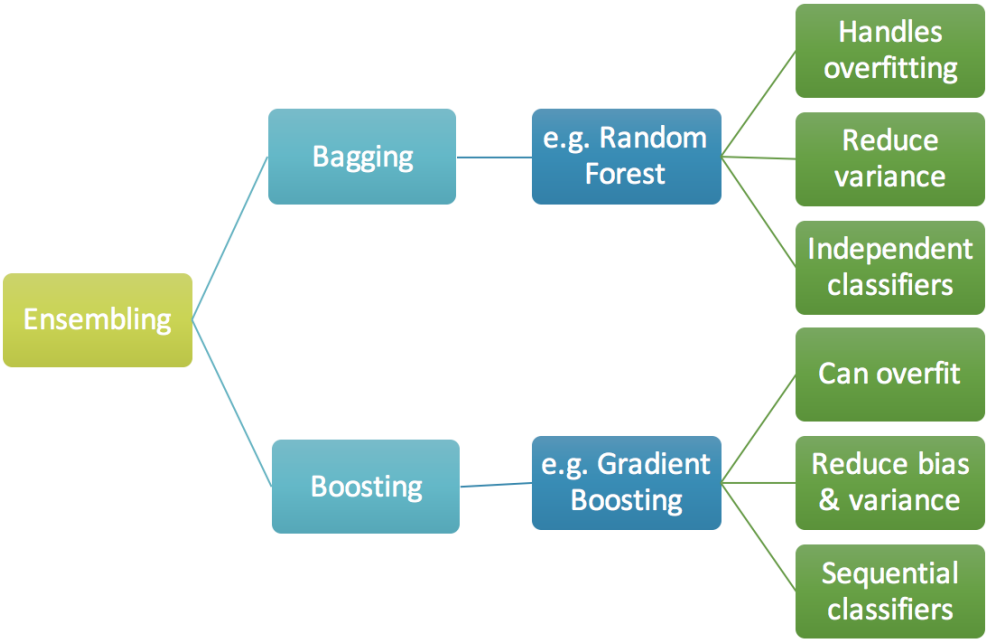


Figure 5.5: Ensemble Methods

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, ... , C_n. The ensemble can be built from different classification algorithms, for example, linear regression, logistic regression, decision tree, support vector 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, Neural Networks, 5(2):241259, 1992. Use a numbered citation, e.g., "[3]"

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:

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

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 (~~Greek chi~~) is the characteristic function $[C_j(x) = i \in A]$, and 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, 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:

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

Here, p_{ij} is the predicted probability of the jth classifier for class label 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 x:

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

We can then calculate the individual class probabilities as follows:

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

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

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

5.2.2 Bagging

????

A resampling technique to that is closely related to cross-validation where a training dataset is divided into random subsets. 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.

Bagging 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

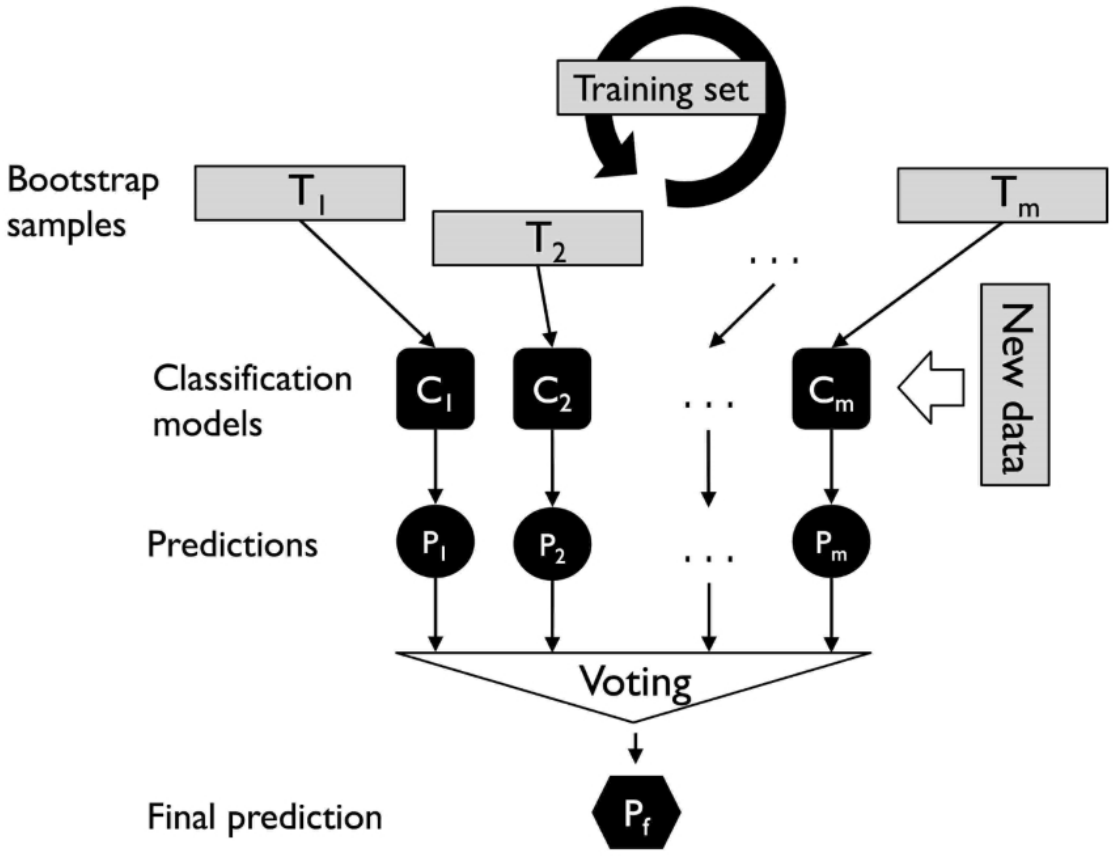


Figure 5.6: The concept of bagging

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 we used in the project is an application of bagging technique.

Refer to the section number

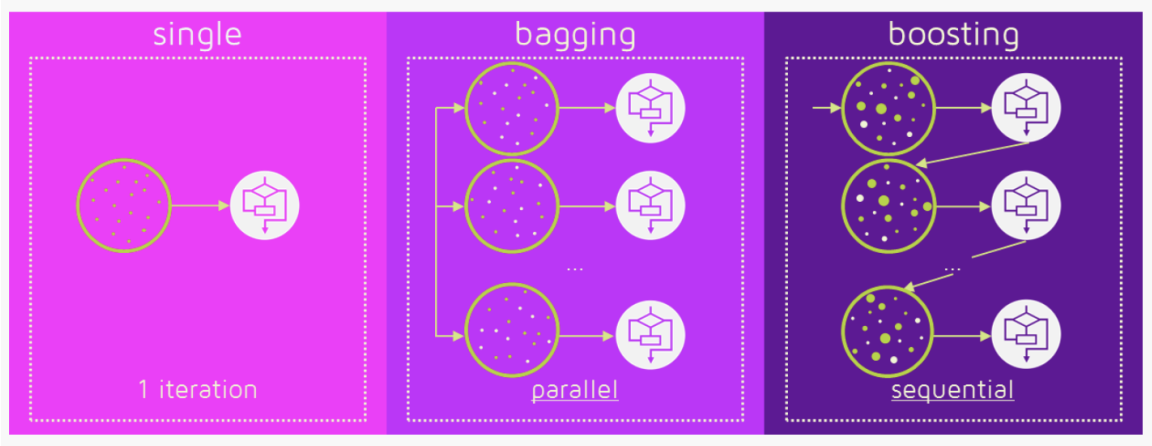


Figure 5.7: Comparison of Bagging and Boosting

5.2.3 Boosting

Explain this, and define "weak"

In boosting, the ensemble consists of weak classifiers. 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 procedure is described in the following steps:

a. Draw a random subset of training samples D_1 without replacement from training set D to train a week classifier C_1.

b. Draw a second random training subset D_2 without replacement from the training set and add 50 percent of the samples that were previously misclassified by C_1 to train another weak classifier C_2.

c. Find the training samples D_3 in training set D, which C_1 and C_2 disagree upon, to train a third weak classifier C_3.

d. Combine the weak classifiers C_1, C_2, and C_3 via majority voting.

In contrast to this original boosting procedure as described above, we use

You can cut this, since AdaBoost is much more common, and I assume that you used AdaBoost in your thesis.

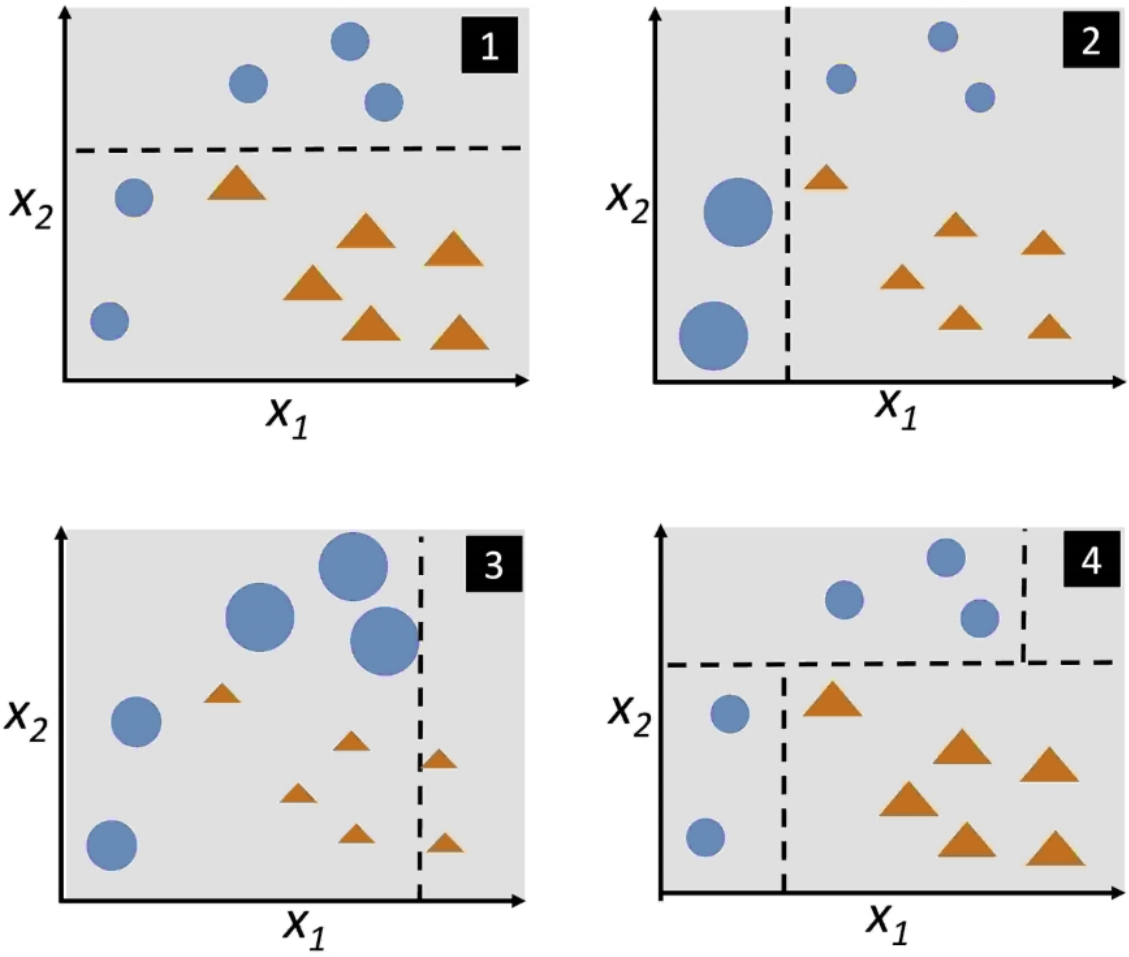


Figure 5.8: The concept of boosting

adaptive boosting, which is a popular variant of boosting. Adaptive boosting (AdaBoost) 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 to build a strong classifier. The mechanism behind AdaBoost is as follows:

Make this an algorithm, set apart from the main text, like a figure.

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: $\hat{y} = (\sum_{j=1}^{m}(\alpha_j \times predict(C_j, X)) > 0)$

5.3 Performance Evaluation Metrics

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

5.3.1 Confusion Matrix

The confusion matrix 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 5.9: Confusion Matrix

Use full sentences

True Negatives (TN): case was negative and predicted negative

True Positives (TP): case was positive and predicted positive

False Negatives (FN): case was positive but predicted negative

False Positives (FP): case was negative but predicted positive

5.3.2 True and False Positive Rates

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.

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

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

5.3.3 Accuracy, Precision, Recall and F measure

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 negatives) among the total number of cases examined.:

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

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

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

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

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

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.:

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

5.3.4 Receiver Operator Characteristics (ROC)

Receiver Operator Characteristics (ROC) ~~graphs~~ curves are useful tools to select classification models based on their performance with respect to True Positive rates and the False Positive.

~~The Receiver Operating Characteristic (ROC, or ROC curve) is a quality measure for binary prediction algorithms by plotting the "False positive rate" vs. the "True positive rate"~~

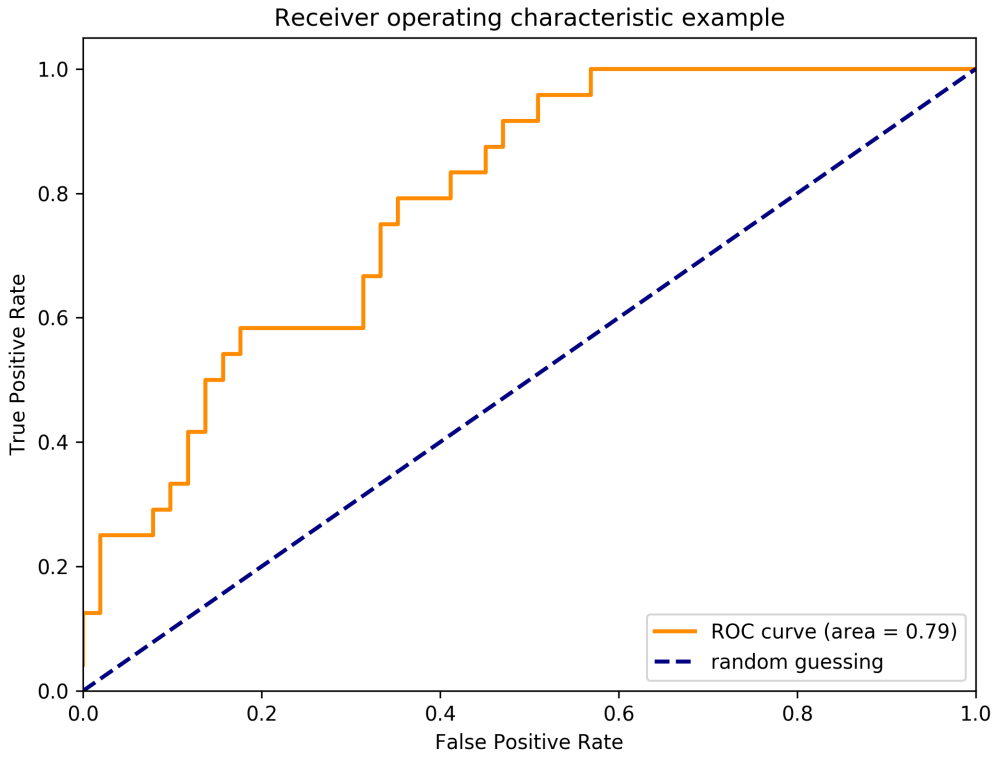


Figure 5.10: Example of a Receiver Operating Characteristic. This plot was created using the Python scikit-learn machine learning library.

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.

Chapter 6

Web Scraping

In The first pass of paper downloading, we download 3,657 papers with a combined size of 4.15 GB. These papers were collected from on-line sources of these journals (shown in the appendix), using the following query terms: soil quality and conservation management. 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 these journals (shown in the appendix), 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 we download is 38,444 and the total size is 29.53 GB.

6.1 Procedure of Web Scraping

The first pass of paper downloading, we download 3,657 papers with a combined size of 4.15 GB. The second time, we download papers from more resources by filtering words like Soil Quality; Soil Management; Dynamic Soil Properties; Soil Health. Finally, the total paper we got is 34,787 and the size is 25.38 GB. The details

about numbersof papers and related sources are in the appendix.

The first time, the library we chose to download papers is ACSEE Digital Library through UNL library.

Refer to this figure in the main text

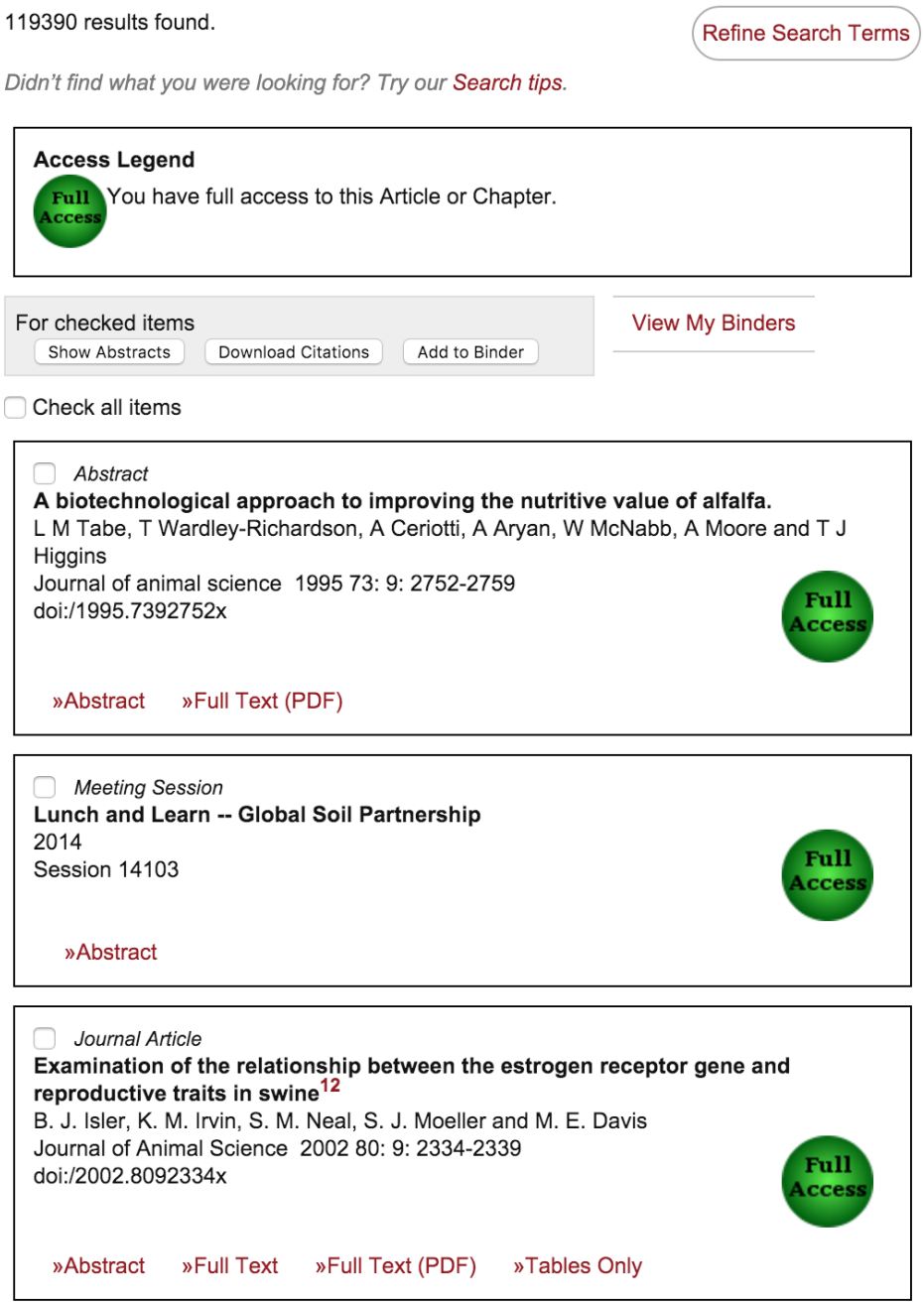
Figure 6.1: ACSEE Digital Library

Not a sentence

Filter out Meeting Session, Book Chapter, and other resources, and keep files from Journal Article. Finally, we got 3,657 papers and the size is 4.15 GB.

The second time, we download papers from more resources by filtering words like Soil Quality; Soil Management; Dynamic Soil Properties; Soil Health. Finally, the total paper we got is 34,787 and the size is 25.38 GB. The number in front of journal title is the number of papers we download.

Repeated



Refer to this figure in the main text

Figure 6.2: Search results without any filter words.

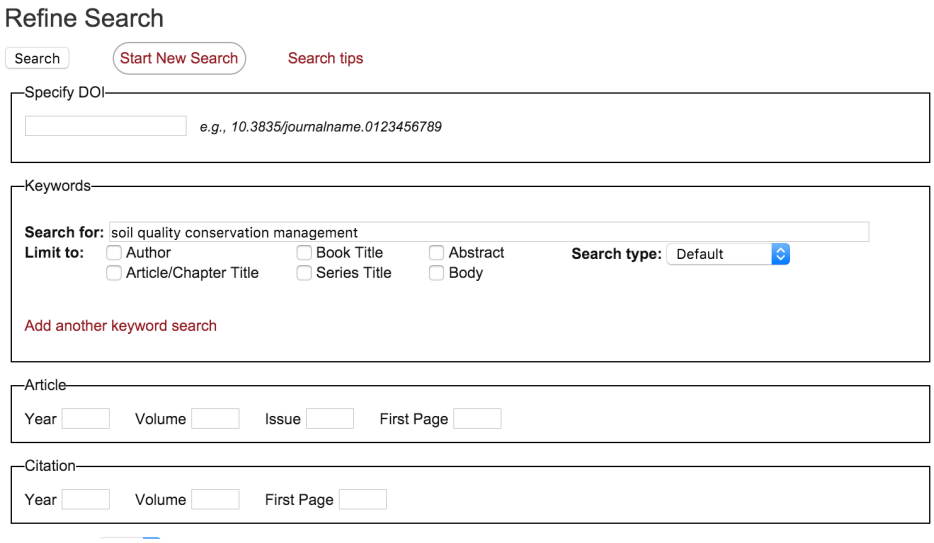


Figure 6.3: Restrict to some terms: soil quality, conservation management.

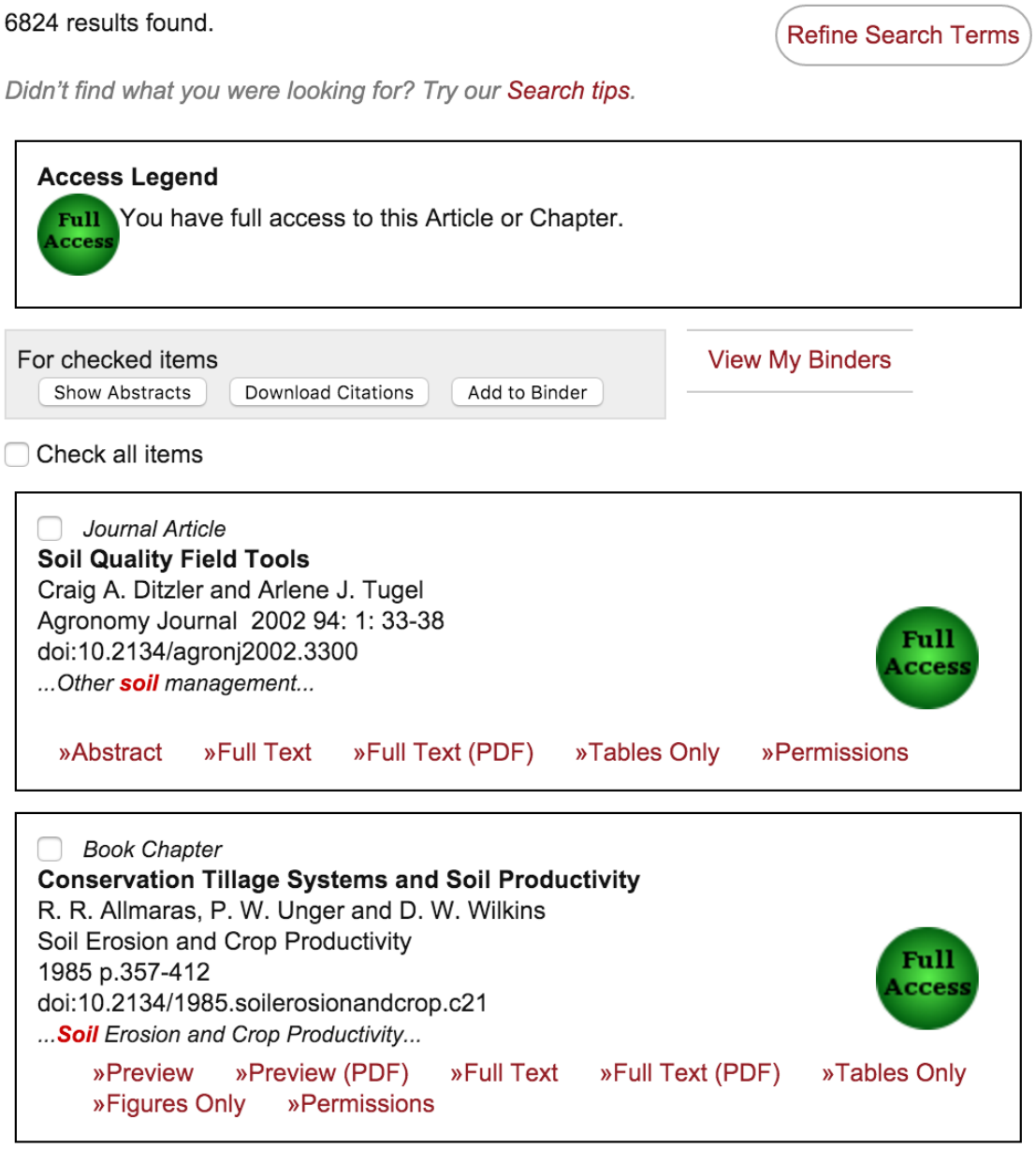


Figure 6.4: Search results with filter words.

You can remove some of the search result screens; it'll be more informative if you simply describe how your scraping script works.

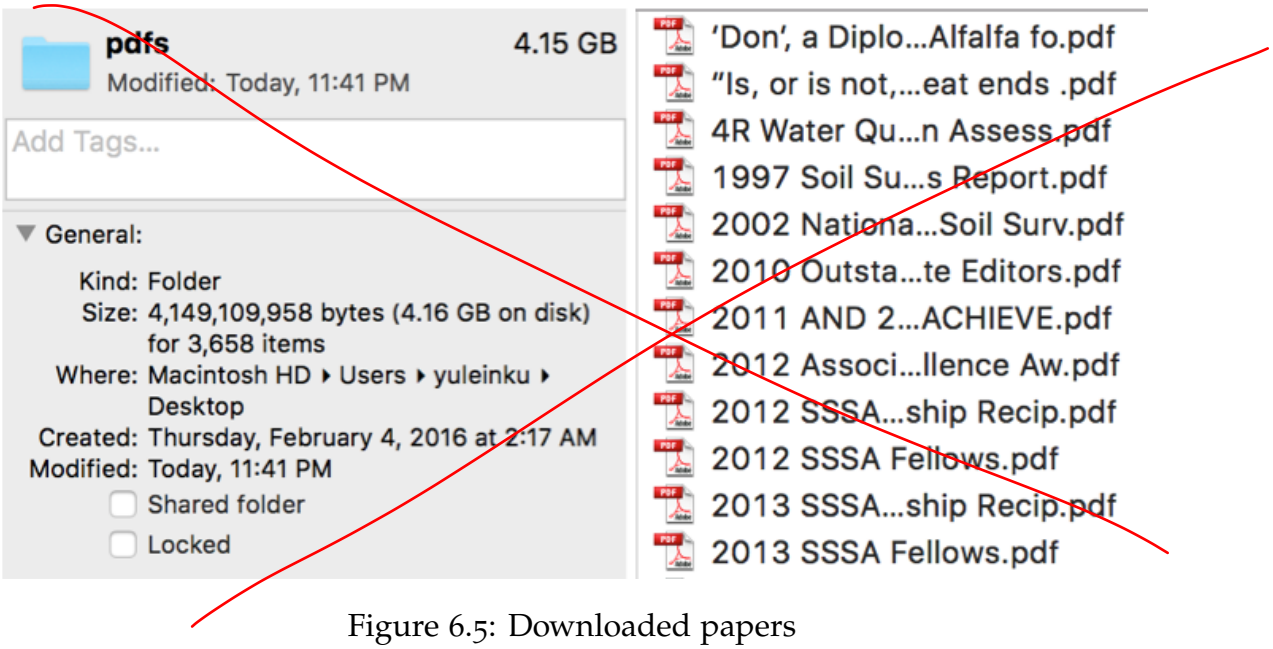


Figure 6.5: Downloaded papers

6.2 Summary of Downloaded Papers

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

Table 6.1: Downloaded Journal Papers and associated recourse

Where's the discussion on your script, how you scraped, etc?

Chapter 7

Text Analysis via Machine 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 ~~learning~~ training on already highlighted documents, in order to save the users new paper seeking time and save the desirable information in queryable database.

Indicate that highlighted => desirable

Mention that you need to first convert pdf to text, then state that, while there are many conversion tools, there's 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, ~~and then~~ making the problem would be more serious. Another common problem is that 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.

Why is this text here? It doesn't relate to the previous paragraph.

Supervised pattern classification is the task of training a model based on labeled training data which then can be used to assign a pre-defined class label to new objects.

In this section, we ~~would~~ 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 ~~method~~. Give the section number

???

The section part consists of 1690 files that are labeled as either positive or negative (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 the paragraph part consists of 7543 files (6045 positive and 1498 negative).

Describe the labeling process: who did it, how it was done, how it's represented in the text files, etc.

After we got these files, we ~~will~~ 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.

7.1 Data Preprocessing

You've used past, present, and future tense of verbs in this thesis. Be consistent.

To handle text data easier, we will be reading the text data into a pandas DataFrame object. This object would give more structured data and better visualization.

7.1.1 Unicode Handling

You can eliminate or severely reduce this section

Unicode [3] starts with ASCII, includes thousands of symbols and its goal is to have everything. It assigns integers, called code points, to characters. It has enough space for future growth since it has room for 1.1 million code points, and now only 110,000 are already assigned,

We need unicode [4] when we handle with the text like splitting, slicing, joining, adding, removing, uppercasing, lowercasing. The reason why we need to use unicode is that Unicode provides a unique number for every character no matter what the platform, the program and the language.

Everything in a computer is bytes. Files on disk are a series of bytes, and network connections only transmit bytes. Even just attempting to print a unicode string will cause an implicit encoding: output is always bytes, so the unicode string has to be encoded into bytes before it can be printed. So almost without exception, all the data going into or out of any program you write, is bytes. The problem with bytes is that by themselves they are meaningless, we need conventions to give them meaning. We need Byte strings ONLY during the Input/Output, like reading/writing from files, sockets.

UTF-8 is easily the most popular encoding for storage and transmission of Unicode. It uses a variable number of bytes for each code point. The higher the code point value, the more bytes it needs in UTF-8. ASCII characters are one byte each, using the same values as ASCII, so ASCII is a subset of UTF-8.[5]

Because bytes and unicode are both important, and we need to deal with both of them, we can't pretend that everything is bytes, or everything is unicode. We need to use each for their purpose, and explicitly convert between them as needed.[6]

Python default source code encodings, if not specified, is ASCII on Python 2 and UTF-8 on Python 3. In Python 2, the unicode type represents a real string, whereas the str type is a sequence of bytes. If we see a str object, convert it to a unicode object right away by calling .decode('utf-8'). Process all strings as unicode objects, not str objects.[7]

Once we're done processing our Unicode strings, if we want to write them out to a file or database, first convert them back to a sequence of bytes (the str type) using the encode method, like calling .encode('utf-8') on it.If we don't follow this convention, then well likely see weird errors when processing strings with non-English characters.[8]

Python 3 makes handling Unicode much simpler. The biggest change is that the str type actually holds Unicode strings, not a sequence of bytes.

This creates a Unicode flow as shown in the figure 7.1: bytes on the outside, Unicode on the inside.[9]

So we have to follow[10]: 1. The data coming into and going out of our program must be bytes. But we don't need to deal with bytes on the inside of our program. The best strategy is to decode incoming bytes as soon as possible, producing unicode. We use unicode throughout our program, and then when outputting data, encode it to bytes as late as possible.

2. We have to know what kind of data we are dealing with. At any point in our program, we need to know whether we have a byte string or a unicode string. This shouldn't be a matter of guessing, it should be by design. In addition, if we

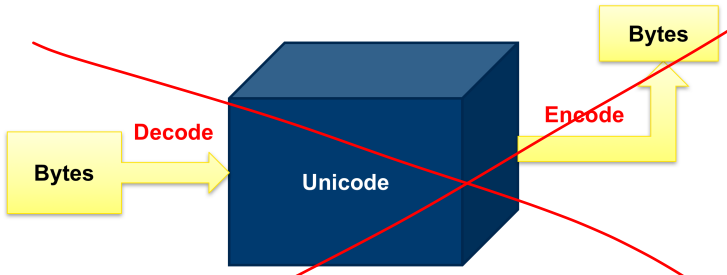


Figure 7.1: Unicode flow

have a byte string, we should know what encoding it is if we ever intend to deal with it as text.

7.1.2 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.

7.1.3 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.

7.1.3.1 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 [cite] has a set of 127 English stop words. And we could use it to remove stop words in the text.

7.1.3.2 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.

7.1.3.3 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[11]. Space

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.

7.1.3.4 N-Grams

In the n-gram model [12], 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. All previous examples were unigrams so far. 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 [13]. 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 [14] and Kanaris and others report that n-grams of size 3 and 4 yield good performances in anti-spam filtering of e-mail messages [15].

7.2 Input Data Representation

Since we can not 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.

7.2.1 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 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],

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)]

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

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.

Same paragraph

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. 0.23 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.]

7.3 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. ~~And~~ this ~~would~~ yield s 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 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.

In this project we use 10-fold cross-validation, and we randomly split the training data into 10 folds without replacement, where 9 folds are used for the model 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.

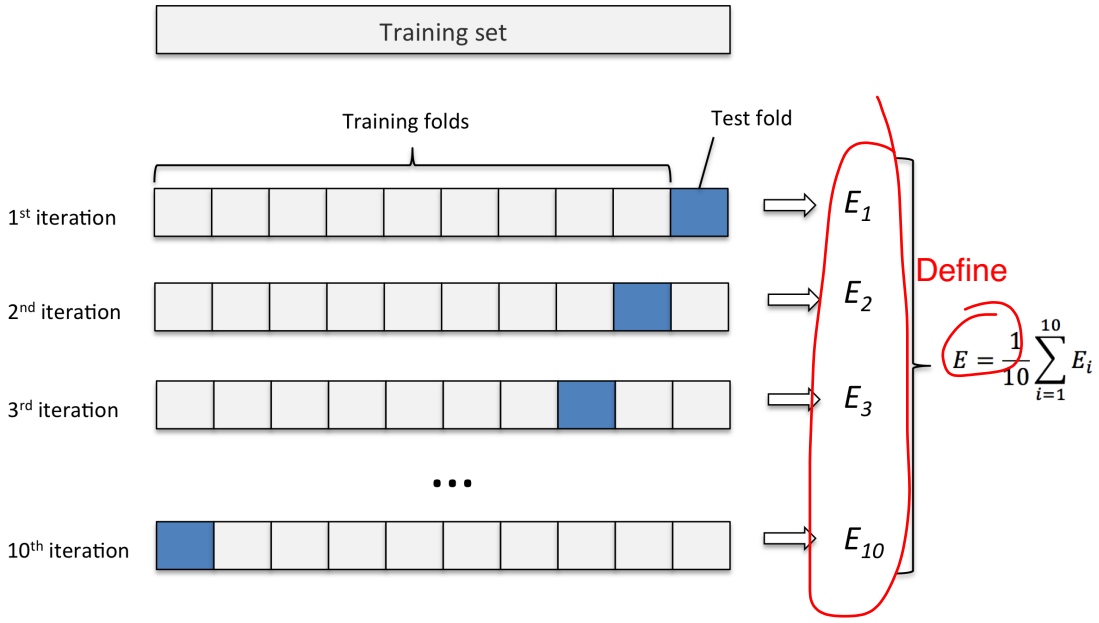


Figure 7.2: 10-fold cross-validation

7.4 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, ~~neuron~~ neural network, which we would get in the ~~fitting~~ ~~model~~ 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.

7.4.1 Logistic Regression

```
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]},
              ]
```

Put code in a figure, and add text to describe it. Do this for each subsection.

- 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.

7.4.2 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}
              ]
```

- 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.

7.4.3 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)
             }
```

- 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.

7.4.4 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],
             }
```

- 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.

7.4.5 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'],
             }
```

- 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.

7.4.6 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"]
             }
```

- 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.

7.4.7 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)
             }
```

- 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.

7.5 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 ~~the~~ 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.

7.6 Performance Evaluation

Need text to present each figure, and to discuss each one.

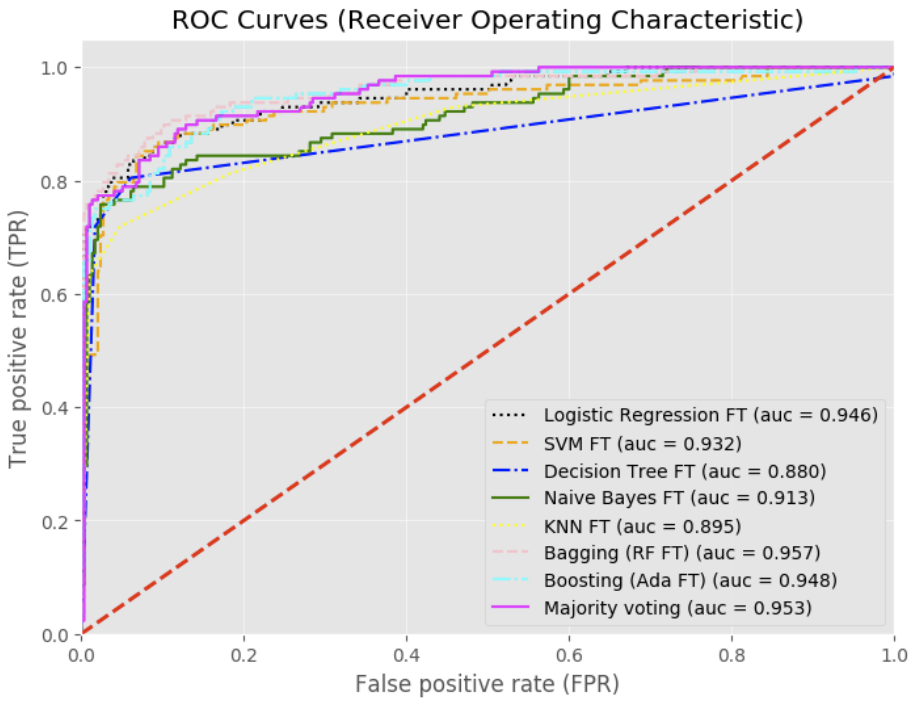


Figure 7.3: ROC Curves for Section text data

Put these in a table.

Train ROC AUC: 0.989, Accuracy: 0.993 [Logistic Regression FT]

Test ROC AUC: 0.869, Accuracy: 0.910 [Logistic Regression FT]

Train ROC AUC: 0.990, Accuracy: 0.994 [SVM FT]

Test ROC AUC: 0.868, Accuracy: 0.908 [SVM FT]

Train ROC AUC: 0.883, Accuracy: 0.929 [Decision Tree FT]

Test ROC AUC: 0.851, Accuracy: 0.903 [Decision Tree FT]

Train ROC AUC: 0.957, Accuracy: 0.973 [Naive Bayes FT]

Test ROC AUC: 0.841, Accuracy: 0.896 [Naive Bayes FT]

Train ROC AUC: 0.886, Accuracy: 0.918 [KNN FT]

Test ROC AUC: 0.836, Accuracy: 0.882 [KNN FT]

Train ROC AUC: 0.842, Accuracy: 0.903 [Bagging (RF FT)]

Test ROC AUC: 0.826, Accuracy: 0.894 [Bagging (RF FT)]

Train ROC AUC: 0.885, Accuracy: 0.928 [Boosting (Ada FT)]

Test ROC AUC: 0.857, Accuracy: 0.908 [Boosting (Ada FT)]

Train ROC AUC: 0.918, Accuracy: 0.949 [Majority voting]

Test ROC AUC: 0.870, Accuracy: 0.917 [Majority voting]

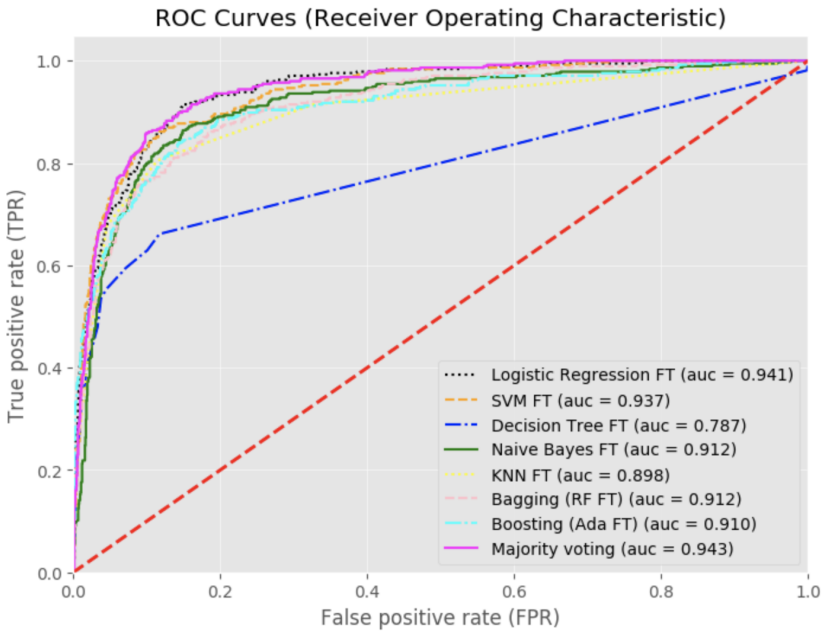


Figure 7.4: ROC Curves for Paragraph text data

Train ROC AUC: 0.967, Accuracy: 0.984 [Logistic Regression FT]

Test ROC AUC: 0.811, Accuracy: 0.899 [Logistic Regression FT]

Train ROC AUC: 0.926, Accuracy: 0.964 [SVM FT]

Test ROC AUC: 0.828, Accuracy: 0.906 [SVM FT]

Train ROC AUC: 0.777, Accuracy: 0.896 [Decision Tree FT]

Test ROC AUC: 0.755, Accuracy: 0.875 [Decision Tree FT]

Train ROC AUC: 0.919, Accuracy: 0.961 [Naive Bayes FT]

Test ROC AUC: 0.722, Accuracy: 0.872 [Naive Bayes FT]

Train ROC AUC: 0.836, Accuracy: 0.915 [KNN FT]

Test ROC AUC: 0.816, Accuracy: 0.899 [KNN FT]

Train ROC AUC: 0.775, Accuracy: 0.896 [Bagging (RF FT)]

Test ROC AUC: 0.752, Accuracy: 0.884 [Bagging (RF FT)]

Train ROC AUC: 0.770, Accuracy: 0.896 [Boosting (Ada FT)]

Test ROC AUC: 0.756, Accuracy: 0.887 [Boosting (Ada FT)]

Train ROC AUC: 0.878, Accuracy: 0.945 [Majority voting]

Test ROC AUC: 0.801, Accuracy: 0.901 [Majority voting]

Time to do what?

	TPR	TNR	Accuracy	Precision	Recall	F-Measure	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 7.1: Section text data

~~The~~ 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. 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.

Are these the learning tasks? You haven't defined the tasks.

	TPR	TNR	Accuracy	Precision	Recall	F-Measure	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	-

Table 7.2: Paragraph text data

Figures 7.5-7.8 Define

~~The~~ Figure 7.5, Figure 7.6, Figure 7.7, and Figure 7.8 show Radar Charts 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 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 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.

Figure 7.5: Radar Chart A of Models for Section text data

Need to define this

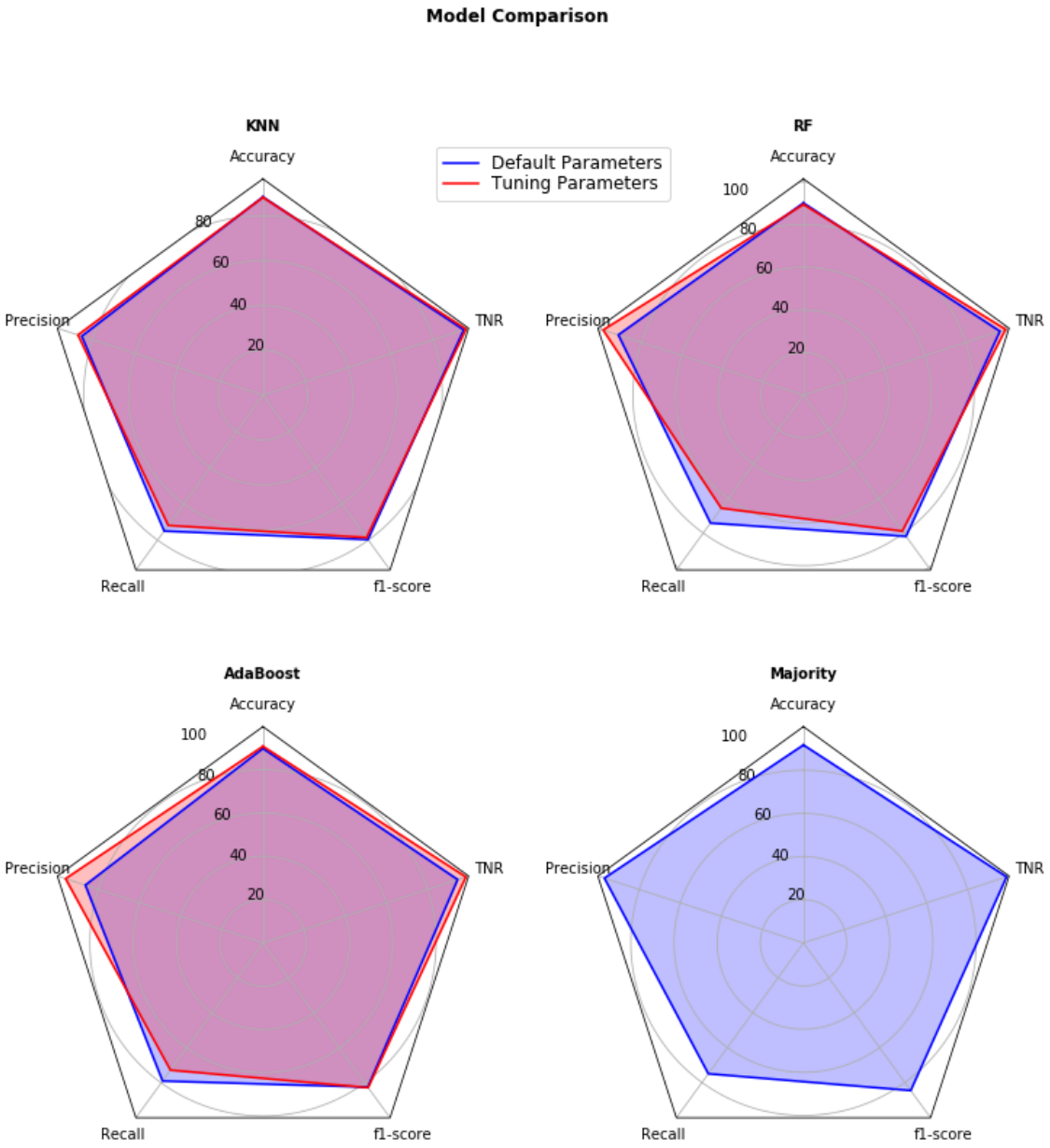


Figure 7.6: Radar Chart B of Models for Section text data

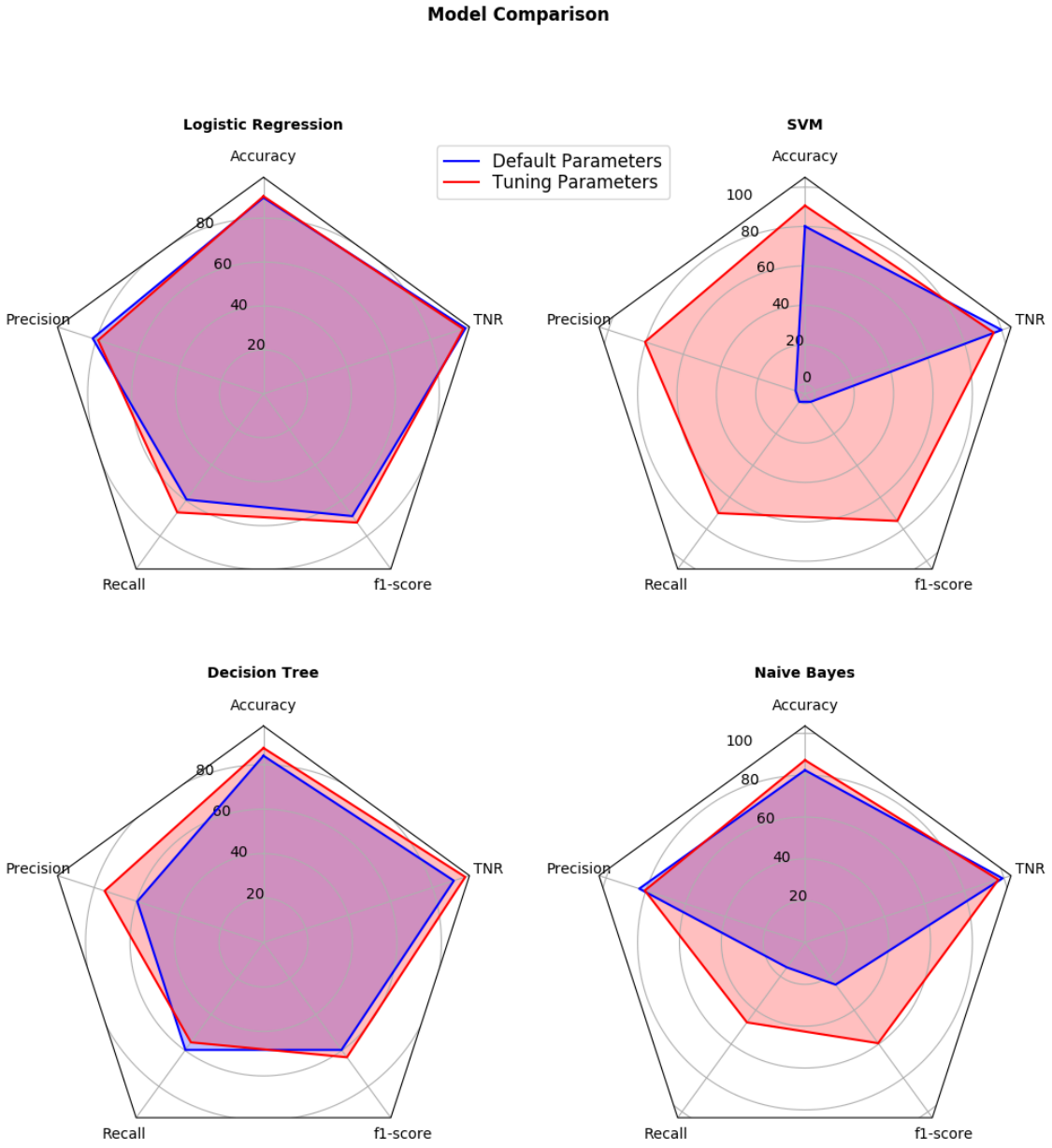


Figure 7.7: Radar Chart A of Models for Paragraph text data

Define

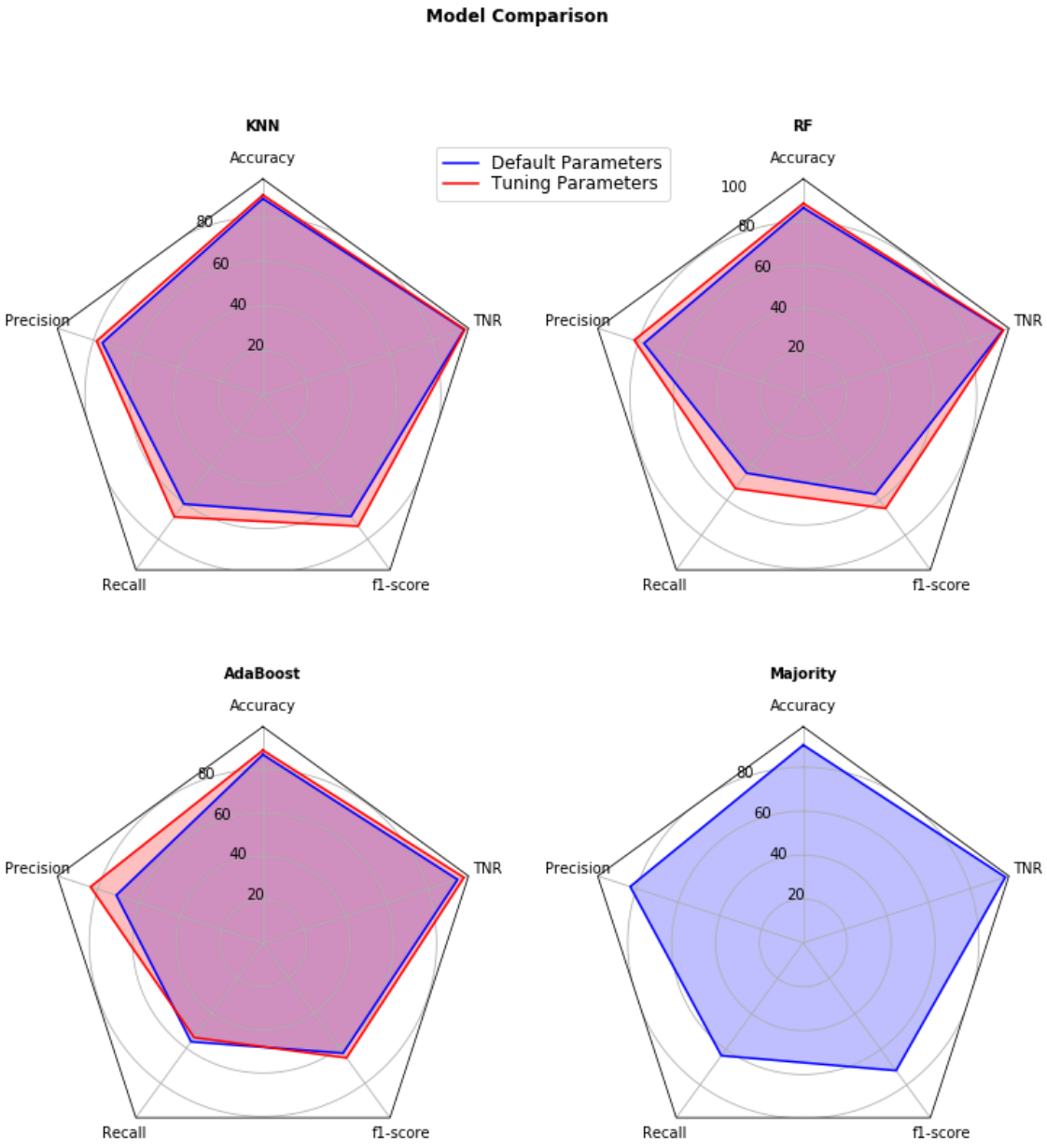


Figure 7.8: Radar Chart B of Models for Paragraph text data

7.7 Conclusion and Discussion

The purpose of Text Classification is that we feed model with users highlighted papers, 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 ~~is~~ for section text classification, and the other ~~is~~ 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.

This is for training, right? Not testing

According to our experiment results, 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. Where was this shown in your experiments?

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. It works very well. 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. ~~It works well.~~

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 and so on. In this project, we use NER to identify persons, organizations, and locations contained in the journal. There are 207 journal papers highlighted by Susan Andrews and Cindy Cambardella, so we use the same papers and tagged the Named Entities as PERSON, ORGANIZATION, LOCATION manually. We need extract PERSON, ORGANIZATION, and LOCATION manually for training model and program performance evaluation. Named entities manual extraction would cost 30-40 minutes per each paper, so the total time is at least 138 hours.

of USDA

Manual?

Give an example of NER, where there is some sample text with xml tags for each entity.

8.1 Purpose

Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, locations, geo-political entities, date, percent and so on. The purpose of NER [2][3][4][5] is to identify all named entities. 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. Named Entity Recognition ~~technique~~ could help us make it a reality. It would extract information about references to PERSONs, ORGANIZATIONs, and LOCATIONs from journal papers. The LOCATION would 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.

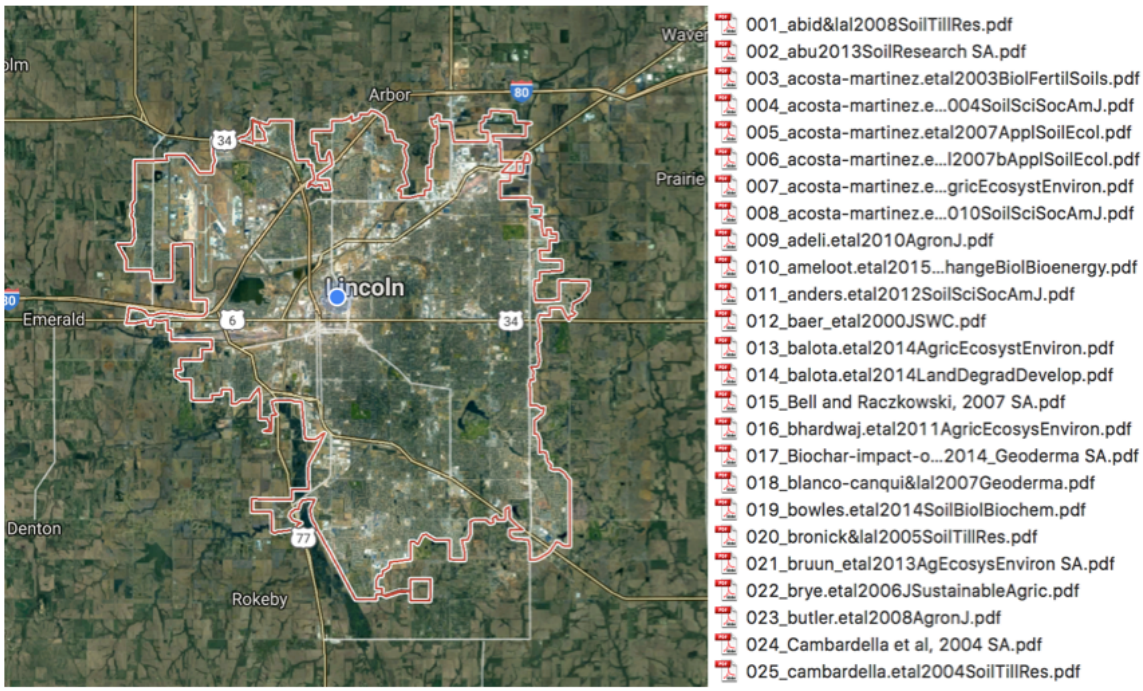


Figure 8.1: Location of Interest and related list of journal papers

Explain the figure. Are the papers from the outlined area?

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

Is this the same one as in the previous chapter?

extracts text contents from a PDF file, and NLTK [6], the Natural Language Tool Kit, serves as one of Pythons leading platforms to analyze natural language data. Not a sentence
Using Regular Expressions to Extract Specific Sections. 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 [9][10], often called Stanford NER. Named Entity Recognition (NER) labels sequences of words in a text that are the names of things, such as person and company names, or gene and protein names. NLTK contains an interface to Stanford NER [11][12], so all codes are written ~~by~~ in Python. We trained our own model via NLTK and Stanford NER, and use them to extract PERSONs, LOCATIONs, and ORGANIZAIONs from journal papers and store them in JSON format. Explain, and cite

The test for the accuracy of a NER tool is to compare the entities extracted by the tools to the hand-labeled extractions. All hand-labeled data are stored in CSV files. The system could extract entitiesPERSON names, LOCATION, and ORGANIZATION names, but only results about LOCATION are shown in this report.

8.3 Evaluation

How did you evaluate? Cross-validation again? What were the test set sizes? What types of entities did you test on?

Terms in Figure 8.2 are Performance Metrics for Different NER methods [13]:

Hand Labeled: Hand labeled data, so the all metrics should be 1. It is the baseline used to test our method. NLTK Chunker-clean: Use trained NER model via NLTK to extract named entities (LOCATION in this case) from clean journal

New paragraph

paper.

NLTK Chunker-noise/clean: Use the named entities extracted from clean journal paperby NLTK tool to test the named entities extracted from noise journal paper.

Stanford NER-clean: Use trained NER model via Stanford NER [14] to extract named entities (LOCATION in this case) from clean journal paper.

Stanford NER-noise/clean: Use the named entities extracted from clean journal paper by Stanford NER tool [15] to test the named entities extracted from noise journal paper.

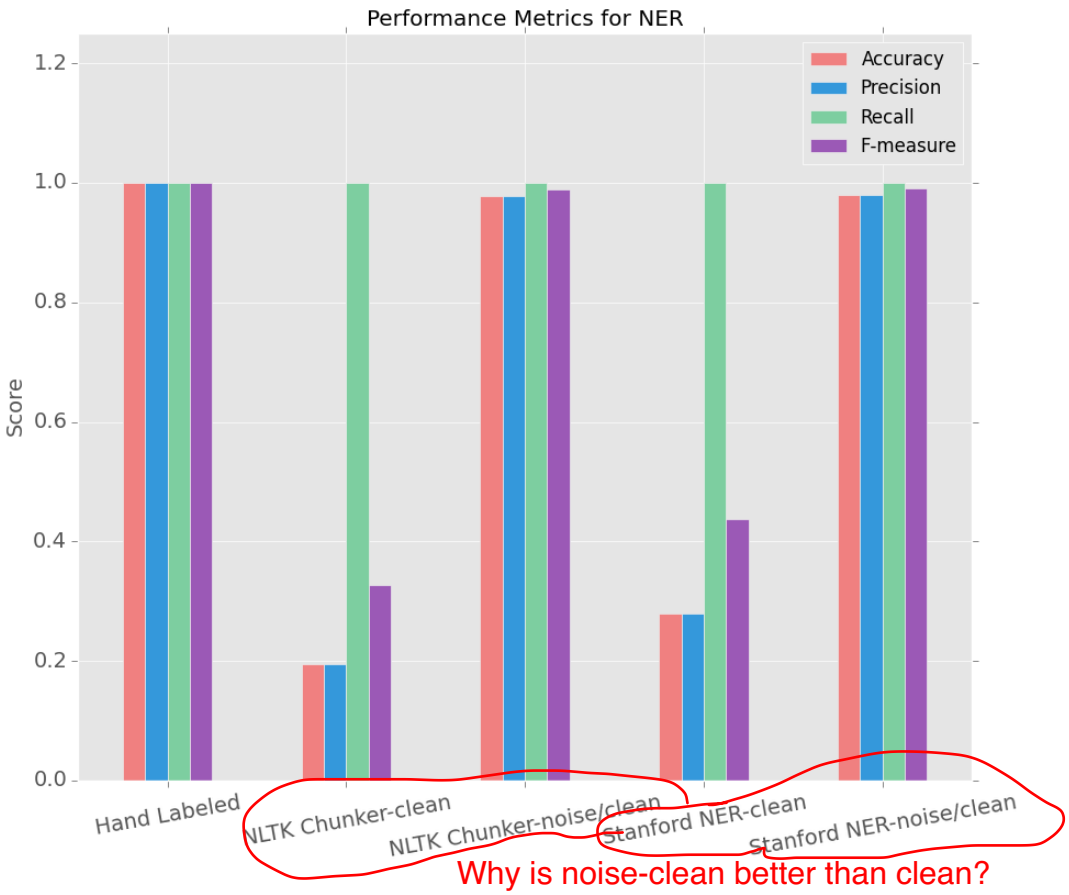


Figure 8.2: Performance Metrics for Different NER methods

Method	Total Truth	Total Run	Accuracy	Precision	Recall	f1-score
Hand Labeled	12	12	1.00	1.00	1.00	1.00
NLTK Chunker-clean	12	31	0.23	0.23	1.00	0.37
NLTK Chunker-noise/clean	31	31	1.00	1.00	1.00	1.00
Stanford NER-clean	12	16	0.69	0.69	1.00	0.81
Stanford NER-noise/clean	16	16	1.00	1.00	1.00	1.00
Ensemble	12	11	1.00	1.00	0.92	0.96

Table 8.1: Performance Metrics for Different NER methods

We can see from ~~the~~ figure 8.2 ~~above~~ that NLTK Chunker-noise/clean and Stanford NER-noise/clean is pretty good, which means the named entities extraction works the same in the noise journal and clean journal paper. So we could feed the noise journal paper directly into the system without any preprocessing, ~~it would not affect the final answer~~.

We use Accuracy, Precision, Recall, and F-measure to evaluate the performance of the method. Total truth is the number of the true LOCATIONs in the paper. Total run is the number of LOCATIONs via yellow highlighted method.

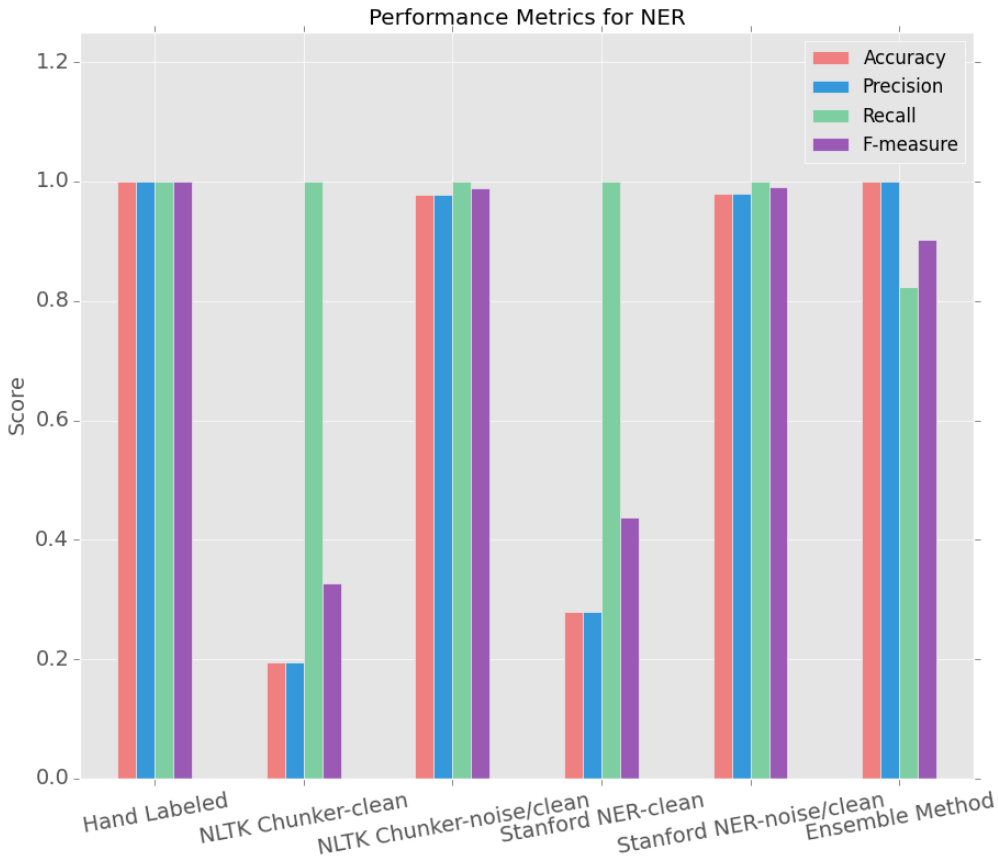


Figure 8.3: Performance Metrics for Different NER methods with Ensemble Method

A Simple Ensemble Classifier

The goal of an ensemble method [16] is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability and robustness over a single estimator. We notice the varying levels of performance by the different NER tools. Using the idea that combining the outputs from various classifiers in an ensemble method can improve the reliability of classifications, we can improve the performance of our named entity extractor tools by creating an ensemble classifier. Our ensemble classifier voting rule is very simple: Return all named entities that exist in at least two of the true positive named entity result

???

sets from our NER tools. We implement this rule using the set module. We first do an intersection operation of the NER results versus the hand labeled entities to get our "true positive" set.

Exactly as expected, we see improved performance across all performance metric scores and, in the end, get a perfect extraction of all named persons from this document.

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"
   ]
}
```


How are these performance metrics?

Figure 8.4: Performance Metrics for Different NER methods with Ensemble Method

Left data above is hand labeled location file. Right data above is the output by our system. It works pretty well.

How is this a quantified performance evaluation?

Discussion: From the results shown in figures above, we could see the clean text and original text lead to similar measures of accuracy, recall, and precision, so we do not need to clean text to extract desired named entities. The NLTK NER model and Stanford NER we trained do not work well, but the ensemble of them

the

could improve performance.

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. There are 1006 tables for 207 journal papers. The algorithm and related program are provided by Prof. Sharad Seth [1] at University of Nebraska–Lincoln.

Describe the goal of this part, where the data came from, etc.

To evaluate our program performance, we need to extract some testing tables from papers and convert it to comma-separated values (CSV) format and check the results after the tables being transferred from human-readable format to 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.

Where does this information come from?

9.1 Purpose

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. We 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). [1]

9.2 Well-Formed Table

The 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

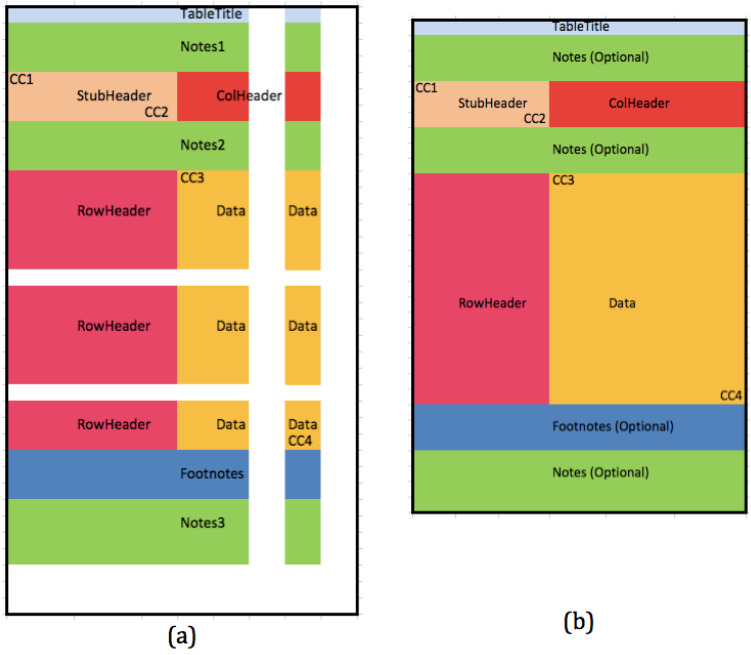


Figure 9.1: Visual WFT model: (a) with and (b) without blank rows and columns

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.

9.3 Algorithm

Is this Seth's algorithm? If so, say so.

Critical cells (CC1, CC2, CC3, CC4) delineate regions. In a WFT every critical cell must appear in the grid. As Fig. shows, 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 ri and column ci be the coordinates of critical cell CCi, a

Use italics and subscripts

WFT satisfies the following constraints: r1 r2 ¡ r3 r4 and c1 c2 ¡ c3 c4. 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.

Is this the entire algorithm? No pseudocode?

9.4 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 what?

The other 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 tables 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.

9.4.1 Well Formed Table result

Not a sentence

The Well Formed Table the program could deal with correctly.

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 9.2: Table 3 in journal paper No.105 [cite the paper]

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

Cell_ID	Row	Column	Content	Class
105_liang.eta	1	1	Soil enzyme a	tabletitle
105_liang.eta	1	2		tabletitle
105_liang.eta	1	3		tabletitle
105_liang.eta	1	4		tabletitle
105_liang.eta	1	5		tabletitle
105_liang.eta	2	1		stubheader
105_liang.eta	2	2	Exoglucanase	colheader
105_liang.eta	2	3	b-Glucosidas	colheader
105_liang.eta	2	4	b-Glucosamii	colheader
105_liang.eta	2	5	Peroxidase	colheader
105_liang.eta	3	1		rowheader
105_liang.eta	3	2	mmol h□1 g□	data
105_liang.eta	3	3		data
105_liang.eta	3	4		data
105_liang.eta	3	5		data
105_liang.eta	4	1	Kinston site	rowheader
105_liang.eta	4	2		data
105_liang.eta	4	3		data
105_liang.eta	4	4		data
105_liang.eta	4	5		data
105_liang.eta	5	1	Austrian wint	rowheader
105_liang.eta	5	2	0.240 a	data
105_liang.eta	5	3	0.854 ab	data
105_liang.eta	5	4	0.378 ab	data

Figure 9.4: Classification Table for Table 3 in Journal Paper No. 105

We could see the program could extract the information from category table and store it in relational database (Microsoft Access Here). Then users could use SQL to query the information they are interested in. And the classification table also make sense.

9.4.2 NOT Well Formed Table result

There are also some not Well Formed Tables the program could not works well.

~~Not Well Formed Table:~~ Refer to Figure 9.5

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 9.5: Table 2 in Journal Paper No. 001 [cite the paper]

We could see in the classification table, the program fails to classify the data at the bottom half table correctly, and classify the data as note.

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 9.6: Classification Table for Table 2 in Journal Paper No. 001

So the category table only contains the data from top half of the original table, but miss the data in bottom half.

???

We have 122 journal papers and they contain 500 tables. The following table shows the number of tables in each paper and the type of table the paper contain, where Y means Well Formed Table, and N indicates not Well Formed Table.

9.4.3 Discussion

a total of

We have ~~totally~~ 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. But the program could handle with

Cell_ID	RowCat_1.1	ColCat_1.1	ColCat_1.2	ColCat_1.3	DATA
001_abid&la	NT	0_10 cma	Unb	ditto	10.41
001_abid&la	NT	0_10 cma	Unb	Db	10.26
001_abid&la	NT	10_20 cma	Unb	UNb	10.92
001_abid&la	NT	10_20 cma	Unb	Db	10.32
001_abid&la	NT	20_30 cma	Unb	UNb	10.49
001_abid&la	NT	20_30 cma	Unb	Db	10.23
001_abid&la	T	0_10 cma	Unb	ditto	10.74
001_abid&la	T	0_10 cma	Unb	Db	10.98
001_abid&la	T	10_20 cma	Unb	UNb	10.24
001_abid&la	T	10_20 cma	Unb	Db	10.28
001_abid&la	T	20_30 cma	Unb	UNb	10.5
001_abid&la	T	20_30 cma	Unb	Db	10.37

Figure 9.7: Category Table for Table 2 in in Journal Paper No. 001

907, which is about 90% of all tables. ~~It works well.~~

Explain more on the experimental setup, how you evaluated it, etc.

Are your results consistent with those from Seth's paper?

How do you map identifying information from one output to the others?

Chapter 10

Database System

Start with, "We now describe the database used in this project. It was designed by the author and implemented by an undergraduate senior design team [in a footnote, list their names]".

Database is designed by myself. And the implementation and modification are done by undergraduate student team.

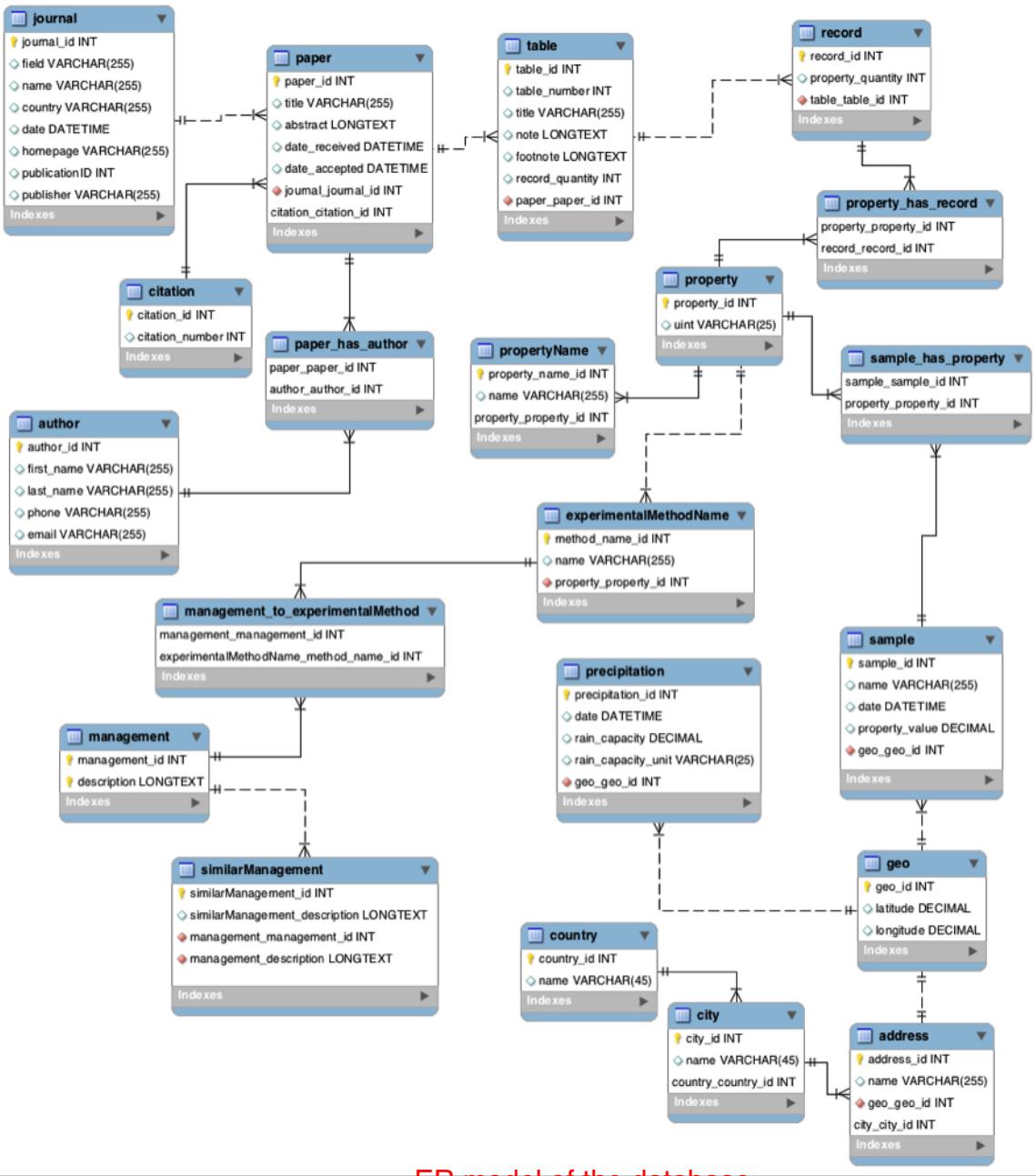
10.1 Database Design

Define, and cite a source

Tables in ER model are linked together, where all the data needed for a single query could be spread over ~~couple of~~ multiple tables, so any change in one table may have an ~~a~~effect on every other table~~s~~ in the system. It is therefore very important that every single table in the database should be designed properly, which is not only in respect to the tuples in a specific table, but also in regards to every other tables in the database. So a proper design can save ~~you~~ a tremendous amount of time and effort.

An ER model organizes data in tables (or relations). A table is made up of rows and columns. A row is also called a record (or tuple). A column is also called a field (or attribute). The relationships among the tables make them to store and retrieve data efficiently. A language called SQL (Structured Query Language) was developed to

Refer to this figure in the main text. Also, explain each table and its fields.



ER model of the database

Figure 10.1: ~~Diagram of Database Design~~

for this purpose.

~~handle with database~~. Databases are customized to fit a particular application. ~~No two customized applications are alike, and hence, no two databases are alike~~.

Keys are very important part of Relational database. They are used to establish and identify relations between tables. They also ensure that each record within a table can be uniquely identified by combination of one or more fields within a table.

10.2 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 ~~following~~ figures. 10.2-10.4

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. Cook		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 Redu	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 ir	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 h	jeq	doi:10.2134/je	Journal
26	Bromacil in Lak	Edwin A. Hebb	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 Acetylei	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 Lear2					1980	Athens	Georgia	Abstract Chan	sssaj	doi:10.2136/sss	Journal
33	Heavy Metal Cu	W. Nelson Bey	Rufus L. Chan	Bernard M. Mi			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 10.2: Journal in Database

Explain more. Is this the "Journal" table? With sample records?

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 on	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 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	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 Surfac	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

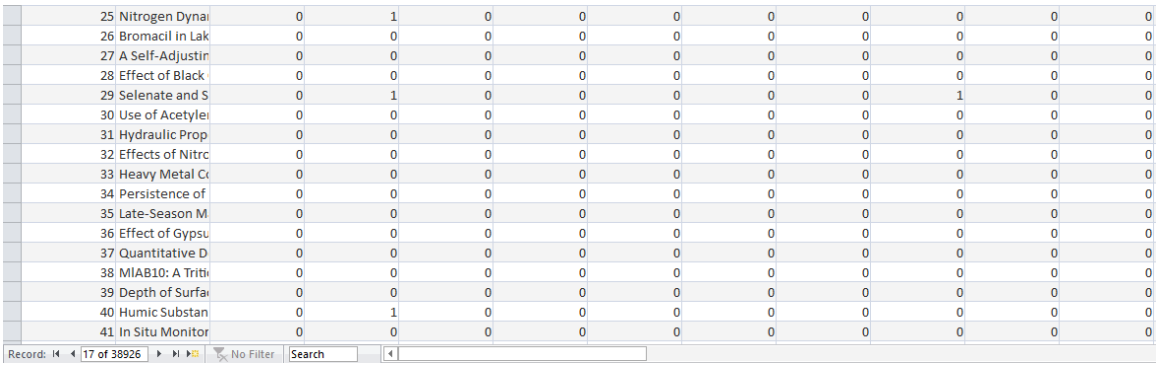
Record: 14 of 38926 No Filter Search

Figure 10.3: 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 on	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

Don't have figures span multiple pages



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 C	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 10.4: Keyword 2 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 10.1: Journal 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

Table 10.2: Keyword 1 and Keyword 2 in Database

Appendix A

Testing, 1, 2, 3, ...

This has been a test of the thesis typesetting system. Had this been an actual thesis, this would have been preceded by an actual thesis.

Bibliography

[1] Rish. "An empirical study of the naive bayes classifier". In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, pages 41–46, 2001. 5.1.4

[2] P. Domingos and M. Pazzani. "On the optimality of the simple bayesian classifier under zero-one loss". 29(2–3):103–130, 1997. 5.1.4

[3] Ned Batchelder. "Pragmatic Unicode". https://nedbatchelder.com/text/unipain.html, March 2012. 7.1.1

[4] Joel Spolsky. "The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets". https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know OCTOBER 2003. 7.1.1

[5] http://wolfprojects.altervista.org/talks/unicode-and-python-3/. 7.1.1

[6] Kumar McMillan. "Unicode In Python, Completely Demystified". http://farmdev.com/talks/unicode/, 2008. 7.1.1

[7] https://stackoverflow.com/questions/21808657/what-is-a-unicode-string. 7.1.1

[8] "Dive into Python 3". http://www.diveintopython3.net/strings.html. 7.1.1

[9] Philip Guo. "Unicode strings in Python: A basic tutorial". http://www.pgbovine.net/unicode-python.htm, November 2015. 7.1.1

[10] http://pycoders-weekly-chinese.readthedocs.io/en/latest/issue5/unipain.html. 7.1.1

[11] M. F. Porter. "An algorithm for suffix stripping". *Program: electronic library and information systems*, 14(3):130–137, 1980. 7.1.3.3

[12] Sebastian Raschka. "Naive Bayes and Text Classification: Introduction and Theory". http://sebastianraschka.com/Articles/2014_naive_bayes_1.html, October 2014. 7.1.3.4

[13] A. Zevcevic. "N-gram based text classification according to authorship". In *Student Research Workshop*, pages 145–149, 2011. 7.1.3.4

[14] V. Keselj, 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. 7.1.3.4

[15] I. Kanaris, K. Kanaris, I. Houvardas, and E. Stamatatos. "Words versus character n-grams for anti-spam filtering". 16(06):1047–1067, 2013. 7.1.3.4

[16] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent Dirichlet Allocation". 3:993–1022, 2003.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient Estimation of Word Representations in Vector Space". 2013.

List of Figures

This goes after the table of contents, not here.

List of Tables

This goes after the table of contents, not here.