Deep Learning for Extracting Adverse Drug Reactions from User Reviews

A project report submitted in partial fulfillment of requirements for the award of degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

By

B.PALLAVI (169X1A0515)

P.PRIYANKA(169X1A0596)

Y.SASIDHAR(169X1A05F5)

Under the Esteemed guidance of

Sri K. Ishthaq Ahamed
Associate Professor
Department of C.S.E.

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

2019 - 2020

Department of Computer Science and Engineering

G. PULLA REDDY ENGINEERING COLLEGE (Autonomous): KURNOOL

(Affiliated to JNTUA, ANANTAPURAMU)

CERTIFICATE

This is to certify that the preliminary project work entitled 'DEEP LEARNING FOR EXTRACTING ADVERSE DRUG REACTIONS FROM USER REVIEWS' *is a bonafide record of work carried out by*

B. PALLAVI (169X1A0515)

P. PRIYANKA(169X1A0596)

Y. SASIDHAR(169X1A05F5)

Under my guidance and supervision in fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

Sri K. Ishthaq Ahamed
Associate Professor,
Department of C.S.E ,
G. Pulla Reddy Engineering College,
Kurnool.

Dr. N. Kasiviswanath,
Professor & Head of the Department,
Department of C.S.E,
G. Pulla Reddy Engineering College,
Kurnool.

DECLARATION

We hereby declare that the project work titled "**Deep Learning for Extracting Adverse Drug Reactions from User Reviews**" is the authentic work carried out by us as students of G. PULLA REDDY ENGINEERING COLLEGE (Autonomous) Kurnool, during December, 2019 – March, 2020 and has not been submitted elsewhere for the award of degree in part or in full to any institute.

B.PALLAVI
(169X1A0515)

P.PRIYANKA
(169X1A0596)

Y.SASIDHAR
(169X1A05F5)

ACKNOWLEDGEMENT

We wish to express our deep sense of gratitude to our project guide, **Sri K. Ishthaq Ahamed** Associate Professor of CSE Department, G. Pulla Reddy Engineering College, for his immaculate guidance, constant encouragement and cooperation which have made possible to bring out this project work.

We are grateful to our project incharge, **Sri K. Ishthaq Ahamed**, Associate Professor of CSE Department, G. Pulla Reddy Engineering College, for helping us and giving us the required information needed for our project work.

We are thankful to our Head of the Department **Dr. N. Kasiviswanath**, for his whole hearted support and encouragement during the project sessions.

We are grateful to our respected Principal **Dr. B. Sreenivasa Reddy**, for providing requisite facilities and helping us in providing such a good environment.

We wish to convey our acknowledgements to all the staff members of the Computer Science Engineering Department for giving the required information needed for our project work.

Finally, we wish to thank all our friends and well-wishers who have helped us directly or indirectly during the course of this project work.

ABSTRACT

Social media is becoming increasingly popular as a platform for sharing personal health-related information. This information can be utilized for public health monitoring tasks, particularly for pharmacovigilance, via the use of natural language processing (NLP) techniques. However, the language in social media is highly informal, and user-expressed medical concepts are often nontechnical, descriptive, and challenging to extract. There has been limited progress in addressing these challenges, and thus far, advanced machine learning-based NLP techniques have been underutilized.

In this work, we focus on the identification of adverse drug reactions (ADRs). ADRs are the main source for the post-marketing surveillance. Traditionally, reports about ADRs have been identified using complaints from individual's patients and their doctors and reports on clinical trials. Currently, drug reactions can be extracted from user reviews provide on the internet, and processing this information in a automatic way is a novel and exciting approach to personalized medicine for each word. Our objective is to design a novel approach for extracting adverse drug reactions from the user reviews: a combination of a bidirectional LSTM –based recurrent neural network and CRF that operates on the scores extracted by this neural networks.

The proposed CNN-based DDI extraction system uses and evaluates the DDI corpus of the 2013 DDI Extraction challenge, which is composed of 730 DrugBank documents and 175 MEDLINE abstracts about DDIs. The results of extraction of ADRs are presented detail in this paper.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1
INTRODUCTION

1 .INTRODUCTION

1.1 INTRODUCTION

An adverse drug reaction (ADR) is an unwanted, undesirable effect of a medication that occurs during usual clinical use. Adverse drug reaction occurs almost daily in health care institutions and can adversely affect a patient’s quality of life, new studies on text mining applications increasingly employ non-standard sources of information to obtain data related to health conditions, and interactions between different drugs. Based on the user reviews given by themselves in the social media posts and medical related websites such rich source of information have been successfully used, for monitor adverse drug reactions through the user complaints. In this work, we focus on the identification of adverse drug reactions (ADRs). ADRs are the main source for the post-marketing surveillance. Traditionally, reports about ADRs have been identified using complaints from individual’s patients and their doctors and reports on clinical trials. Currently, drug reactions can be extracted from user reviews provide on the internet, and processing this information in a automatic way is a novel and exciting approach to personalized medicine for each word. On top of this RNN, We use a sequential CRF to jointly decode the word labels for the each and every sentence. Similarly in past a successful implementation was proposed for two sequence labelling tasks: Past of speech (POS) tagging and the name entity recognition (NER). In this model we evaluate ADR extraction on an annotated corpus CADEC (CSIRO Adverse Drug Event Corpus). The CADEC annotated text contains of 1250 medical forum posts taken from AskaPatient.com[7], where each post has been manually annotated with mention of ADEs. Our results shows the join model of RNN and CRF improves the performance of state- of –art here we introduce the join model that combines CRF and RNN model for extraction of ADR by conducting the empirical logic of this model on benchmark dataset and to show that the experimental result will improves our state of art performance.

1.2 MOTIVATION

Pre-marketing clinical trials are used to identify adverse events (such as side effects) of drugs before they are introduced in the market. However such trials will inevitably fail to find many of the Adverse Drug Reactions (ADRs) associated with a drug due to the size and time constraints of such trials. In particular, they are unlikely to find rarer ADRs, those which only occur in combination with other drugs. The increasing numbers of ADR-related deaths indicate that post-marketing ADRs are an important public health problem, indicating the need for continuous automated ADR surveillance.

1.3 PROBLEM DEFINITION

To extract Adverse Drug Reactions (ADRs) based on User Reviews, DrugBank and MEDLINE.
To notify people regarding Drug-Drug Interactions.
To caution people not to use drugs in combination which results in ADRs.

1.4 OBJECTIVE OF THE PROJECT

To propose a novel approach for extracting adverse drug reactions from the user reviews: a combination of a bidirectional LSTM –based recurrent neural network and CRF that operates on the scores extracted by this neural networks

1.5 LIMITATIONS OF THE PROJECT

If the users stop giving reviews, then there won't be anything to say about the ADRs.

1.6 ORGANIZATION OF THE REPORT

- **Introduction:** Contains Introduction to the project, Motivation, Problem Definition, Objective of the project, Limitations of the project, Software and Hardware specifications and Learning from Data
- **Literature Survey:** Contains the referenced IEEE paper description.
- **System Analysis:** Contains the related work of the project like biomedical text mining, Pharmacovigilance from social media, CRF, Types of RNN, CNN .
- **System Design:** Contains the system architecture and the datasets used to build the project.
- **Implementation:** Includes Process Steps and Project Code
- **Results:** Includes the output of the trained data and project output snippet.
- **Conclusion and Future Enhancements**
- **References**

1.7 SOFTWARE AND HARDWARE SPECIFICATIONS

Hardware Requirements:

GPU Processor

4GB RAM

Keyboard, mouse, monitor

Active Internet Connection

Software Requirements

Operating System: Windows 7 or higher or Linux

Programming Languages: PYTHON 3.7 IDLE or Anaconda

1.8 LEARNING FROM DATA

If you show a picture to a three-year-old and ask if there is a tree in it, you will likely get the correct answer. If you ask a thirty-year-old what the definition of a tree is, you will likely get an inconclusive answer. We didn't learn what a tree is by studying the mathematical definition of trees. We learned it by looking at trees. In other words, we learned from 'data'.

Learning from data is used in situations where we don't have an analytic solution, but we do have data that we can use to construct an empirical solution. This premise covers a lot of territory, and indeed learning from data is one of the most widely used techniques in science, engineering, and economics, among other fields.

1.8.1 Problem Setup

What do financial forecasting, medical diagnosis, computer vision, and search engines have in common? They all have successfully utilized learning from data. The repertoire of such applications is quite impressive. Let us open the discussion with a real-life application to see how learning from data works.

Consider the problem of predicting how a movie viewer would rate the various movies out there. This is an important problem if you are a company that rents out movies, since you want to recommend to different viewers the movies they will like. Good recommender systems are so important to business that the movie rental company Netflix offered a prize of one million dollars to anyone who could improve their recommendations by a mere 10%.

The main difficulty in this problem is that the criteria that viewers use to rate movies are quite complex. Trying to model those explicitly is no easy task, so it may not be possible to come up with an analytic solution. However, we know that the historical rating data reveal a lot about how people rate movies, so we may be able to construct a good empirical solution.

There is a great deal of data available to movie rental companies, since they often ask their viewers to rate the movies that they have already seen.

Figure 1.1 illustrates a specific approach that was widely used in the million-dollar competition. Here is how it works. You describe a movie as a long array of different factors, e.g., how much comedy is in it, how complicated is the plot, how handsome is the lead actor, etc. Now, you describe each viewer with corresponding factors; how much do they like comedy, do they prefer simple or complicated plots, how important are the looks of the lead actor, and so on. How this viewer will rate that movie is now estimated based on the match/mismatch of these factors. For example, if the movie is pure comedy and the viewer hates comedies, the chances are he won't like it. If you take dozens of these factors describing many facets of a movie's content and a viewer's taste, the conclusion based on matching all the factors will be a good predictor of how the viewer will rate the movie.

The power of learning from data is that this entire process can be automated, without any need for analysing movie content or viewer taste. To do so, the learning algorithm 'reverse-engineers' these factors based solely on previous ratings. It starts with random factors, then tunes these factors to make them more and more aligned with how viewers have rated movies before, until they are ultimately able to predict how viewers rate movies in general. The factors we end up with may not be as intuitive as 'comedy content', and in fact can be quite subtle or even incomprehensible. After all, the algorithm is only trying to find the best way to predict how a viewer would rate a movie, not necessarily explain to us how it is done. This algorithm was part of the winning solution in the million-dollar competition.

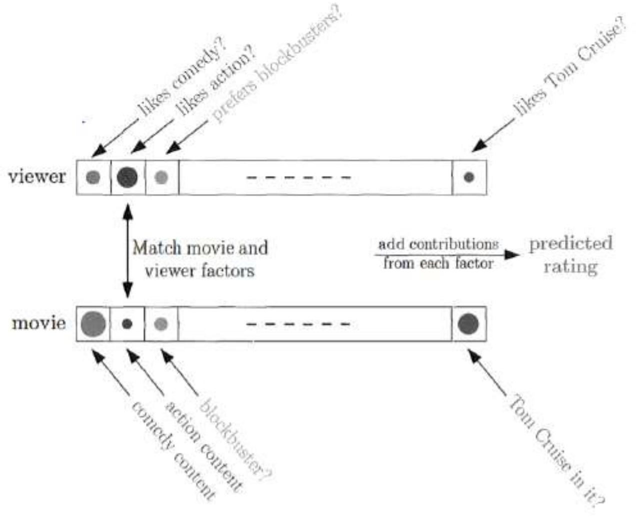


Fig 1.1 Approach used in million-dollar competition

Components of Learning

The movie rating application captures the essence of learning from data, and so do many other applications from vastly different fields. In order to abstract the common core of the learning problem.

Suppose that a bank receives thousands of credit card applications every day, and it wants to automate the process of evaluating them. Just as in the case of movie ratings, the bank knows of no magical formula that can pinpoint when credit should be approved, but it has a lot of data. This calls for learning from data, so the bank uses historical records of previous customers to figure out a good formula for credit approval.

Each customer record has personal information related to credit, such as annual salary, years in residence, outstanding loans, etc. The record also keeps track of whether approving credit for that customer was a good idea, i.e., did the bank make money on that customer. This data guides the construction of a successful formula for credit approval that can be used on future applicants.

Let us give names and symbols to the main components of this learning problem. There is the input x (customer information that is used to make a credit decision) , the unknown target function f: X -- Y (ideal formula for credit approval) , where X is the input space (set of all possible inputs x) , and Y is the output space (set of all possible outputs, in this case just a yes/no decision)

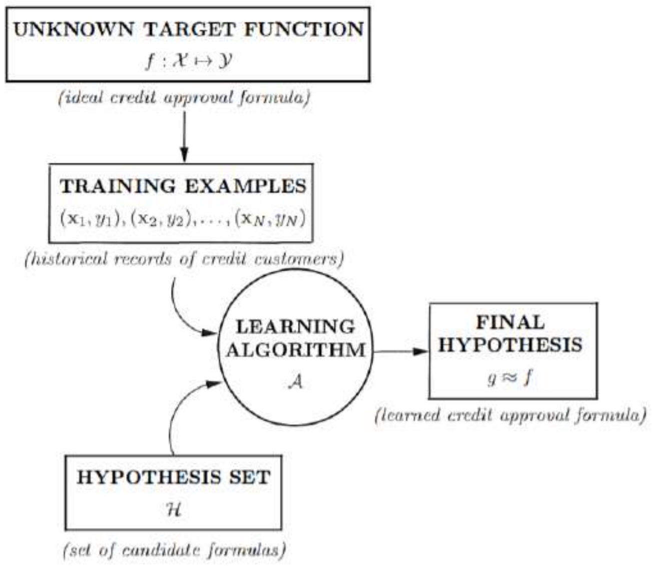

UNKNOWN TARGET FUNCTION
$f : \mathcal{X} \mapsto \mathcal{Y}$
(ideal credit approval formula)
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
(historical records of credit customers)
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g \approx f$
(learned credit approval formula)
HYPOTHESIS SET
$\mathcal{H}$
(set of candidate formulas)

Fig 1.2 Components of the learning problem

Finally, there is the learning algorithm that uses the data set D to pick a formula g: X -- Y that approximates f. The algorithm chooses g from a set of candidate formulas under consideration, which we call the hypothesis set H. For instance, H could be the set of all linear formulas from which the algorithm would choose the best linear fit to the data, as we will introduce later in this section.

When a new customer applies for credit, the bank will base its decision on g (the hypothesis that the learning algorithm produced) , not on f (the ideal target function which remains unknown) . The decision will be good only to the extent that g faithfully replicates f. To achieve that, the algorithm chooses g that best matches f on the training examples of previous customers, with the hope that it will continue to match f on new customers. Whether or not this hope is justified remains to be seen. Figure 1.2 illustrates the components of the learning problem.

We will use the setup in Figure 1.2 as our definition of the learning problem. Later on, we will consider a number of refinements and variations to this basic setup as needed. However, the essence of the problem will remain the same. There is a target to be learned. It is unknown to us. We have a set of examples generated by the target. The learning algorithm uses these examples to look for a hypothesis that approximates the target.

A Simple Learning Model

Let us consider the different components of Figure 1.2. Given a specific learning problem, the target function and training examples are dictated by the problem. However, the learning algorithm and hypothesis set are not. These are solution tools that we get to choose. The hypothesis set and learning algorithm are referred to informally as the learning model.

Here is a simple model. Let $X = R^d$ be the input space, where R^d is the d-dimensional Euclidean space, and let $Y = \{ + 1, -1\}$ be the output space, denoting a binary (yes/no) decision. In our credit example, different coordinates of the input vector $x \in R^d$ correspond to salary, years in residence, outstanding debt, and the other data fields in a credit application. The binary output y corresponds to approving or denying credit. We specify the hypothesis set H through a functional form that all the hypotheses $h \in H$ share. The functional form h(x) that we choose here gives different weights to the different coordinates of x, reflecting their relative importance in the credit decision. The weighted coordinates are then combined to form a 'credit score' and the result is compared to a threshold value. If the applicant passes the threshold, credit is approved; if not, credit is denied:

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold,}$$
$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold.}$$

This formula can be written more compactly

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right)$$

where xi,⋯ ,xd are the components of the vector x; h(x) = +1 means 'approve credit' and h(x) = -1 means 'deny credit'; sign(s) = +1 if s> 0 and sign(s) = -1 if s < 0.1 The weights are w1, ·· · , wd, and the threshold is determined by the bias term b since in Equation (1.1), credit is approved if

$$\sum_{i=1}^{d} w_i x_i > b$$

This model of His called the perceptron, a name that it got in the context of artificial intelligence. The learning algorithm will search H by looking for weights and bias that perform well on the data set. Some of the weights w1, · · · , Wd may end up being negative, corresponding to an adverse effect on credit approval. For instance, the weight of the 'outstanding debt' field should come out negative since more debt is not good for credit. The bias value b may end up being large or small, reflecting how lenient or stringent the bank should be in extending credit. The optimal choices of weights and bias define the final hypothesis g ∈ H that the algorithm produces.

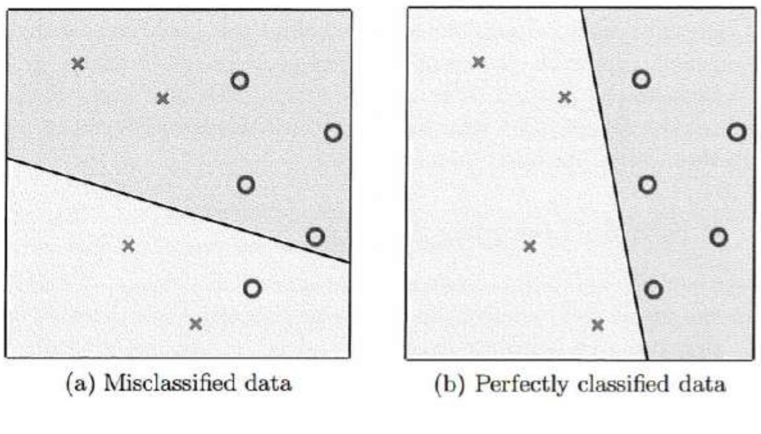


Fig 1.3 Classification of Data

To simplify the notation of the perceptron formula, we will treat the bias b as a weight wo = b and merge it with the other weights into one vector w = [w0, w1, · · · , wd]T, where T denotes the transpose of a vector, so w is a column vector. We also treat x as a column vector and modify it to become x = [x0, xi, · · · , xd]T, where the added coordinate x0 is fixed at x0 = 1. Formally speaking, the input space is now

$$\mathcal{X} = \{1\} \times \mathbb{R}^d = \{[x_0, x_1, \cdots, x_d]^{\mathrm{T}} \mid x_0 = 1, x_1 \in \mathbb{R}, \cdots, x_d \in \mathbb{R}\}.$$

With this convention, wTx = $\sum_{i=0}^{d} WiXi$ and so Equation (1.1) can be rewritten in vector form as

$$h(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

We now introduce the perceptron learning algorithm (PLA) . The algorithm will determine what w should be, based on the data. Let us assume that the data set is linearly separable, which means that there is a vector w that makes (1.2) achieve the correct decision h(xn) = Yn on all the training examples, as shown in Figure 1.3.

Our learning algorithm will find this w using a simple iterative method. Here is how it works. At iteration t, where t = 0, 1, 2, ... , there is a current value of the weight vector, call it w(t). The algorithm picks an example from (x1 , Y1) · · · (xN, YN) that is currently misclassified, call it (x(t) , y(t)), and uses it to update w(t) . Since the example is misclassified, we have y (t) # sign(wT(t)x(t)). The update rule is

W(t+1) = w(t) + y(t)x(t)

This rule moves the boundary in the direction of classifying x(t) correctly, as depicted in the figure above. The algorithm continues with further iterations until there are no longer misclassified examples in the data set.

Although the update rule in (1.3) considers only one training example at a time and may 'mess up' the classification of the other examples that are not involved in the current iteration, it turns out that the algorithm is guaranteed to arrive at the right solution in the end. The proof is the subject of Problem 1.3. The result holds regardless of which example we choose from among the misclassified examples in (x1, Y1) · · · (xN, YN) at each iteration, and regardless of how we initialize the weight vector to start the algorithm. For simplicity, we can pick one of the misclassified examples at random (or cycle through the examples and always choose the first misclassified one) , and we can initialize w(O) to the zero vector.

Within the infinite space of all weight vectors, the perceptron algorithm manages to find a weight vector that works, using a simple iterative process. This illustrates how a learning algorithm can effectively search an infinite hypothesis set using a finite number of simple steps. This feature is characteristic of many techniques that are used in learning, some of which are far more sophisticated than the perceptron learning algorithm.

The perceptron learning algorithm succeeds in achieving its goal; finding a hypothesis that classifies all the points in the data set D = { (x 1, y1) · · · (x N, y N)} correctly. Does this mean that this hypothesis will also be successful in classifying new data points that are not in V? This turns out to be the key question in the theory of learning, a question that will be thoroughly examined in this book.

Learning versus Design:

So far, we have discussed what learning is. Now, we discuss what it is not. The goal is to distinguish between learning and a related approach that is used for similar problems. While learning is based on data, this other approach does not use data. It is a 'design' approach based on specifications, and is often discussed alongside the learning approach in pattern recognition literature.

Consider the problem of recognizing coins of different denominations, which is relevant to vending machines, for example. We want the machine to recognize quarters, dimes, nickels and pennies. We will contrast the 'learning from data' approach and the 'design from specifications' approach for this problem. We assume that each coin will be represented by its size and mass, a two-dimensional input.

In the learning approach, we are given a sample of coins from each of the four denominations and we use these coins as our data set. We treat the size and mass as the input vector, and the denomination as the output. Figure shows what the data set may look like in the input space. There is some variation of size and mass within each class, but by and large coins of the same denomination cluster together. The learning algorithm searches for a hypothesis that classifies the data set well. If we want to classify a new coin, the machine measures its size and mass, and then classifies it according to the learned .

In the design approach, we call the United States Mint and ask them about the specifications of different coins. We also ask them about the number of coins of each denomination in circulation, in order to get an estimate of the relative frequency of each coin. Finally, we make a physical model of the variations in size and mass due to exposure to the elements and due to errors in measurement. We put all of this information together and compute the full joint probability distribution of size, mass, and coin denomination. Once we have that joint distribution, we can construct the optimal decision rule to classify coins based on size and mass. The rule chooses the denomination that has the highest probability for a given size and mass, thus achieving the smallest possible probability of error.

The main difference between the learning approach and the design approach is the role that data plays. In the design approach, the problem is well specified and one can analytically derive f without the need to see any data. In the learning approach, the problem is much less specified, and one needs data to pin down what f is.

Both approaches may be viable in some applications, but only the learning approach is possible in many applications where the target function is unknown. We are not trying to compare the utility or the performance of the two approaches. We are just making the point that the design approach is distinct from learning. This book is about learning.

1.8.2 Types of Learning

The basic premise of learning from data is the use of a set of observations to uncover an underlying process. It is a very broad premise, and difficult to fit into a single framework. As a result, different learning paradigms have arisen to deal with different situations and different assumptions. In this section, we introduce some of these paradigms.

The learning paradigm that we have discussed so far is called supervised learning. It is the most studied and most utilized type of learning, but it is not the only one. Some variations of supervised learning are simple enough to be accommodated within the same framework. Other variations are more profound and lead to new concepts and techniques that take on lives of their own. The most important variations have to do with the nature of the data set.

Supervised Learning

When the training data contains explicit examples of what the correct output should be for given inputs, then we are within the supervised learning setting that we have covered so far. Consider the hand-written digit recognition problem (task (b) of Exercise 1.1). A reasonable data set for this problem is a collection of images of hand-written digits, and for each image, what the digit actually is. We thus have a set of examples of the form (image , digit). The learning is supervised in the sense that some 'supervisor' has taken the trouble to look at each input, in this case an image, and determine the correct output, in this case one of the ten categories {O, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

While we are on the subject of variations, there is more than one way that a data set can be presented to the learning process. Data sets are typically created and presented to us in their entirety at the outset of the learning process. For instance, historical records of customers in the credit-card application, and previous movie ratings of customers in the movie rating application, are already there for us to use. This protocol of a 'ready' data set is the most common in practice, and it is what we will focus on in this book. However, it is worth noting that two variations of this protocol have attracted a significant body of work.

One is active learning, where the data set is acquired through queries that we make. Thus, we get to choose a point x in the input space, and the supervisor reports to us the target value for x. As you can see, this opens the possibility for strategic choice of the point x to maximize its information value, similar to asking a strategic question in a game of 20 questions.

Another variation is called online learning, where the data set is given to the algorithm one example at a time. This happens when we have streaming data that the algorithm has to process 'on the run'. For instance, when the movie recommendation system discussed in Section 1.1 is deployed, online learning can process new ratings from current users and movies. Online learning is also useful when we have limitations on computing and storage that preclude us from processing the whole data as a batch. We should note that online learning can be used in different paradigms of learning, not just in supervised learning.

Unsupervised Learning

In the unsupervised setting, the training data does not contain any output information at all. We are just given input examples xi, · ·· , XN. You may wonder how we could possibly learn anything from mere inputs. Consider the coin classification problem that we discussed earlier in Figure. Suppose that we didn't know the denomination of any of the coins in the data set. This unlabelled data is shown in Figure. We still get similar clusters, but they are now unlabelled so all points have the same 'color'. The decision regions in unsupervised learning may be identical to those in supervised learning, but without the labels. However, the correct clustering is less obvious now, and even the number of clusters may be ambiguous.

Nonetheless, this example shows that we can learn something from the inputs by themselves. Unsupervised learning can be viewed as the task of spontaneously finding patterns and structure in input data. For instance, if our task is to categorize a set of books into topics, and we only use general properties of the various books, we can identify books that have similar properties and put them together in one category, without naming that category

Unsupervised learning can also be viewed as a way to create a higher level representation of the data. Imagine that you don't speak a word of Spanish, but your company will relocate you to Spain next month. They will arrange for Spanish lessons once you are there, but you would like to prepare yourself a bit before you go. All you have access to is a Spanish radio station. For a full month, you continuously bombard yourself with Spanish; this is an unsupervised learning experience since you don't know the meaning of the words. However, you gradually develop a better representation of the language in your brain by becoming more tuned to its common sounds and structures. When you arrive in Spain, you will be in a better position to start your Spanish lessons. Indeed, unsupervised learning can be a precursor to supervised learning. In other cases, it is a stand-alone technique.

1.8.3 Error and Noise

We close this chapter by revisiting two notions in the learning problem in order to bring them closer to the real world. The first notion is what approximation means when we say that our hypothesis approximates the target function well. The second notion is about the nature of the target function. In many situations, there is noise that makes the output of f not uniquely determined by the input. What are the ramifications of having such a 'noisy' target on the learning problem?

Error Measures

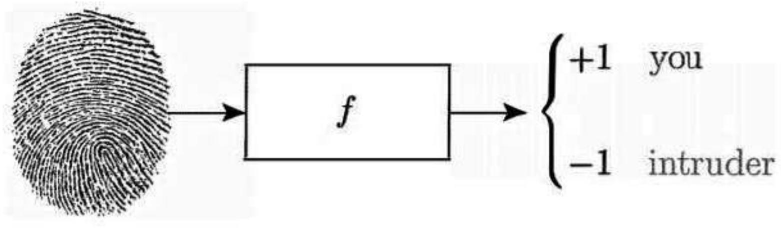
Learning is not expected to replicate the target function perfectly. The final hypothesis g is only an approximation of f. To quantify how well g approximates f, we need to define an error measure3 that quantifies how far we are from the target.

The choice of an error measure affects the outcome of the learning process. Different error measures may lead to different choices of the final hypothesis, even if the target and the data are the same, since the value of a particular error measure may be small while the value of another error measure in the same situation is large. Therefore, which error measure we use has consequences for what we learn. What are the criteria for choosing one error measure over another? We address this question here. First, let's formalize this notion a bit. An error measure quantifies how well each hypothesis h in the model approximates the target function f,

$$\text{Error} = E(h, f).$$

While E(h, f) is based on the entirety of h and f, it is almost universally defined based on the errors on individual input points x. If we define a pointwise error measure e(h(x), f(x)), the overall error will be the average value of this pointwise error. So far, we have been working with the classification error e(h(x) , f(x)) = [h(x) ≠ f(x)].

Example 1.1 (Fingerprint verification). Consider the problem of verifying that a fingerprint belongs to a particular person. What is the appropriate error measure?



The target function takes as input a fingerprint, and returns + 1 if it belongs to the right person, and -1 if it belongs to an intruder.

There are two types of error that our hypothesis h can make here. If the correct person is rejected (h = -1 but f = +1) , it is called false reject, and if an incorrect person is accepted (h = +1 but f = -1), it is called false accept.

		f	
		+1	−1
h	+1	no error	false accept
	−1	false reject	no error

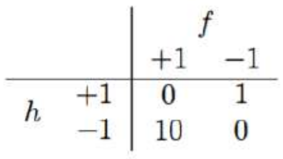
How should the error measure be defined in this problem? If the right person is accepted or an intruder is rejected, the error is clearly zero. We need to specify the error values for a false accept and for a false reject. The right values depend on the application.

Consider two potential clients of this fingerprint system. One is a supermarket who will use it at the checkout counter to verify that you are a member of a discount program. The other is the CIA who will use it at the entrance to a secure facility to verify that you are authorized to enter that facility.

For the supermarket, a false reject is costly because if a customer gets wrongly rejected, she may be discouraged from patronizing the supermarket in the future. All future revenue from this annoyed customer is lost. On the other hand, the cost of a false accept is minor. You just gave away a discount to someone who didn't deserve it, and that person left their fingerprint in your system they must be bold indeed.

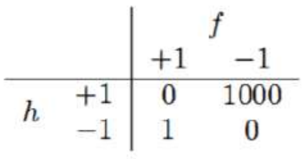
For the CIA, a false accept is a disaster. An unauthorized person will gain access to a highly sensitive facility. This should be reflected in a much higher cost for the false accept. False rejects, on the other hand, can be tolerated since authorized persons are employees (rather than customers as with the supermarket). The inconvenience of retrying when rejected is just part of the job, and they must deal with it.

The costs of the different types of errors can be tabulated in a matrix. For our examples, the matrices might look like:



		f	
		$+1$	-1
h	$+1$	0	1
	-1	10	0

Supermarket



		f	
		$+1$	-1
h	$+1$	0	1000
	-1	1	0

CIA

The moral of this example is that the choice of the error measure depends on how the system is going to be used, rather than on any inherent criterion that we can independently determine during the learning process. However, this ideal choice may not be possible in practice for two reasons. One is that the user may not provide an error specification, which is not uncommon. The other is that the weighted cost may be a difficult objective function for optimizers to work with. Therefore, we often look for other ways to define the error measure, sometimes with purely practical or analytic considerations in mind. We have already seen an example of this with the simple binary error used in this chapter, and we will see other error measures in later chapters.

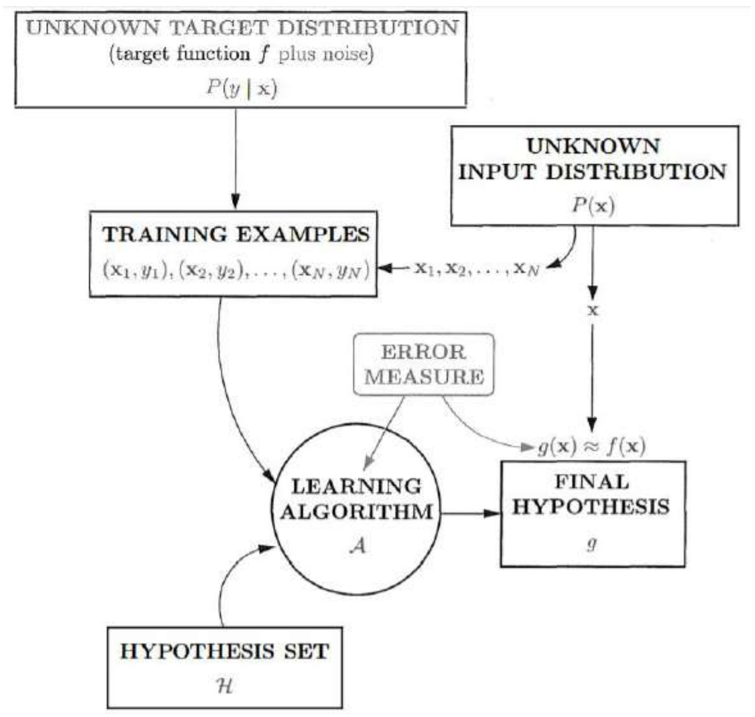

UNKNOWN TARGET DISTRIBUTION
(target function f plus noise)
$P(y \mid \mathbf{x})$
UNKNOWN INPUT DISTRIBUTION
$P(\mathbf{x})$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
$\mathbf{x}$
ERROR MEASURE
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
g
HYPOTHESIS SET
$\mathcal{H}$

Fig 1.4 the general (supervised) learning problem

Noisy Targets

In many practical applications, the data we learn from are not generated by a deterministic target function. Instead, they are generated in a noisy way such that the output is not uniquely determined by the input. For instance, in the credit-card example we presented in Section 1.1, two customers may have identical salaries, outstanding loans, etc., but end up with different credit behavior. Therefore, the credit 'function' is not really a deterministic function, but a noisy one.

This situation can be readily modelled within the same framework that we have. Instead of y = f(x) , we can take the output y to be a random variable that is affected by, rather than determined by, the input x. Formally, we have a target distribution P(y I x) instead of a target function y = f (x). A data point (x, y) is now generated by the joint distribution P(x, y) = P(x)P(y | x) . One can think of a noisy target as a deterministic target plus added noise. If y is real-valued for example, one can take the expected value of y given x to be the deterministic f (x) , and consider y - f (x) as pure noise that is added to f.

1.8.4 Linear Classification

The linear model for classifying data into two classes uses a hypothesis set of linear classifiers, where each h has the form

$$h_1(x) = sign(\omega^T x)$$

for some column vector w ∈ Rd+l , where d is the dimensionality of the input space, and the added coordinate x0 = 1 corresponds to the bias 'weight' w0 (recall that the input space X = {1} x Rd is considered d-dimensional since the added coordinate x0 = 1 is fixed) . We will use h and w interchangeably to refer to the hypothesis when the context is clear. When we left Chapter 1, we had two basic criteria for learning:

1. Can we make sure that Eout (g) is close to Ein(g)? This ensures that what we have learned in sample will generalize out of sample.
2. 2. Can we make Ein (g) small? This ensures that what we have learned in sample is a good hypothesis.

The first criterion was studied in Chapter 2. Specifically, the VC dimension of the linear model is only d + 1 (Exercise 2.4) . Using the VC generalization bound (2.12), and the bound (2.10) on the growth function in terms of the VC dimension, we conclude that with high probability,

$$E_{0ut}(g) = E_{in}(g) + 0\sqrt{\frac{d}{N} ln\, N}$$

Thus, when N is sufficiently large, Ein and Eout will be close to each other (see the definition of 0(-) in the Notation table) , and the first criterion for learning is fulfilled.

In Chapter 1, we introduced the perceptron learning algorithm (PLA) . Start with an arbitrary weight vector w(O) . Then, at every time step t 2: 0, select any misclassified data point (x(t) , y(t)), and update w(t) as follows:

W(t+1) = w(t) + y(t)x(t)

The intuition is that the update is attempting to correct the error in classifying x(t) . The remarkable thing is that this incremental approach of learning based on one data point at a time works. As discussed in Problem 1.3, it can be proved that the PLA will eventually stop updating, ending at a solution wPLA with Ein (wPLA) = 0. Although this result applies to a restricted setting (linearly separable data) , it is a significant step. The PLA is clever it doesn't na1vely test every linear hypothesis to see if it (the hypothesis) separates the data; that would take infinitely long. Using an iterative approach, the PLA manages to search an infinite hypothesis set and output a linear separator in (provably) finite time.

As far as PLA is concerned, linear separability is a property of the data, not the target. A linearly separable D could have been generated either from a linearly separable target, or (by chance) from a target that is not linearly separable. The convergence proof of PLA guarantees that the algorithm will work in both these cases, and produce a hypothesis with Ein = 0. Further, in both cases, you can be confident that this performance will generalize well out of sample, according to the VC bound.

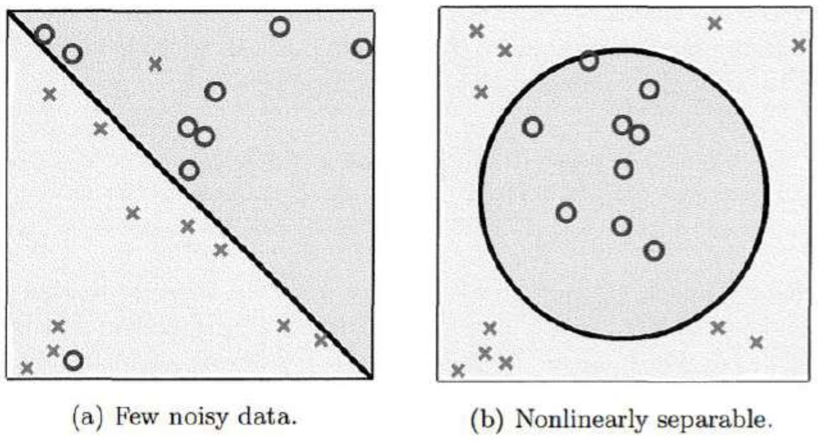


Fig 1.5 Noisy data and nonlinearly separable data

Non-Separable Data

The situation in Figure 3.l(a) is actually encountered very often: even though a linear classifier seems appropriate, the data may not be linearly separable because of outliers or noise. To find a hypothesis with the minimum Ein, we need to solve the combinatorial optimization problem:

$$\min_{w \in R^{d+1}} \frac{1}{N} \sum_{n=1}^{N} [sign(w^T x_n) \neq y_n]$$

The difficulty in solving this problem arises from the discrete nature of both sign(·) and [-]. In fact, minimizing Ein(w) in (3.2) in the general case is known to be NP-hard, which means there is no known efficient algorithm for it, and if you discovered one, you would become really, really famous. Thus, one has to resort to approximately minimizing Ein.

One approach for getting an approximate solution is to extend PLA through a simple modification into what is called the pocket algorithm. Essentially, the pocket algorithm keeps 'in its pocket' the best weight vector encountered up to iteration t in PLA. At the end, the best weight vector will be reported as the final hypothesis. This simple algorithm is shown below.

We sample some digits from the US Postal Service Zip Code Database. These 16 x 16 pixel images are preprocessed from the scanned handwritten zip codes. The goal is to recognize the digit in each image. We alluded to this task in part (b) of Exercise 1.1. A quick look at the images reveals that this is a non-trivial task (even for a human) , and typical human Eout is about 2.5%. Common confusion occurs between the digits { 4, 9} and {2, 7}. A machine-learned hypothesis which can achieve such an error rate would be highly desirable.

While the digits can be roughly separated by a line in the plane representing these two features, there are poorly written digits (such as the '5' depicted in the top-left corner) that prevent a perfect linear separation.

We now run PLA and pocket on the data set and see what happens. Since the data set is not linearly separable, PLA will not stop updating. In fact, as can be seen in Figure 3.2(a), its behavior can be quite unstable. When it is forcibly terminated at iteration 1, 000, PLA gives a line that has a poor Ein = 2.24% and Eout = 6.37%. On the other hand, if the pocket algorithm is applied to the same data set, as shown in Figure 3.2(b) , we can obtain a line that has a better Ein = 0.45% and a better Eout = 1 .89%**.**

1.8.5 Overfitting

Paraskavedekatriaphobia1 (fear of Friday the 13th) , and superstitions in general, are perhaps the most illustrious cases of the human ability to overfit. Unfortunate events are memorable, and given a few such memorable events, it is natural to try and find an explanation. In the future, will there be more unfortunate events on Friday the 13th's than on any other day?

Overfitting is the phenomenon where fitting the observed facts (data) well no longer indicates that we will get a decent out-of-sample error, and may actually lead to the opposite effect. You have probably seen cases of overfitting when the learning model is more complex than is necessary to represent the target function. The model uses its additional degrees of freedom to fit idiosyncrasies in the data (for example, noise), yielding a final hypothesis that is inferior. Overfitting can occur even when the hypothesis set contains only functions which are far simpler than the target function, and so the plot thickens.

The ability to deal with overfitting is what separates professionals from amateurs in the field of learning from data. We will cover three themes: When does overfitting occur? What are the tools to combat overfitting? How can one estimate the degree of overfitting and 'certify' that a model is good, or better than another? Our emphasis will be on techniques that work well in practice.

When does Overfitting Occurs?

Overfitting literally means "Fitting the data more than is warranted." The main case of overfitting is when you pick the hypothesis with lower Ein, and it results in higher Eout. This means that Ein alone is no longer a good guide for learning. Let us start by identifying the cause of overfitting.

Consider a simple one-dimensional regression problem with five data points. We do not know the target function, so let's select a general model, maximizing our chance to capture the target function.

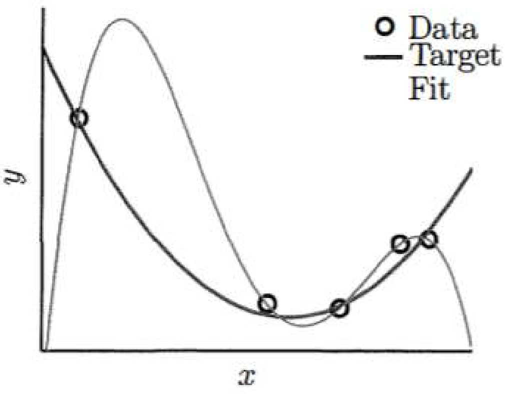


Fig 1.6 Typical overfitting scenario

Since 5 data points can be fit by a 4th order polynomial, we select 4th order polynomials. The result is shown on the right. The target function is a 2nd order polynomial (blue curve), with a little added noise in the data points. Though the target is simple, the learning algorithm used the full power of the 4th order polynomial to fit the data exactly, but the result does not look anything like the target function. The data has been 'overfit.' The little noise in the data has misled the learning, for if there were no noise, the fitted red curve would exactly match the target. This is a typical overfitting scenario, in which a complex model uses its additional degrees of freedom to 'learn' the noise.

The fit has zero in-sample error but huge out-of-sample error, so this is a case of bad generalization a likely outcome when overfitting is occurring. However, our definition of overfitting goes beyond bad generalization for any given hypothesis. Instead, overfitting applies to a process: in this case, the process of picking a hypothesis with lower and lower Ein resulting in higher and higher Eout.

A Case Study: Overfitting with Polynomials

Let's dig deeper to gain a better understanding of when overfitting occurs. We will illustrate the main concepts using data in one-dimension and polynomial regression, a special case of a linear model that uses the feature transform x ≠ (1, x, x^2 , · · ·). Consider the two regression problems below:

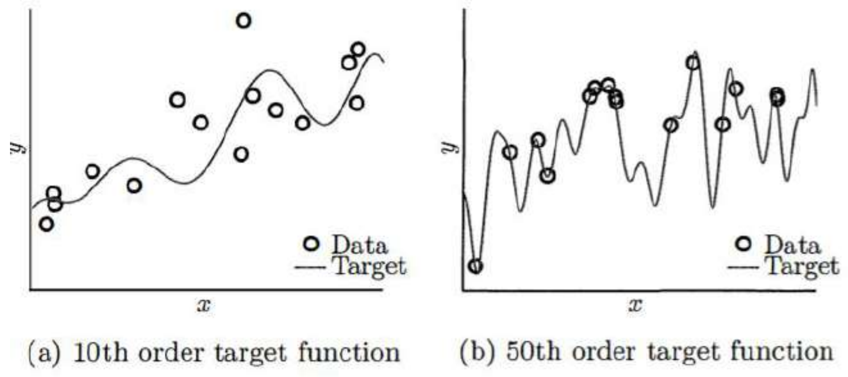


Fig 1.7 Nth order target function

In both problems, the target function is a polynomial and the data set D contains 15 data points. In (a), the target function is a 10th order polynomial and the sampled data are noisy (the data do not lie on the target function curve). In (b) , the target function is a 50th order polynomial and the data are noiseless.

The best 2nd and 10th order fits are shown in Figure 4.1, and the in-sample and out-of-sample errors are given in the following table.

10th order noisy target

	2nd Order	10th Order
E_{in}	0.050	0.034
E_{out}	0.127	**9.00**

50th order noiseless target

	2nd Order	10th Order
E_{in}	0.029	10^{-5}
E_{out}	0.120	**7680**

What the learning algorithm sees is the data, not the target function. In both cases, the 10th order polynomial heavily overfits the data, and results in a nonsensical final hypothesis which does not resemble the target function. The 2nd order fits do not capture the full nature of the target function either, but they do at least capture its general trend, resulting in significantly lower out-of-sample error. The 10th order fits have lower in-sample error and higher out of sample error, so this is indeed a case of overfitting that results in pathologically bad generalization.

These two examples reveal some surprising phenomena. Let's consider first the 10th order target function, Figure 4.l(a). Here is the scenario. Two learners, 0 (for overfitted) and R (for restricted), know that the target function is a 10th order polynomial, and that they will receive 15 noisy data points. Learner 0 uses model H10, which is known to contain the target function, and finds the best fitting hypothesis to the data. Learner R uses model H2, and similarly finds the best fitting hypothesis to the data.

The surprising thing is that learner R wins (lower out-of-sample error) by using the smaller model, even though she has knowingly given up the ability to implement the true target function. Learner R trades off a worse in-sample error for a huge gain in the generalization error, ultimately resulting in lower out-of-sample error.

What is funny here? A folklore belief about learning is that best results are obtained by incorporating as much information about the target function as is available. But as we see here, even if we know the order of the target and naively incorporate this knowledge by choosing the model accordingly (H10), the performance is inferior to that demonstrated by the more 'stable' 2nd order model.

The models H2 and H 10 were in fact the ones used to generate the learning curves in Chapter 2, and we use those same learning curves to illustrate overfitting in Figure 4.2. If you mentally superimpose the two plots, you can see that there is a range of N for which H10 has lower Ein but higher Eout than H2 does, a case in point of overfitting.

Is learner R always going to prevail? Certainly not. For example, if the data was noiseless, then indeed learner 0 would recover the target function exactly from 15 data points, while learner R would have no hope. This brings us to the second example, Figure 4.l(b). Here, the data is noiseless, but the target function is very complex (50th order polynomial). Again learner R wins, and again because learner 0 heavily overfits the data. Overfitting is not a disease inflicted only upon complex models with many more degrees of freedom than warranted by the complexity of the target function. In fact the reverse is true here, and overfitting is just as bad. What matters is how the model complexity matches the quantity and quality of the data we have, not how it matches the target function.

Catalysts for Overfitting

A skeptical reader should ask whether the examples in Figure 4.1 are just pathological constructions created by the authors, or is overfitting a real phenomenon which has to be considered carefully when learning from data? The next exercise guides you through an experimental design for studying overfitting within our current setup. We will use the results from this experiment to serve two purposes: to convince you that overfitting is not the result of some rare pathological construction, and to unravel some of the conditions conducive to overfitting.

Figure 4.3 shows how the extent of overfitting depends on certain parameters of the learning problem. In the figure, the colors map to the level of overfitting, with redder regions showing worse overfitting. These red regions are large overfitting is real, and here to stay.

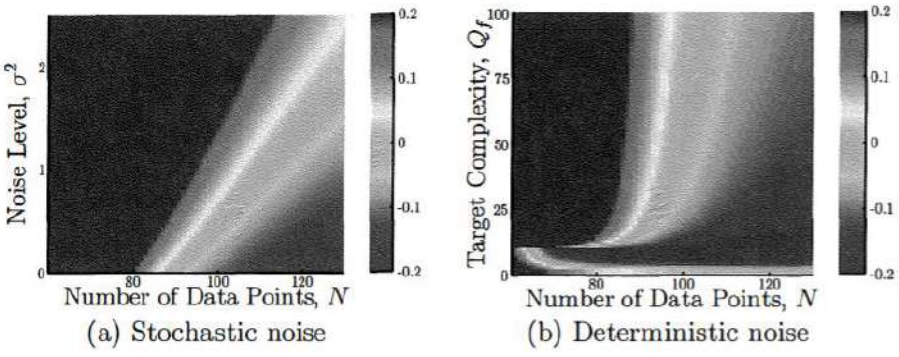


Fig 1.8 dependence of overfitting

Figure 4.3(a) reveals that there is less overfitting when the noise level σ^2drops or when the number of data points N increases (the linear pattern in Figure 4.3(a) is typical) . Since the 'signal' f is normalized to E[f 2] = 1, the noise level σ^2 is automatically calibrated to the signal level. Noise leads the learning astray, and the larger, more complex model is more susceptible to noise than the simpler one because it has more ways to go astray. Figure 4.3(b) reveals that target function complexity Q f affects overfitting in a similar way to noise, albeit nonlinearly. Deterministic noise. Why does a higher target complexity lead to more overfitting when comparing the same two models? The intuition is that for a given learning model, there is a best approximation to the target function. The part of the target function 'outside' this best fit acts like noise in the data. We can call this deterministic noise to differentiate it from the random stochastic noise. Just as stochastic noise cannot be modelled, the deterministic noise is that part of the target function which cannot be modelled. The learning algorithm should not attempt to fit the noise; however, it cannot distinguish noise from signal. On a finite data set, the algorithm inadvertently uses some of the degrees of freedom to fit the noise, which can result in overfitting and a spurious final hypothesis.

Figure illustrates deterministic noise for a quadratic model fitting a more complex target function. While stochastic and deterministic noise have similar effects on overfitting, there are two basic differences between the two types of noise. First, if we generated the same data (x values) again, the deterministic noise would not change but the stochastic noise would. Second, different models capture different 'parts' of the target function, hence the same data set will have different deterministic noise depending on which model we use. In reality, we work with one model at a time and have only one data set on hand. Hence, we have one realization of the noise to work with and the algorithm cannot differentiate between the two types of noise.

1.8.6 Validation

So far, we have identified overfitting as a problem, noise (stochastic and deterministic) as a cause, and regularization as a cure. In this section, we introduce another cure, called validation. One can think of both regularization and validation as attempts at minimizing Eout rather than just Ein. Of course the true Eout is not available to us, so we need an estimate of Eout based on information available to us in sample. In some sense, this is the Holy Grail of machine learning: to find an in-sample estimate of the out-of-sample error. Regularization attempts to minimize Eout by working through the equation

$E_{out}(h) \quad E_{in}(h)$+ over penality,

and concocting a heuristic term that emulates the penalty term. Validation, on the other hand, cuts to the chase and estimates the out-of-sample error directly.

$E_{out}(h) \quad E_{in}(h)$+ over penality.

The Validation Set

The idea of a validation set is almost identical to that of a test set. V\Te remove a subset from the data; this subset is not used in training. We then use this held-out subset to estimate the out-of-sample error. The held-out set is effectively out-of-sample, because it has not been used during the learning.

However, there is a difference between a validation set and a test set. Although the validation set will not be directly used for training, it will be used in making certain choices in the learning process. The minute a set affects the learning process in any way, it is no longer a test set. However, as we will see, the way the validation set is used in the learning process is so benign that its estimate of Eout remains almost intact.

Let us first look at how the validation set is created. The first step is to partition the data set D into a training set Dtrain of size (N - K) and a validation set Dval of size K. Any partitioning method which does not depend on the values of the data points will do; for exan1ple, we can select N - K points at random for training and the remaining for validation.

Now, we run the learning algorithm using the training set Dtrain to obtain a final hypothesis g ∈ H, where the 'minus' superscript indicates that some data points were taken out of the training. We then compute the validation error for g using the validation set Dval :

$$E_{val}(g^{-}) = \frac{1}{k}\sum_{x_n \in D_{val}} e\,(g^{-}(x_n), y_n)\ ,$$

Cross Validation

Validation relies on the following chain of reasoning,

$$E_{out}(g) \approx E_{out}(g^{-}) \approx E_{val}(g^{-})\ ,$$

which highlights the dilemma we face in trying to select K. We are going to output g. When K is large, there is a discrepancy between the two out-of-sample errors Eout(g) (which Eval directly estimates) and Eout (g) (which is the final error when we learn using all the data 'D). We would like to choose K as small as possible in order to minimize the discrepancy between Eout (g) and Eout(g) ; ideally K 1. However, if we make this choice, we lose the reliability of the validation estimate as the bound on the RHS of (4.9) becomes huge. The validation error Eval (g) will still be an unbiased estimate of Eout (g) (g is trained on N - 1 points) , but it will be so unreliable as to be useless since it is based on only one data point. This brings us to the cross validation estimate of out-of-sample error. We will focus on the leave-one-out version which corresponds to a validation set of size K 1, and is also the easiest case to illustrate. More popular versions typically use larger K, but the essence of the method is the same.

There are N ways to partition the data into a training set of size N - 1 and a validation set of size 1. Specifically, let

$$D_n = (x_1, y_1),\ldots,(x_{n-1}, y_{n-1}), (x_{n+1}, y_{n+1}), \ldots, (x_N, y_N)$$

be the data set D after leaving out data point (xn , Yn), which has been shaded in red. Denote the final hypothesis learned from Vn by $g^{-}n$. Let en be the error made by $g^{-}n$ on its validation set which is just a single data point { (xn , Yn)} :

$e_n = E_{val}(g^{-}{}_{n}) = \mathrm{e}(g^{-}{}_{n}(x_n), y_n)$

The cross validation estimate is the average value of the en 's,

$$E_{cv} = \frac{1}{N}\sum_{n=1}^{N} e_n$$

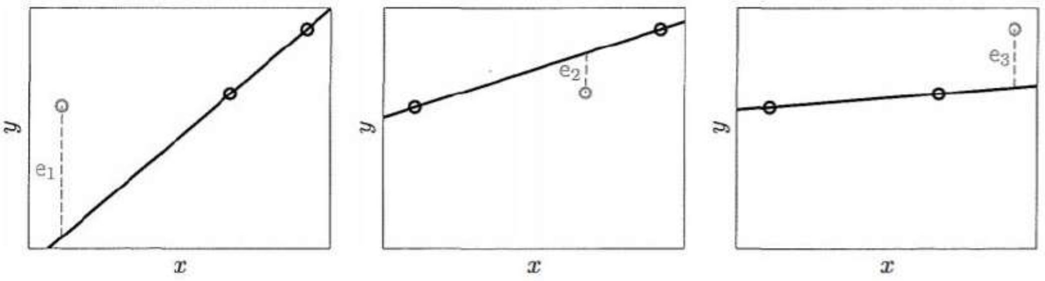


Figure 1.9 cross Validation

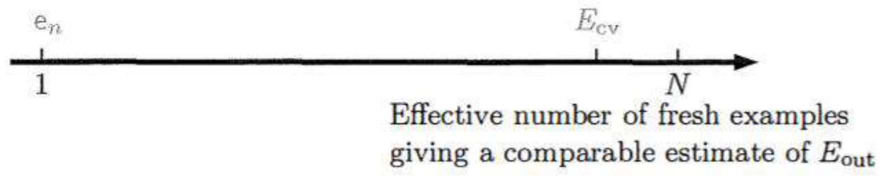
Figure 4.13 illustrates cross validation on a simple example. Each en is a wild, yet unbiased estimate for the corresponding Eout ($g^{-}n$). With cross validation, we have N functions g1, . . . , g]v together with the N error estimates e1, ... , eN. The hope is that these N errors together would be almost equivalent to estimating Eout on a reliable validation set of size N, while at the same time we managed to use N - 1 points to obtain each $g^{-}n$. Let's try to understand why Ecv is a good estimator of Eout·

First and foremost, Ecv is an unbiased estimator of 'Eout (g) ' . We have to be a little careful here because we don't have a single hypothesis g , as we did when using a single validation set. Depending on the (xn , Yn) that was taken out, each g;, can be a different hypothesis. To understand the sense in which Ecv estimates Eout , we need to revisit the concept of the learning curve.

Ideally, we would like to know Eout (g) . The final hypothesis g is the result of learning on a random data set 'D of size N. It is almost as useful to know the expected performance of your model when you learn on a data set of size N; the hypothesis g is just one such instance of learning on a data set of size N. This expected performance averaged over data sets of size N, when viewed as a function of N, is exactly the learning curve shown in Figure 4.2.

Now that we have our cross validation estimate of Eout, there is no need to output any of the g;, as our final hypothesis. We might as well squeeze every last drop of performance and retrain using the entire data set 'D, outputting g as the final hypothesis and getting the benefit of going from N - 1 to N on the learning curve. In this case, the cross validation estimate will on average be an upper estimate for the out-of-sample error: Eout (g) <= Ecv, so expect to be pleasantly surprised, albeit slightly.

If the N cross validation errors e1 , ... , eN were equivalent to N errors on a totally separate validation set of size N, then Ecv would indeed be a reliable estimate, for decent-sized N. The equivalence would hold if the individual en 's were independent of each other. Of course, this is too optimistic. Consider two validation errors en , em. The validation error en depends on g;, which was trained on data containing (xm, Ym) · Thus, en has a dependency on (xm, Ym) · The validation error em is computed using (xm, Ym) directly, and so it also has a dependency on (Xm, Ym). Consequently, there is a possible correlation between en and em through the data point (Xm, Ym). That correlation wouldn't be there if we were validating a single hypothesis using N fresh (independent) data points.


e_n
E_{cv}
1
N
Effective number of fresh examples giving a comparable estimate of E_{out}

Cross validation for model selection. In Figure 4.11, the estimates Em for the out-of-sample error of model Hm were obtained using the validation set. Instead, we may use cross validation estimates to obtain Em: use cross validation to obtain estimates of the out-of-sample error for each model Hi , ... , HM, and select the model with the smallest cross validation error. Now, train this model selected by cross validation using all the data to output a final hypothesis, making the usual leap of faith that Eout (g) tracks Eout (g) well.

If we use the in-sample error after fitting all the data (three points) , then the linear model wins because it can use its additional degree of freedom to fit the data better. The same is true with the cross-validation data sets of size two - the linear model has perfect in-sample error. But, with cross validation, what matters is the error on the outstanding point in each of these fits. Even to the naked eye, the average of the cross-validation errors is smaller for the constant model which obtained Ecv 0.065 versus Ecv 0.184 for the linear model. The constant model wins, according to cross validation. The constant model also has lower Eout and so cross validation selected the correct model in this example.

So far, we have lived in a world of unlimited computation, and all that mattered was out-of-sample error; in reality, computation time can be of consequence, especially with huge data sets. For this reason, leave-one-out cross validation may not be the method of choice.4 A popular derivative of leave one-out cross validation is V-fold cross validation. 5 In V-fold cross validation, the data are partitioned into D disjoint sets (or folds) D1 , ... , Dv, each of size approximately N /V; each set Dv in this partition serves as a validation set to compute a validation error for a hypothesis g learned on a training set which is the complement of the validation set, D \ Dv. So, you always validate a hypothesis on data that was not used for training that particular hypothesis. The V-fold cross validation error is the average of the D validation errors that are obtained, one from each validation set Dv. Leave-one-out cross validation is the same as N-fold cross validation. The gain from choosing D « N is computational. The drawback is that you will be estimating Eout for a hypothesis g trained on less data (as compared with leave-one-out) and so the discrepancy between Eout (g) and Eout (g) will be larger

CHAPTER-2
LITERATURE SURVEY

2. LITERATURE SURVEY

Various IEEE papers and textbooks have been referred to get an overview on Extraction of Adverse Drug Reactions. These IEEE papers and textbooks explained how to extract large set of data, train and test. We have studied about various existing mechanisms to extract ADRs.

2.1 DRUG-DRUG INTERACTION VIA CONVOLUTIONAL NEURAL NETWORKS

Authors: Shengyu Liu, Buzhou Tang, Qingcai Chen, and Xiaolong Wang

Introduction

Drug-drug interactions (DDIs) occur when two or more drugs are taken in combination that alters the way one or more drugs act in human body and may result in unexpected side effects. The unexpected side effects caused by DDIs are always very dangerous (may lead to deaths) and greatly increase healthcare costs. The more DDIs healthcare professionals know, the less medical accidents occur. Therefore, DDIs have always been attracting much attention in drug safety and healthcare management. There are several publicly available databases supporting healthcare professionals to find DDIs. For example, DrugBank, which is an online drug database, consists of 8311 drugs entries. Each drug entry contains more than 200 fields, including a DDI field.

However, the databases have a few limitations. Firstly, most DDI databases are dictionaries with a DDI field described in text such as DrugBank. The DDIs in these databases cannot be directly accessed like relational databases by healthcare professionals. Secondly, new DDIs are often detected by healthcare professionals and presented in literature, including scientific articles, books, and technical reports. It is impossible for healthcare professionals to find DDIs from the overwhelming amount of literature manually and to keep up-to-date with the latest DDI findings.

Therefore, DDI extraction, which detects DDIs in unstructured text and classifies them into predefined categories automatically, has become an increasing interest in medical text mining.

DDI extraction is a typical relation extraction task in natural language processing (NLP).Many methods have been proposed for DDI extraction and can be divided into two categories: rule-based and machine learning-based methods . Rule-based methods use manually defined rules to extract DDIs, whereas machine learning-based methods treat DDI extraction as a standard supervised learning problem over annotated corpora. Compared with rule-based methods, machine learning-based methods usually show better performance and better portability. Due to lack of annotated corpora, early DDI extraction methods are almost all rulebased. For example, Segura-Bedmar et al. defined a set of domain-specific rules to extract DDIs in DrugBank.With the organization of DDI Extraction challenges in 2011 and 2013, machine learning-based methods have been proposed for DDI extraction on the public corpora of the challenges. Both DDI Extraction 2011 and DDI Extraction 2013 are designed to extract drug-drug interactions from biomedical texts.

DDIs without type information are labeled in the DDI Extraction 2011 corpus, while, in the DDI Extraction 2013 challenge, DDIs are divided into four types, that is, “*mechanism,*” “*effect,*” “*advice,*” and “*int.*” The top performing systems on these corpora are based on support vector machines (SVM) with a large number of manually defined features .For example, the best system of the 2013 DDI Extraction challenge is based on SVM with a hybrid kernel using trigger words, dependency tree and parse tree features, and so forth. The subsequential best system is based on linear SVM with rich features, including word, word pair, dependency graph, parse tree, and noun phrase-constrained coordination features. These systems have to suffer from fussy feature engineering. Most of features used in these systems are usually generated by existing NLP toolkits which are imperfect. Errors caused by the NLP toolkits inevitably propagate in the DDI extraction systems.

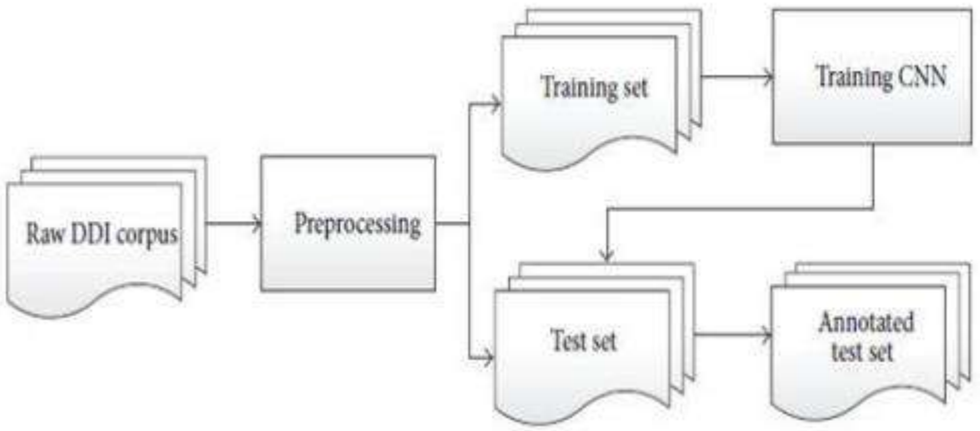


Fig. 2.1 Overall work flow of the CNN-based method for DDI extraction

Convolutional neural networks (CNN), a robust machine learning method proposed recently which almost does not need manually defined features, has exhibited great potential for many NLP tasks such as sentiment analysis, semantic parsing , and search query retrieval . However, it has never been used for DDI extraction. In this paper, we deploy CNN for this task. Inputs of the CNN-based method are sentences in which drugs are annotated. The CNN based method consists of four layers: a look-up table layer, a convolutional layer, a max pooling layer, and a softmax layer. Given a sentence with two drugs of interest, in the look-up table layer, each word is represented by word embeddings and position embeddings , and then the sentence is represented by a matrix that concatenates word embeddings and position embeddings of its words in the order of their occurrence.

In the convolutional layer, the matrix of the sentence is convolved with filters of different sizes, generating a group of feature vectors. The number of feature vectors is equal to that of filters and the size of each vector is determined by the context window considered. In the max pooling layer, the group of vectors is converted into a new vector by reducing each vector in the group into a feature. Finally, the vector obtained in the max pooling layer is fed to the fully connected soft max layer for classification. The word embeddings used in the look-up table layer are initialized by the “Order” algorithm on the MEDLINE abstracts in 2013 , whereas the position embeddings are randomly initialized.

Evaluation on the 2013 DDI Extraction challenge corpus demonstrates that the CNN-based DDI extraction system achieves a precision, a recall, and an F-score of 75.72%,64.66%, and 69.75%, respectively. It outperforms the best performing system by 2.75% in F-score, indicating that CNN is a good choice for DDI extraction.

Methods

DDI extraction is recognized as a multiclass classification problem for all possible interacting pairs of drugs in the same sentence. Each pair of drugs is classified into one of the predefined types of DDIs or classified as a non-interacting pair. Given a sentence with drugs, a total of Cn,2 = $n(n-1)/2$ DDI candidates need to be classified. Figure 1 illustrates the overall workflow of our CNN-based method for DDI extraction. The pre-processing module first blinds drugs, tokenizes sentences, normalizes tokens, and filters out non-interacting pairs from DDI candidates. Then the CNN module is used for DDI extraction. In the training phase, DDI candidates that annotated in the training set are positive samples with different types, and the other candidates are negative samples. The task of training is to obtain a CNN model on these samples. In the test phase, all DDI candidates are classified into different types of DDIs or non-DDI.

Preprocessing

To ensure generalization of machine learning-based methods, we follow previous studies to blind drugs in a sentence in the following way: the two drugs of interest are replaced by “drug1” and “drug2” in the order of their occurrence, respectively, and all the other drugs are replaced by “drug0.” For example, given a sentence with four drugs, “When *ALFENTA* is administered in combination with other *CNS depressants* such as *barbiturates,* or *tranquilizers,*” where drugs are highlighted in italic, C 4,2 = 6 DDI candidates with context (called instances) are generated, as shown in Table 1.After drug blinding, we use the Natural Language Toolkit (NLTK) to tokenize sentences and convert all words to lowercase.

Among all DDI instances, there are a large number of negative instances (non interacting drug pairs with context), which usually affect the performances of machine learning based DDI extraction systems because of data imbalance problem . Therefore, filtering out negative instances as many as possible is very important for subsequent DDI extraction module. In this study, we define the following four criteria for negative instance filtering.

An instance, denoted by “drug1, drug2,” is a negative instance if

(1) the two drugs have the same name,

(2) one drug is an abbreviation or acronym of the other,

(3) the two drugs appear in the same coordinate structure that has more than two drugs as elements,

(4) one drug is a special case of the other.

Exact string matching is used to determine whether the first criterion is satisfied, and some simple rules are defined to determine whether any one of the other three criteria is satisfied such as “drug1 (drug2),” “drug1, drug2, and drug0,” and “drug1 such as drug2.” For example, the fourth and fifth instances in Table 1 are negative instances because of criterion4.

Convolutional Neural Networks for Drug-Drug Interaction Extraction: The CNN model proposed for DDI extraction in this study is a four-layer model (shown in Figure 2), which is a variant of the model for sentence classification in .Besides word embeddings, position embeddings are also integrated into the CNN model in to encode relative distances between words and the two drugs of interest.

Look-Up Table

The CNN model takes DDI instances as input and generates their representation in look-up table layer. As required by CNN, we set all instances to be of the same length by appending padding, denoted by “#,” to short instances. The maximal length of all instances is a proper choice of the same length, denoted by n. Given a DDI instance $S = w1w2w3 \cdots wn$ with two drugs of interest ("drug1" and "drug2") at positions $p1$ and $p2$, a word wi is represented by d

w-dimensional word embeddings $\mathbf{e}wi$ and$2d$

p-dimensional position embeddings $[\mathbf{e}di1\mathrm{T}, \mathbf{e}di2\mathrm{T}]\mathrm{T}$ looked up from corresponding dictionaries. $\mathbf{C} \in \mathrm{R}dw\times|V|$, $\mathbf{D}1\in\mathrm{R}dp\times(2n-1)$, and $\mathbf{D}2\in\mathrm{R}dp\times(2n-1)$, where V is the vocabulary and $di1 = i - p1$ and $di2 = i - p2$ (ranging from $-n + 1$ to $n - 1$) are, respectively, the relative distance between

the word and the first drug and that between the word and the second drug.

For the two types of embeddings, word embeddings can be initialized by employing unsupervised word embeddings algorithm on large-scale un annotated texts, whereas position embeddings only can be randomly initialized.

Convolution

The matrix of a DDI instance (i.e., **x**) is fed to the convolutional layer to generate features by convolving **x** with filters of different sizes. As there are various types of filters of different sizes, we can obtain a group of feature vectors. The number of feature vectors is equal to the number of filters.

Max Pooling

The max pooling layer extracts the most important feature from each feature vector to reduce the computational complexity of subsequent layers. Concretely, the feature of maximum value $\hat{f} = \max\{f1, f2, f3, \ldots, fn-k+1\}$ is extracted to represent a feature vector $\mathbf{f} = [f1, f2, f3, \ldots, fn-k+1]$. Correspondingly, if there are l feature filters, the matrix of a DDI instance (i.e., **x**) is converted into a new vector of length l, denoted by $\mathbf{z} = [\hat{f}1, \hat{f}2, \hat{f}3, \ldots, \hat{f}l]$,

where $\hat{f}i$ is the feature extracted from the ith feature vector.

Softmax Regression

To prevent neural networks from overfitting, we follow [19] to randomly drop out units (along with their connections) from the networks during training. The feature vector **z** obtained by max pooling is not directly fed to the fully connected softmax layer for classification. Firstly, we randomly set each element of **z** to zero with a probability p (following the Bernoulli distribution) and

obtain a new feature vector $\mathbf{z}d$.Then the vector $\mathbf{z}d$ is fed to the fully connected softmax layer. At test time, the feature vector **z** is directly fed to the softmax layer for classification without dropout.

Model Training

The following parameters of the CNN model need to be updated during training: the word embeddings matrix, the position embeddings matrixes, the filters, and the weight matrix of the softmax layer. We use stochastic gradient descent with shuffled mini batches and the AdaDelta update rule as to learn the parameters. At each gradient descent step, we rescale the weight vectors of the softmax layer when their $l2$-norms exceed a certain threshold.

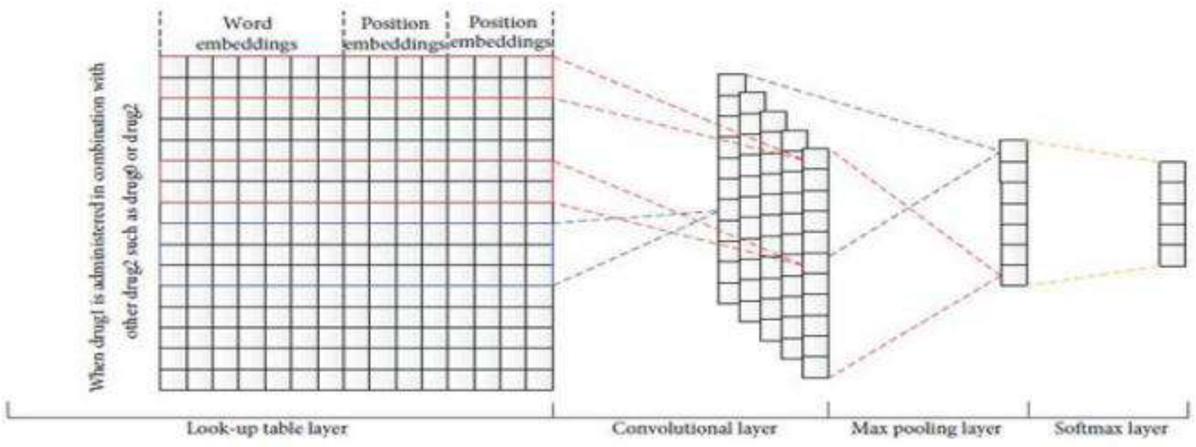


Fig. 2.2 Architecture of the CNN model for DDI Extraction

CHAPTER-3
SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 BIOMEDICAL TEXT MINING

Biomedical text mining (including **biomedical natural language processing** or **BioNLP**) refers to the methods and study of how text mining may be applied to texts and literature of the biomedical and molecular biology domains. As a field of research, biomedical text mining incorporates ideas from NLP, bioinformatics, medical informatics and computational linguistics. The strategies developed through studies in this field are frequently applied to the biomedical and molecular biology literature available through services such as PubMed.

Availability of annotated text data

Large annotated corpora used in the development and training of general purpose text mining methods (e.g., sets of movie dialogue, product reviews, or Wikipedia article text) are not specific for biomedical language. While they may provide evidence of general text properties such as parts of speech, they rarely contain concepts of interest to biologists or clinicians. Development of new methods to identify features specific to biomedical documents therefore requires assembly of specialized corpora. Resources designed to aid in building new biomedical text mining methods have been developed through the Informatics for Integrating Biology and the Bedside (i2b2) challenges and biomedical informatics researchers. Text mining researchers frequently combine these corpora with the controlled vocabularies and ontologies available through the National Library of Medicine's Unified Medical Language System (UMLS) and Medical Subject Headings (MeSH).

Machine learning-based methods often require very large data sets as training data to build useful models. Manual annotation of large text corpora is not realistically possible. Training data may therefore be products of weak supervision or purely statistical methods.

Data structure variation

Like other text documents, biomedical documents contain unstructured data. Research publications follow different formats, contain different types of information, and are interspersed with figures, tables, and other non-text content. Both unstructured text and semi-structured document elements, such as tables, may contain important information that should be text mined . Clinical documents may vary in structure and language between departments and locations. Other types of biomedical text, such as drug labels, may follow general structural guidelines but lack further details.

Relationship discovery

Biomedical documents describe connections between concepts, whether they are interactions between biomolecules, events occurring subsequently over time (i.e., temporal relationships), or causal relationships. Text mining methods may perform relation discovery to identify these connections, often in concert with named entity recognition. **Hedge cue detection**

The challenge of identifying uncertain or "hedged" statements has been addressed through hedge cue detection in biomedical literature.

Claim detection

Multiple researchers have developed methods to identify specific scientific claims from literature. In practice, this process involves both isolating phrases and sentences denoting the core arguments made by the authors of a document (a process known as argument mining, employing tools used in fields such as political science) and comparing claims to find potential contradictions between them.

Information retrieval and question answering

Biomedical text mining supports applications for identifying documents and concepts matching search queries. Search engines such as PubMed search allow users to query literature databases with words or phrases present in document contents, metadata, or indices such as MeSH. Similar approaches may be used for medical literature retrieval. For more fine-grained results, some applications permit users to search with natural language queries and identify specific biomedical relationships. The National Library of Medicine with partner the Semantics Scholar project[36] of the Allen Institute for AI, on March 16, 2020, launched the COVID-19 Open Research Dataset (CORD-19) to enable text mining of the current literature

Applications

Text mining applications in the biomedical field include computational approaches to assist with studies in protein docking, protein interactions, and protein-disease associations.

Gene cluster identification

Methods for determining the association of gene clusters obtained by microarray experiments with the biological context provided by the corresponding literature have been developed.[90]

Protein interactions

Automatic extraction of protein interactions and associations of proteins to functional concepts (e.g. gene ontology terms) has been explored.[citation needed] The search engine PIE was developed to identify and return protein-protein interaction mentions from MEDLINE-indexed articles. The extraction of kinetic parameters from text or the subcellular location of proteins have also been addressed by information extraction and text mining technology.

Gene-disease associations

Text mining can aid in gene prioritization, or identification of genes most likely to contribute to genetic disease. One group compared several vocabularies, representations and ranking algorithms to develop gene prioritization benchmarks on the novel virus.

3.2 PHARMACOVIGILANCE FROM SOCIAL MEDIA

Social media has been increasingly used for medical and pharmacological research since the early 2010s; the term "pharmacovigilance" was coined for automated monitoring of social media for potentially adverse drug effects and interactions.

NLP techniques have been applied in five main domain of texts: (i) biomedical literature, clinical trial records, and electronic medical/health records (e.g., medical correspondence and letters); (ii) short messages from Twitter; (iii) user reviews from health-related and e-commerce websites; (iv) web search logs; and (v) forum discussions and message boards about medications, health conditions, treatment modality, and so on. Most of these works focused on creating linguistic methods based on keywords for extracting major adverse effects, classifiers to detect whether a text contains ADRs or is relevant to drug reactions, and sequence labelling algorithms to extract mentions of ADRs. A review of techniques applied to drug reaction detection has been given in.

In opinion mining, one of the major tasks is the identification of opinion targets (also called aspects) or opinion expressions. This task has been studied by many researchers using frequency-based methods and unsupervised and supervised methods. In, authors described linguistic resources for event extraction: linguistics databases and vocabularies such as thesauri. Currently, most of the state-of-the-art methods are based on CRF with a set of handcrafted features and bidirectional RNNs. Irsoy and Cardie applied deep RNNs to extract direct or expressive subjective expressions; in their experiments, 3-layer RNN outperformed CRF, semi-CRF, and 1-layer (i.e., shallow) RNN. Liu et al. applied RNNs for aspect extraction from data sets about laptops and restaurants, and RNNs based on pretrained word embeddings outperformed feature-rich CRF-based models.

In recent years, there has been a growing interest in the area of detecting ADRs from social media. It started in 2010 with a pioneering study of Leaman et al. who analysed user posts regarding six drugs from the health-related social network Daily Strength. FDA alerts were used as a gold standard to evaluate discovered associations between drugs and ADRs. Yang et al. conducted an experiment for ten drugs and five ADRs to examine associations between them on texts from online healthcare communities using association mining techniques. Rastegar-Mojarad et al. developed a rule-based system to extract drug effects. Feldman et al. identified ADRs on texts from health-related online forums. They employed dictionary-based drug detection, and symptoms were extracted with a combination of dictionary-based and pattern-based methods. Point wise mutual information (PMI) was computed to evaluate the likelihood of a drug-ADR relation. The authors analyzed several case studies of drugs to show that some ADRs were reported prior to the FDA communication. One limitation of this work is the amount of annotated data; the test set contained less than 500 samples. See for a comprehensive review of ADR extraction from social media data with NLP-based approaches.

Supervised machine learning techniques have been successfully applied to detect ADRs. Bian et al. utilized an SVM classifier to identify tweets describing ADRs. YomTov and Gabrilovich analyzed web search query logs to extract information related to drugs and adverse reactions. ADR extraction has been regarded in many studies as a sequence labelling problem using conditional random fields (CRF). CRFs with a rich set of contextual, lexicon-based, grammatical, and semantic features were used in. In, the semantic features were based on word clusters using k-means clustering on pretrained word embeddings. A set of experiments showed that contextual and semantic features are the most effective to classify ADRs in tweets. We also note a Social Media Mining Shared Task Workshop (organized as part of the Pacific Symposium on Bio computing 2016) devoted to mining pharmacological and medical information from Twitter, with a competition based on a published dataset. Supervised models tend to work well when trained on fully labeled data. Although there is a large amount of unlabelled data from social media, labelled data are time consuming to obtain. Gupta et al. used semi supervised learning of patterns to identify drugs, symptoms, and conditions. Lexico-syntactic patterns have been learned with a seed dictionary of terms, and a bootstrapped rule-based method extracted specific entities that were missing from the seed dictionaries. One limitation of this approach is that it does not identify long descriptive phrases. Stanovsky et al. employed an active learning technique to create a bootstrap lexicon of ADRs. The main advantage of this approach is that it can identify entities with a small number of hand-written rules or hand-labelled examples. We mark these works as possibilities for future improvements of this area.

3.3 CONDITIONAL RANDOM FIELDS

The amount of text data being generated in the world is staggering. Google processes more than 40,000 searches EVERY second! According to a Forbes report, every single minute we send 16 million text messages and post 510,00 comments on Facebook. For a layman, it is difficult to even grasp the sheer magnitude of data out there?

News sites and other online media alone generate tons of text content on an hourly basis. Analyzing patterns in that data can become daunting if you don't have the right tools. Here we will discuss one such approach, using entity recognition, called Conditional Random Fields (CRF).

WHAT IS ENTITY RECOGNITION?

Entity recognition has seen a recent surge in adoption with the interest in Natural Language Processing (NLP). An entity can generally be defined as a part of text that is of interest to the data scientist or the business. Examples of frequently extracted entities are names of people, address, account numbers, locations etc. These are only simple examples and one could come up with one's own entity for the problem at hand.

To take a simple application of entity recognition, if there's any text with "London" in the dataset, the algorithm would automatically categorize or classify that as a location (you must be getting a general idea of where I'm going with this).

Let's take a simple case study to understand our topic in a better way.

CASE STUDY OBJECTIVE & UNDERSTANDING DIFFERENT APPROACHES

Suppose that you are part of an analytics team in an insurance company where each day, the claims team receives thousands of emails from customers regarding their claims. The claims operations team goes through each email and updates an online form with the details before acting on them.

You are asked to work with the IT team to automate the process of pre-populating the online form. For this task, the analytics team needs to build a custom entity recognition algorithm.

To identify entities in text, one must be able to identify the pattern. For example, if we need to identify the claim number, we can look at the words around it such as “my id is” or “my number is”, etc. Let us examine a few approaches mentioned below for identifying the patterns.

1**. Regular expressions:** Regular expressions (RegEx) are a form of finite state automaton. They are very helpful in identifying patterns that follow a certain structure. For example, email ID, phone number, etc. can be identified well using RegEx. However, the downside of this approach is that one needs to be aware of all the possible exact words that occur before the claim number. This is not a learning approach, but rather a brute force one

2. Hidden Markov Model (HMM): This is a sequence modeling algorithm that identifies and learns the pattern. Although HMM considers the future observations around the entities for learning a pattern, it assumes that the features are independent of each other. This approach is better than regular expressions as we do not need to model the exact set of word(s). But in terms of performance, it is not known to be the best method for entity recognition

3. MaxEnt Markov Model (MEMM): This is also a sequence modeling algorithm. This does not assume that features are independent of each other and also does not consider future observations for learning the pattern. In terms of performance, it is not known to be the best method for identifying entity relationships either

4. Conditional Random Fields (CRF): This is also a sequence modeling algorithm. This not only assumes that features are dependent on each other, but also considers the future observations while learning a pattern. This combines the best of both HMM and MEMM. In terms of performance, it is considered to be the best method for entity recognition problem

FORMULATING CONDITIONAL RANDOM FIELDS (CRF)

The bag of words (BoW) approach works well for multiple text classification problems. This approach assumes that presence or absence of word(s) matter more than the sequence of the words. However, there are problems such as entity recognition, part of speech identification where word sequences matter as much, if not more. Conditional Random Fields (CRF) comes to the rescue here as it uses word sequences as opposed to just words.

Let us now understand how CRF is formulated.

Below is the formula for CRF where Y is the hidden state (for example, part of speech) and X is the observed variable (in our example this is the entity or other words around it).

$$p(\mathbf{y}|\mathbf{x}) = \underbrace{\frac{1}{Z(\mathbf{x})}}_{\text{Normalization}} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \underbrace{\theta_k}_{\text{Weight}} \underbrace{f_k(y_t, y_{t-1}, \mathbf{x}_t)}_{\text{Feature}}\right\}$$

Broadly speaking, there are 2 components to the CRF formula:

1. Normalization: You may have observed that there are no probabilities on the right side of the equation where we have the weights and features. However, the output is expected to be a probability and hence there is a need for normalization. The normalization constant Z(x) is a sum of all possible state sequences such that the total becomes 1. You can find more details in the reference section of this article to understand how we arrived at this value.

2. Weights and Features: This component can be thought of as the logistic regression formula with weights and the corresponding features. The weight estimation is performed by maximum likelihood estimation and the features are defined by us.

3.4 TYPES OF RECURRENT NEURAL NETWORKS

The different types of RNN are:

- One to One RNN
- One to Many RNN
- Many to One RNN
- Many to Many RNN

We will review the basic idea of RNN and then, move on to the different types of RNN and explore them in depth.

Recurrent Neural Network is a generalization of feed-forward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.

TYPES OF RNN

So we have established that Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. RNN models are mostly used in the fields of natural language processing and speech recognition. Let’s look at its types:

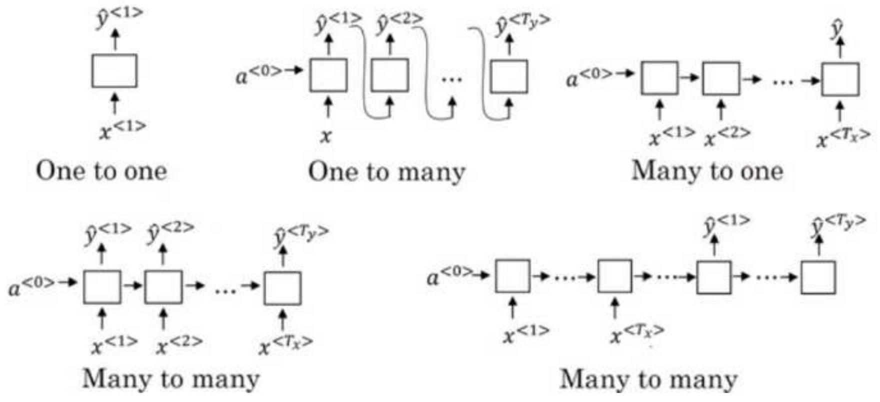


Fig. Types of RNN

One to One RNN

One to One RNN (Tx=Ty=1) is the most basic and traditional type of Neural network giving a single output for a single input, as can be seen in the above image.

One to Many

One to Many (Tx=1, Ty>1) is a kind of RNN architecture is applied in situations that give multiple output for a single input. A basic example of its application would be Music generation. In Music generation models, RNN models are used to generate a music piece (multiple outputs) from a single musical note (single input).

Many to One

Many-to-one RNN architecture (Tx>1, Ty=1) is usually seen for sentiment analysis model as a common example. As the name suggests, this kind of model is used when multiple inputs are required to give a single output.

Take for example The Twitter sentiment analysis model. In that model, a text input (words as multiple inputs) gives its fixed sentiment (single output). Another example could be movie ratings model that takes review texts as input to provide a rating to a movie that may range from 1 to 5.

Many-to-Many

As is pretty evident, Many-to-Many RNN (Tx>1, Ty>1) Architecture takes multiple input and gives multiple output, but Many-to-Many models can be two kinds as represented above:

1. **Tx=Ty:**

This refers to the case when input and output layers have the same size. This can be also understood as every input having a output, and a common application can be found in Named-entity Recognition.

2. **Tx!=Ty:**

Many-to-Many architecture can also be represented in models where input and output layers are of different size, and the most common application of this kind of RNN architecture is seen in Machine Translation. For example, "I Love you", the 3 magical words of the English language translates to only 2 in Spanish, "te amo". Thus, machine translation models are capable of returning words more or less than the input string because of a non-equal Many-to-Many RNN architecture works in the background.

3.5 CONVOLUTIONAL NEURAL NETWORK

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

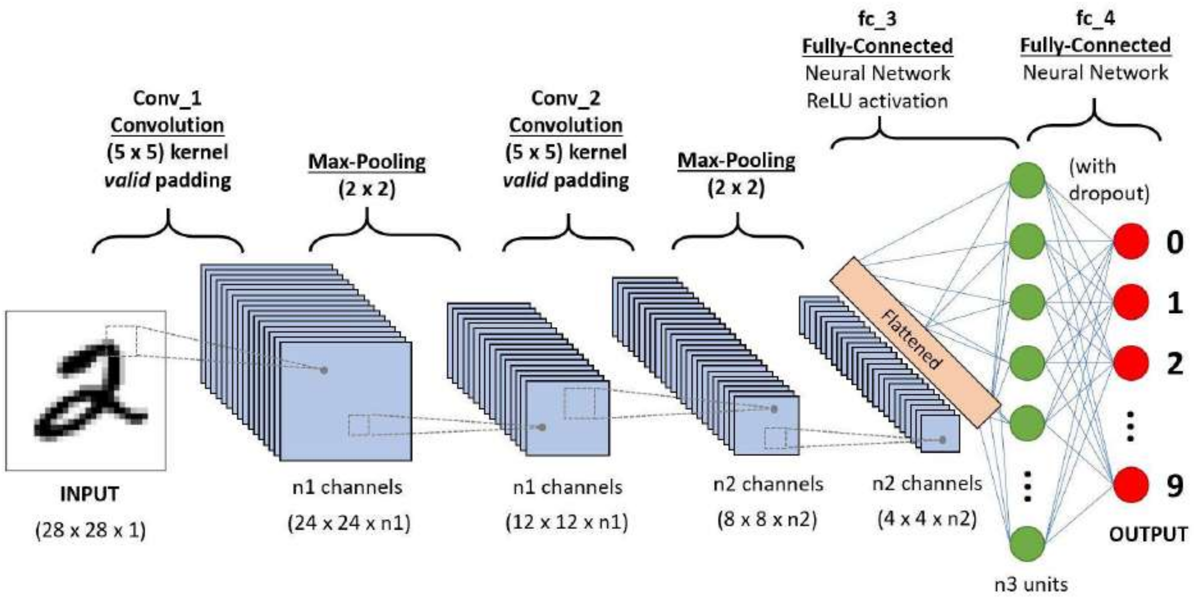


Fig 3.1 A CNN sequence to classify handwritten digits

Why ConvNets over Feed-Forward Neural Nets?

In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

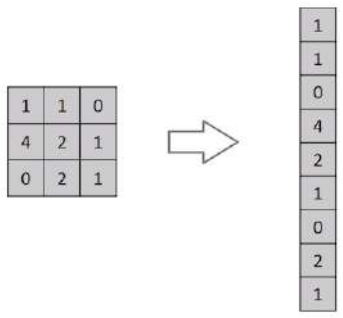


Fig 3.2 Flattening of a 3x3 image matrix into a 9x1 vector

Convolution Layer — the Kernel

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

In the above demonstration, the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix.

Kernel/Filter,

K = 1 0 1

0 1 0

1 0 1

The Kernel shifts 9 times because of Stride Length = 1 (Non-Stride), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.

When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — **same Padding**.

On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — **Valid Padding**.

Pooling Layer

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel. On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.

Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling**.

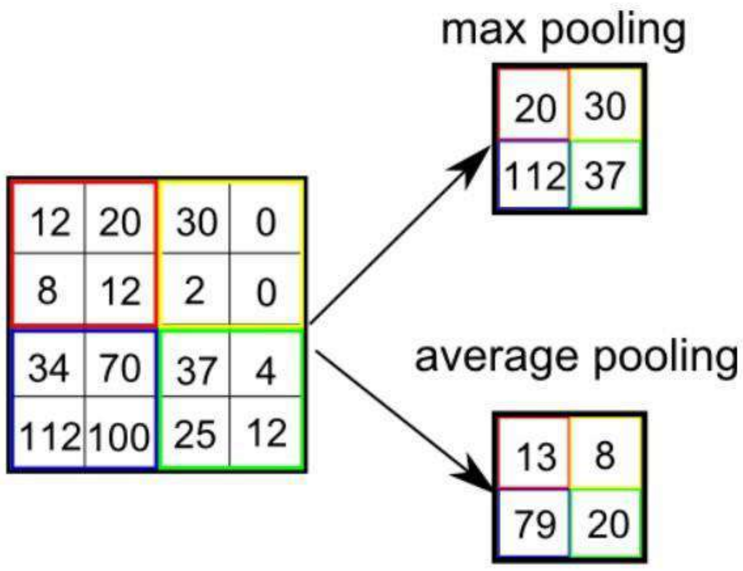


Fig 3.3 Types of Pooling

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Classification — Fully Connected Layer (FC Layer)

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

CHAPTER-4
SYSTEM DESIGN

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

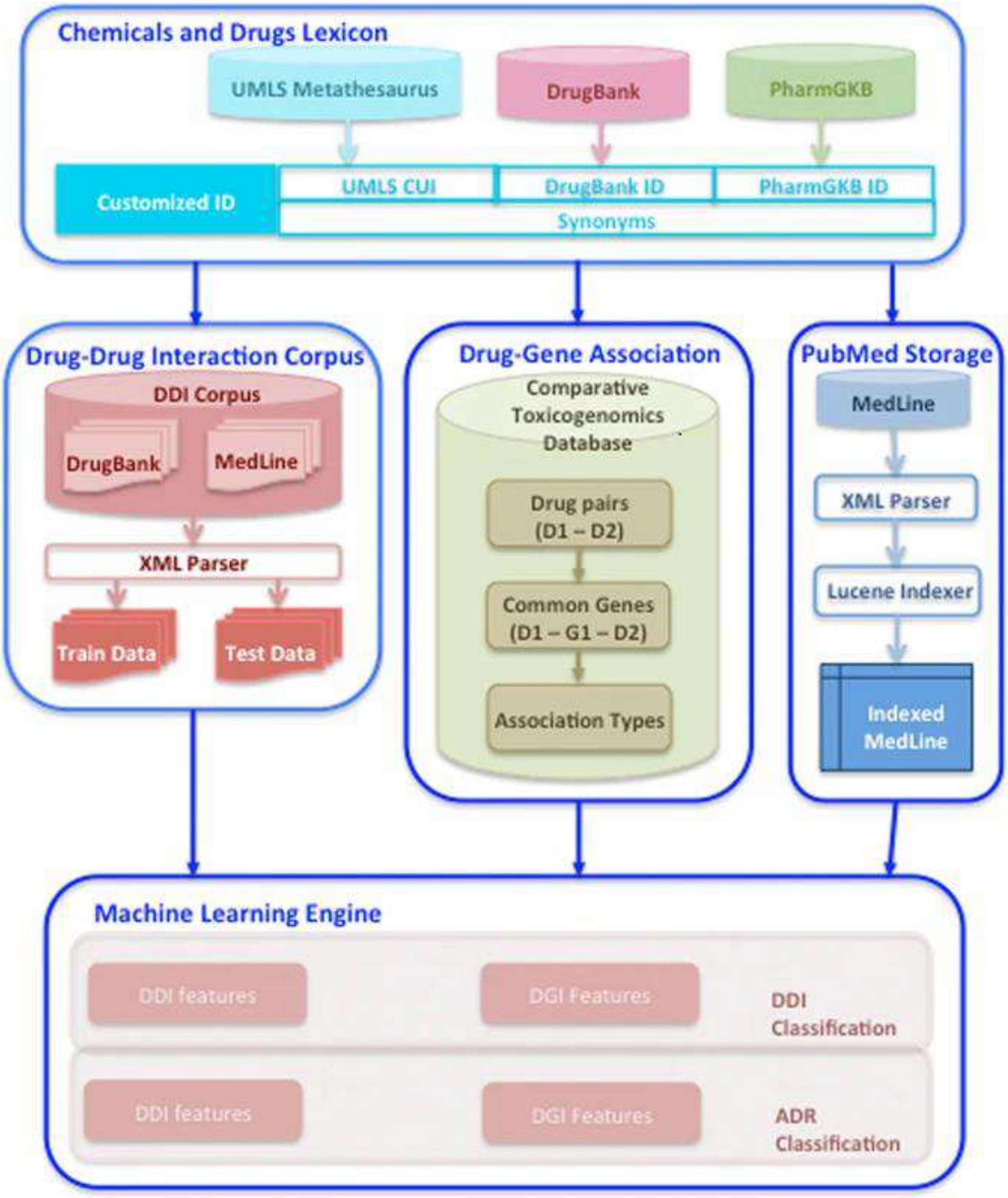


Fig 4.1 System Architecture

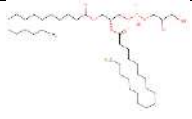
4.2 DATASETS

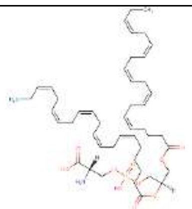
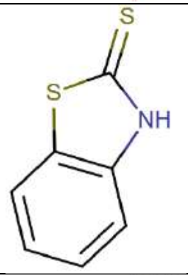
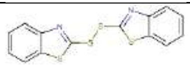
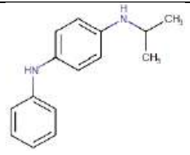
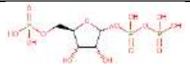
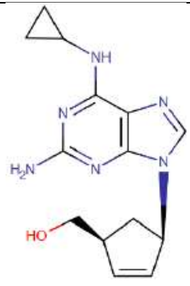
4.2.1 DrugBank

The DrugBank database is a comprehensive, freely accessible, online database containing information on drugs and drug targets.As both a bioinformatics and a cheminformatics resource, DrugBank combines detailed drug (i.e. chemical, pharmacological and pharmaceutical) data with comprehensive drug target (i.e. sequence, structure, and pathway) information. DrugBank uses a fair bit of content from Wikipedia. Wikipedia also often links to Drugbank.

The latest release of the database (version 5.0) contains 9591 drug entries including 2037 FDA-approved small molecule drugs, 241 FDA-approved biotech (protein/peptide) drugs, 96 nutraceuticals and over 6000 experimental drugs.Additionally, 4270 non-redundant protein (i.e.drugtarget/enzyme/transporter/carrier) sequences are linked to these drug entries. Each DrugCard entry contains more than 200 data fields with half of the information being devoted to drug/chemical data and the other half devoted to drug target or protein data.

Four additional databases, HMDB, T3DB, SMPDB and FooDB are also part of a general suite of metabolomic/cheminformatic databases. HMDB contains equivalent information on more than 40,000 human metabolites, T3DB contains information on 3100 common toxins and environmental pollutants, and SMPDB contains pathway diagrams for nearly 700 human metabolic pathways and disease pathways, while FooDB contains equivalent information on ~ 28,000 food components and food additives.

NAME	WEIGHT	STRUCTURE	DESCRIPTION	CATEGORIES
1-Palmitoyl-2-oleoyl-sn-glycero-3-(phospho-rac-(1-glycerol))	749.02 C40H77O10P		A synthetic lung surfactant used to treat infant respiratory distress syndrome.	Not Available
1,2-Distearoyllecithin	790.161 C44H88NO8P		Not Annotated	Alcohols / Carbohydrates / Glycerophosphates / Glycerophospholipids / Lipids / Membrane Lipids / Phosphatidic Acids / Phospholipids / Sugar Alcohols / Sugar Phosphates /

				Triose Sugar Alcohols
1,2-icosapentoyl-sn-glycero-3-phosphoserine	842.064 $C_{47}H_{72}NO_{10}P$		Not Annotated	Not Annotated
2-mercaptobenzothiazole	167.251 $C_7H_5NS_2$		Not Annotated	Standardized Chemical Allergen
2,2'-Dibenzothiazyl disulfide	332.47 $C_{14}H_8N_2S_4$		2,2'-Dibenzothiazyl disulfide is approved for use within allergenic epicutaneous patch tests which are indicated for use as an aid in the diagnosis of allergic contact dermatitis (ACD) in persons 6...	Latex Hypersensitivity / Standardized Chemical Allergen
4-(Isopropylamino)diphenylamine	226.323 $C_{15}H_{18}N_2$		4-(Isopropylamino)diphenylamine is approved for use within allergenic epicutaneous patch tests which are indicated for use as an aid in the diagnosis of allergic contact dermatitis (ACD) in persons 6 years...	Antioxidants / Standardized Chemical Allergen
5-O-phosphono-alpha-D-ribofuranosyl diphosphate	390.0696 $C_5H_{13}O_{14}P_3$		The key substance in the biosynthesis of histidine, tryptophan, and purine and pyrimidine nucleotides.	Pentosephosphates
Abacavir	286.3323 $C_{14}H_{18}N_6O$		An antiviral nucleoside reverse transcriptase inhibitor used in combination with other antiretrovirals for the treatment of HIV.	Anti-HIV Agents / Human Immunodeficiency Virus Nucleoside Analog Reverse Transcriptase Inhibitor / Nucleoside and Nucleotide Reverse Transcriptase

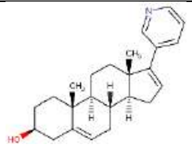
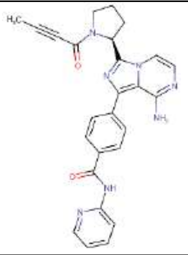
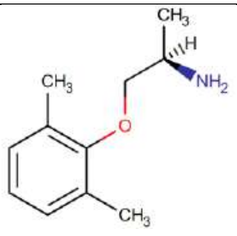
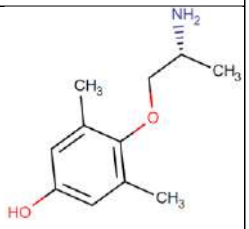
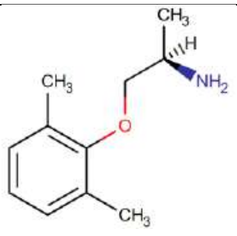
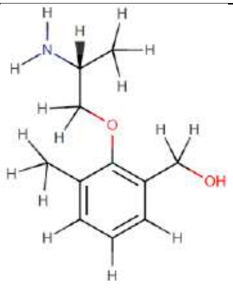
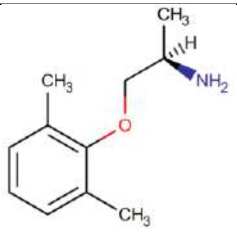
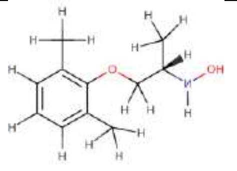
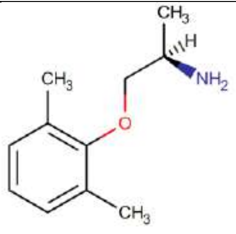
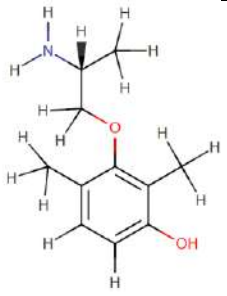
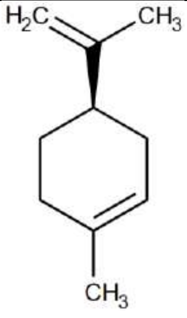
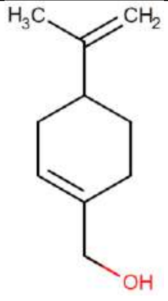
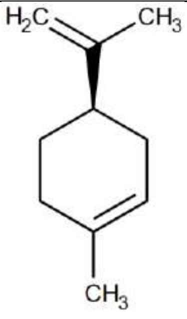
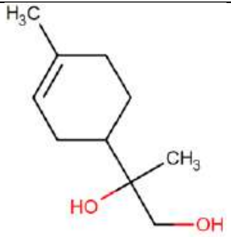
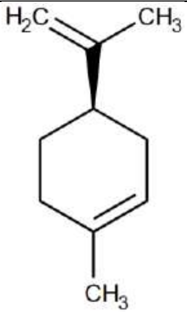
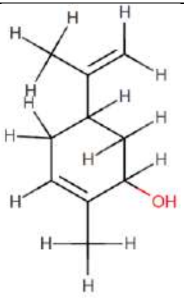
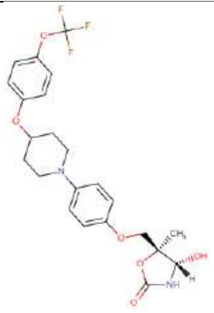
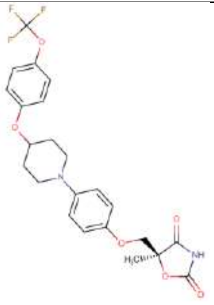
				Inhibitors / Reverse Transcriptase Inhibitors
Abiraterone	349.509 $C_{24}H_{31}NO$		An antiandrogen used in the treatment of prostate cancer.	Androstenes / Antineoplastic Agents / Cytochrome P-450 Enzyme Inhibitors / Cytochrome P450 17A1 Inhibitors / Steroid Synthesis Inhibitors
Acalabrutinib	465.517 $C_{26}H_{23}N_7O_2$		A Bruton tyrosine kinase inhibitor used to treat mantle cell lymphoma, chronic lymphocytic leukemia, and small lymphocytic lymphoma.	Kinase Inhibitor

Table 4.1 DrugBank drugs

SUBSTRATE		ENZYME	PRODUCTS	
(2R)-1-(2,6-dimethylphenoxy)propan-2-amine DB07129		Cytochrome P450 1A2 / Cytochrome P450 2D6	(R)-(-)-Mexiletine, p-hydroxyl	
(2R)-1-(2,6-dimethylphenoxy)propan-2-amine DB07129		Cytochrome P450 1A2 / Cytochrome P450 2D6	(R)-(-)-Mexiletine, 2-OH-hydroxymethyl	
(2R)-1-(2,6-dimethylphenoxy)propan-2-amine DB07129		Cytochrome P450 1A2 / Cytochrome P450 2B6 / Cytochrome P450 2E1	(R)-(-)-Mexiletine, N-hydroxy	

(2R)-1-(2,6-dimethylphenoxy)propan-2-amine DB07129		Cytochrome P450 1A2 / Cytochrome P450 2D6	(R)-(-)-Mexiletine, m-hydroxyl	
(4R)-limonene DB08921		Cytochrome P450 2C9 / Cytochrome P450 2C19 / Cytochrome P450 3A4	Perillyl alcohol	
(4R)-limonene DB08921		Not AvailableNot Available	Uroterpenol	
(4R)-limonene DB08921		Cytochrome P450 2C9 / Cytochrome P450 2C19	Carveol	
Carveol(4R,5S)-DM-6721 DBMET01969		Cytochrome P450 3A4	(S)-DM-6717	

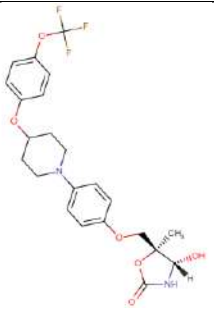
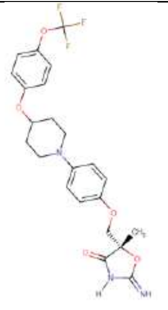
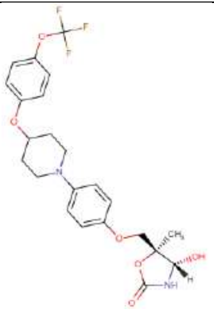
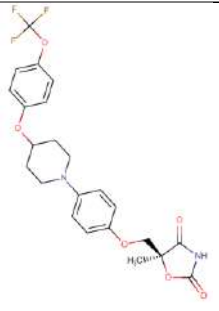
(4RS,5S)-DM-6720 DBMET01966		Cytochrome P450 3A4 / Cytochrome P450 1A1	(S)-DM-6718	
(4S,5S)-DM-6722 DBMET01970		Cytochrome P450 3A4	(S)-DM-6717	

Table 4.2 DrugBank drug Reactions

4.2.2 MEDLINE

MEDLINE (Medical Literature Analysis and Retrieval System Online, or MEDLARS Online) is a bibliographic database of life sciences and biomedical information. It includes bibliographic information for articles from academic journals covering medicine, nursing, pharmacy, dentistry, veterinary medicine, and health care. MEDLINE also covers much of the literature in biology and biochemistry, as well as fields such as molecular evolution.

Compiled by the United States National Library of Medicine (NLM), MEDLINE is freely available on the Internet and searchable via PubMed and NLM's National Center for Biotechnology Information's Entrez system.

CHAPTER-5
IMPLEMENTATION

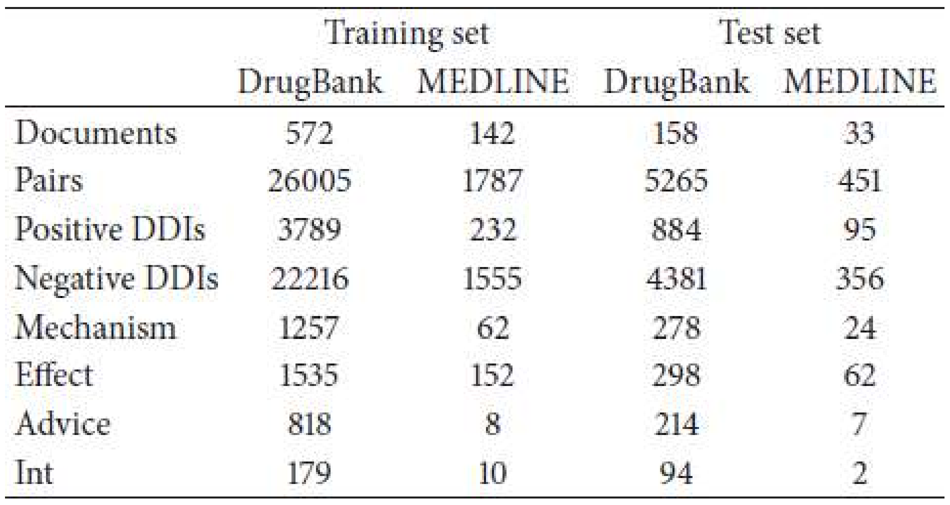
5. IMPLEMENTATION

5.1 PROCESS STEPS

Dataset-Testing and Training:

The CNN-based DDI extraction system is developed and evaluated on the DDI corpus of the 2013 DDI Extraction challenge, which is composed of 730 DrugBank documents and 175 MEDLINE abstracts about DDIs. The corpus is split into two parts: a training set (572 DrugBank documents and 142MEDLINE abstracts) for system development and a test set (158 DrugBank documents and 33 MEDLINE abstracts) for system evaluation. All drugs and pairs of drugs in each sentence are annotated. Among the pairs of drugs (totally 33508), 5000 interacting pairs (i.e., DDIs) are classified into the following four types: mechanism, effect, advice, and int. The definitions of the four types of DDIs are as follows.

- Mechanism. Mechanism is assigned when pharmacokinetic mechanism of a DDI is described.
- Effect. Effect is assigned when effect of a DDI is described.
- Advice. Advice is assigned when a recommendation or advice regarding a DDI is given.
- Int. Int is assigned when the sentence simply states that a DDI occurs and does not provide any information about the DDI.



	Training set		Test set	
	DrugBank	MEDLINE	DrugBank	MEDLINE
Documents	572	142	158	33
Pairs	26005	1787	5265	451
Positive DDIs	3789	232	884	95
Negative DDIs	22216	1555	4381	356
Mechanism	1257	62	278	24
Effect	1535	152	298	62
Advice	818	8	214	7
Int	179	10	94	2

Table 5.1 Statistics of the DDI corpus of the 2013 DDIExtraction challenge.

As four types of DDIs are defined in the corpus of the 2013 DDI Extraction challenge, DDI instances need to be classified into five categories: mechanism, effect, advice, int and non-interacting, corresponding to the output of the last layer of the CNN model. The word embeddings matrix used in our experiments is initialized by an unsupervised word embeddings learning algorithm “Order” [22] on 17.3-gigabyte unannotated article abstracts extracted from MEDLINE released in 2013 [24].We also adopt the NLTK to preprocess the abstracts, including splitting them into sentences, tokenizing the sentences, and converting all words to lowercase. Finally, we obtain 110 million sentences with 2.8 billion words from a vocabulary of size 1.99 million. Following previous works [19], we set the dimension of word embeddings to 300 and randomly initialized word embeddings of words not present in the vocabulary. For the position embeddings matrixes, we follow to randomly initialize the position embeddings and determine the dimension of position embeddings heuristically (finally set to 10). The maximal length of the DDI instances is set to 150, that is, the maximal length of sentences in the DDIExtraction 2013 corpus. Following [19], we used three kinds of filters for convolution; that is, k is set to 3, 4, and 5 for filter $\mathbf{t} \in \mathrm{R}(dw+2dp)\times k$, and we used 200 filters of each kind at the convolutional layer. The dropout rate (p), $l2$-norm threshold, and mini batch size are, respectively, set to 0.5, 3, and 50, the same as . Our CNN-based DDI extraction system will be released after the publication of this study. To investigate the effect of different factors, we start with a baseline system without using position embeddings and negative instance filtering module and then add them gradually. We also compare our system with other state-of-the-art systems. The performances of all DDI extraction systems are measured by precision (P), recall (R), and F-score (F), which are calculated by the evaluation tool provided by the 2013 DDIExtraction challenge organizers.

5.2 PROJECT CODE

Read Dataset

read_dataset.py

```
# -*- coding: utf-8 -*-

import glob

import os

from pyexpat import ExpatError

from xml.dom import minidom

import pandas as pd

from nltk.corpus import stopwords

from tqdm import tqdm
```

```
STOP_WORDS = set(stopwords.words('english')) | set('the')

pd.set_option('display.width', 1000)

dataset_csv_file = 'dataset_dataframe.csv'

types = set()

training_dataset_dataframe = None

def get_entity_dict(sentence_dom):

    entities = sentence_dom.getElementsByTagName('entity')

    entity_dict = {}

    for entity in entities:

        id = entity.getAttribute('id')

        word = entity.getAttribute('text')

        entity_dict[id] = word

    return entity_dict


def normalize_sentence(row):

    sentence = row.sentence_text.replace('.', ' . ')

    sentence = sentence.replace(',', ' , ')

    e1 = row.e1

    e2 = row.e2

    new_sentence_tokenized = []

    i = 0

    for word in sentence.split():

        if word in STOP_WORDS:

            continue
        if word.lower() == e1.lower():
            new_sentence_tokenized.append('DRUG')
            i += 1
        elif word.lower() == e2.lower():
            new_sentence_tokenized.append('OTHER_DRUG')
            i += 1
        elif i == 0:
            new_sentence_tokenized.append(word + '_bf')
        elif i == 1:
            new_sentence_tokenized.append(word + '_be')
        else:
            new_sentence_tokenized.append(word + '_af')
    normalized_sentence = ' '.join(new_sentence_tokenized).strip()
    # print(e1, e2, ' :  sentence :', sentence, 'new_sentence', normalized_sentence, '\n\n')
    return normalized_sentence


def get_dataset_dataframe(directory=None):
    global training_dataset_dataframe, dataset_csv_file
    if training_dataset_dataframe:
        return training_dataset_dataframe
    global types
    if directory is None:
        directory                                                                    =
os.path.expanduser('C:/Users/Dell/Desktop/ddi/dataset/DDICorpus/Train/MedLine/')
    dataset_csv_file_prefix = str(directory.split('/')[-3]).lower() + '_'
```

```
dataset_csv_file = dataset_csv_file_prefix + dataset_csv_file
if os.path.isfile(dataset_csv_file):
    df = pd.read_csv(dataset_csv_file)
    return df
lol = []
total_files_to_read = glob.glob(directory + '*.xml')
print('total_files_to_read:' , len(total_files_to_read) , ' from dir: ' , directory)
for file in tqdm(total_files_to_read):
    try:
        DOMTree = minidom.parse(file)
        sentences = DOMTree.getElementsByTagName('sentence')
        for sentence_dom in sentences:
            entity_dict = get_entity_dict(sentence_dom)
            pairs = sentence_dom.getElementsByTagName('pair')
            sentence_text = sentence_dom.getAttribute('text')
            for pair in pairs:
                ddi_flag = pair.getAttribute('ddi')
                #print(pair.attributes().items())
                if not os.path.isfile('types'):
                    types.add(pair.getAttribute('type'))
                if ddi_flag == 'true':
                    e1 = pair.getAttribute('e1')
                    e2 = pair.getAttribute('e2')
                    relation_type = pair.getAttribute('type')
                    lol.append([sentence_text, entity_dict[e1], entity_dict[e2], relation_type])
        except ExpatError:
            pass
    pd.to_pickle(types, 'types')
    df = pd.DataFrame(lol, columns='sentence_text,e1,e2,relation_type'.split(','))
    df['normalized_sentence'] = df.apply(normalize_sentence, axis=1)
    df.to_csv(dataset_csv_file)
    df = pd.read_csv(dataset_csv_file)
    return df


def get_training_label(row):
    global types
    types = pd.read_pickle('types')
    types = [t for t in types if t]
    type_list = list(types)
    relation_type = row.relation_type
    X = [i for i, t in enumerate(type_list) if relation_type == t]
    # s = np.sum(X)
    if X:
        return X[0]
    else:
        return 1
```

GRAMMER

feature_vector.py

```
# -*- coding: utf-8 -*-
```

```
from nltk.util import ngrams

from dataset.read_dataset import get_dataset_dataframe

from grammar.chunker import Chunker

from grammar.syntactic_grammar import PatternGrammar

frequent_word_pairs = None

K = 200

import pandas as pd

from spacy.lang.en import English

parser = English()

import os

from itertools import combinations

from collections import Counter


def get_dataset_dictionary():

    top_post_fixed_word_file = 'top_post_fixed_word.pkl'

    if os.path.isfile(top_post_fixed_word_file):

        return pd.read_pickle(top_post_fixed_word_file)

    df = get_dataset_dataframe()

    word_counter = Counter()

    for _, row in df.iterrows():

        unique_tokens = sorted(set(word for word in row.normalized_sentence.split()))

        # exclude duplicates in same line and sort to ensure one word is always before other

        bi_grams = ngrams(row.normalized_sentence.split(), 2)

        word_counter += Counter([' '.join(bi_gram).strip() for bi_gram in bi_grams])
        word_counter += Counter(unique_tokens)
    frequent_words = sorted(list(dict(word_counter.most_common(100000)).keys()))  # return
the actual Counter object
    pd.to_pickle(frequent_words, top_post_fixed_word_file)
    return frequent_words


def extract_top_word_pair_features():
    frequent_phrase_pickle_path = 'frequent_phrase.pkl'
    if not os.path.isfile(frequent_phrase_pickle_path):
        df = get_dataset_dataframe()
        pair_counter = Counter()
        for _, row in df.iterrows():
            unique_tokens = sorted(set(word for word in row.normalized_sentence.split()))
            # exclude duplicates in same line and sort to ensure one word is always before other
            combos = combinations(unique_tokens, 2)
            pair_counter += Counter(combos)
        frequent_phrase = sorted(list(dict(pair_counter.most_common(K)).keys()))  # return the
actual Counter object
        pd.to_pickle(frequent_phrase, frequent_phrase_pickle_path)
    else:
        frequent_phrase = pd.read_pickle(frequent_phrase_pickle_path)
    print('frequent_phrase: ' , frequent_phrase[:5])
    return frequent_phrase


def extract_top_syntactic_grammar_trio():
    top_syntactic_grammar_trio_file = 'top_syntactic_grammar_trio_file.pkl'

    if os.path.isfile(top_syntactic_grammar_trio_file):

        return pd.read_pickle(top_syntactic_grammar_trio_file)

    df = get_dataset_dataframe()

    trio_counter = Counter()

    for _, row in df.iterrows():

        combos = extract_syntactic_grammar(row.sentence_text)

        trio_counter += Counter(combos)

    frequent_trio_counter = sorted(list(dict(trio_counter.most_common(K)).keys()))   # return
the actual Counter object

    pd.to_pickle(frequent_trio_counter, top_syntactic_grammar_trio_file)

    return frequent_trio_counter


def extract_dependency_relations(sentence):

    # TODO : introduce dependency relation later

    parsedEx = parser(sentence)

    for token in parsedEx:

        print(token.orth_, token.dep_, token.head.orth_)


def extract_syntactic_grammar(sentence):

    grammar = PatternGrammar().get_syntactic_grammar(0)

    chunk_dict = Chunker(grammar).chunk_sentence(sentence)

    trigrams_list = []

    for key, pos_tagged_sentences in chunk_dict.items():
        pos_tags = [token[1] for pos_tagged_sentence in pos_tagged_sentences for token in pos_tagged_sentence]
        if len(pos_tags) > 2:
            trigrams = ngrams(pos_tags, 3)
            trigrams_list = [' '.join(trigram) for trigram in trigrams]
    return trigrams_list
if __name__ == '__main__':
    df = get_dataset_dataframe()
    print(get_dataset_dictionary())
```

Chunker.py

```
# -*- coding: utf-8 -*-
from nltk.util import ngrams
from dataset.read_dataset import get_dataset_dataframe
from grammar.chunker import Chunker
from grammar.syntactic_grammar import PatternGrammar
frequent_word_pairs = None
K = 200
import pandas as pd
from spacy.lang.en import English
parser = English()
import os
from itertools import combinations
from collections import Counter


def get_dataset_dictionary():
    top_post_fixed_word_file = 'top_post_fixed_word.pkl'

    if os.path.isfile(top_post_fixed_word_file):

        return pd.read_pickle(top_post_fixed_word_file)

    df = get_dataset_dataframe()

    word_counter = Counter()

    for _, row in df.iterrows():

        unique_tokens = sorted(set(word for word in row.normalized_sentence.split()))

        # exclude duplicates in same line and sort to ensure one word is always before other

        bi_grams = ngrams(row.normalized_sentence.split(), 2)

        word_counter += Counter([' '.join(bi_gram).strip() for bi_gram in bi_grams])

        word_counter += Counter(unique_tokens)

    frequent_words = sorted(list(dict(word_counter.most_common(100000)).keys()))  # return
the actual Counter object

    pd.to_pickle(frequent_words, top_post_fixed_word_file)

    return frequent_words


def extract_top_word_pair_features():

    frequent_phrase_pickle_path = 'frequent_phrase.pkl'

    if not os.path.isfile(frequent_phrase_pickle_path):

        df = get_dataset_dataframe()

        pair_counter = Counter()

        for _, row in df.iterrows():

            unique_tokens = sorted(set(word for word in row.normalized_sentence.split()))

            # exclude duplicates in same line and sort to ensure one word is always before other

            combos = combinations(unique_tokens, 2)
            pair_counter += Counter(combos)
        frequent_phrase = sorted(list(dict(pair_counter.most_common(K)).keys()))  # return the
actual Counter object
        pd.to_pickle(frequent_phrase, frequent_phrase_pickle_path)
    else:
        frequent_phrase = pd.read_pickle(frequent_phrase_pickle_path)
    print('frequent_phrase: ' , frequent_phrase[:5])
    return frequent_phrase


def extract_top_syntactic_grammar_trio():
    top_syntactic_grammar_trio_file = 'top_syntactic_grammar_trio_file.pkl'
    if os.path.isfile(top_syntactic_grammar_trio_file):
        return pd.read_pickle(top_syntactic_grammar_trio_file)
    df = get_dataset_dataframe()
    trio_counter = Counter()
    for _, row in df.iterrows():
        combos = extract_syntactic_grammar(row.sentence_text)
        trio_counter += Counter(combos)
    frequent_trio_counter = sorted(list(dict(trio_counter.most_common(K)).keys()))  # return
the actual Counter object
    pd.to_pickle(frequent_trio_counter, top_syntactic_grammar_trio_file)
    return frequent_trio_counter


def extract_dependency_relations(sentence):
    # TODO : introduce dependency relation later
    parsedEx = parser(sentence)
    for token in parsedEx:
        print(token.orth_, token.dep_, token.head.orth_)


def extract_syntactic_grammar(sentence):
    grammar = PatternGrammar().get_syntactic_grammar(0)
    chunk_dict = Chunker(grammar).chunk_sentence(sentence)
    trigrams_list = []
    for key, pos_tagged_sentences in chunk_dict.items():
        pos_tags = [token[1] for pos_tagged_sentence in pos_tagged_sentences for token in
pos_tagged_sentence]
        if len(pos_tags) > 2:
            trigrams = ngrams(pos_tags, 3)
            trigrams_list = [' '.join(trigram) for trigram in trigrams]
    return trigrams_list
if __name__ == '__main__':
     df = get_dataset_dataframe()
     print(get_dataset_dictionary())
```

pos_tagger.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import nltk
from nltk import PerceptronTagger
class PosTagger:
    def __init__(self, sentence):
        """
        Args:
            sentence:
        """
        self.sentence = sentence
        self.tagger = PosTagger.get_tagger()

    def pos_tag(self):
        """
        Returns:
        """
        tokens = nltk.word_tokenize(self.sentence)
        pos_tagged_tokens = self.tagger.tag(tokens)
        return pos_tagged_tokens
    @staticmethod
    def get_tagger():
        """
        Returns:
        """
        return PerceptronTagger()
```

syntactic_grammer.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import nltk
```

```
syntactic_compiled_grammar = {}

class PatternGrammar:

    @property

    def syntactic_grammars(self):

        grammar = {

            0: """

                JJ_VBG_RB_DESCRIBING_NN:                                                    {
(<CC|,>?<JJ|JJ.>*<VB.|V.>?<NN|NN.>)+<RB|RB.>*<MD>?<WDT|DT>?<VB|VB.>?<RB|
RB.>*(<CC|,>?<RB|RB.>?<VB|VB.|JJ.|JJ|RB|RB.>+)+}

                """,

            1: """

                    VBG_DESRIBING_NN: {<NN|NN.><VB|VB.>+<RB|RB.>*<VB|VB.>}

                """,

        }

        return grammar


    def get_syntactic_grammar(self, index):

        global syntactic_compiled_grammar

        compiled_grammar = syntactic_compiled_grammar.get(index, None)

        if compiled_grammar is None:

            compiled_grammar = self.compile_syntactic_grammar(index)

            syntactic_compiled_grammar[index] = compiled_grammar

        return compiled_grammar


    def compile_syntactic_grammar(self, index):
        return nltk.RegexpParser(self.syntactic_grammars[index])
```

Test

predict_on_test_dataset.py

```
from sklearn.metrics import classification_report
from dataset.read_dataset import get_dataset_dataframe
from training.train import extract_training_data_from_dataframe, trained_model_pickle_file
import pandas as pd
import os
def predict():
    df = get_dataset_dataframe(directory=os.path.expanduser('C:/Users/Dell/Desktop/ddi/dataset/DDI Corpus/Test/test_for_ddi_extraction_task/MedLine/'))
    X, Y = extract_training_data_from_dataframe(df)
    model = pd.read_pickle(trained_model_pickle_file)
    y_pred  = model.predict(X)
    print(classification_report(Y, y_pred))
```

Train

train.py

```
# -*- coding: utf-8 -*-
from itertools import combinations
from nltk import ngrams
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np
dataset_dictionary = None
```

```
top_word_pair_features = None

top_syntactic_grammar_list = None

trained_model_pickle_file = 'trained_model.pkl'


def get_empty_vector(n):

    return [0 for _ in range(n)]


def get_top_word_dataset_dictionary():

    from feaure_extraction.feature_vector import get_dataset_dictionary

    global dataset_dictionary

    if dataset_dictionary is None:

        dataset_dictionary = get_dataset_dictionary()

    return dataset_dictionary


def get_top_word_pair_features():

    from feaure_extraction.feature_vector import extract_top_word_pair_features

    global top_word_pair_features

    if top_word_pair_features is None:

        top_word_pair_features = extract_top_word_pair_features()

    return top_word_pair_features


def get_top_syntactic_grammar_list():

    from feaure_extraction.feature_vector import extract_top_syntactic_grammar_trio

    global top_syntactic_grammar_list
    if top_syntactic_grammar_list is None:
        top_syntactic_grammar_list = extract_top_syntactic_grammar_trio()
    return top_syntactic_grammar_list


def get_word_feature(normalized_sentence):
    unique_tokens = set(word for word in normalized_sentence.split())
    # exclude duplicates in same line and sort to ensure one word is always before other
    bi_grams = set(ngrams(normalized_sentence.split(), 2))
    words = unique_tokens | bi_grams
    dataset_dictionary = get_top_word_dataset_dictionary()
    X = [i if j in words else 0 for i, j in enumerate(dataset_dictionary)]
    return X


def get_frequent_word_pair_feature(normalized_sentence):
    unique_tokens = sorted(set(word for word in normalized_sentence.split()))
    # exclude duplicates in same line and sort to ensure one word is always before other
    combos = combinations(unique_tokens, 2)
    top_word_pair_features = get_top_word_pair_features()
    X = [i if j in combos else 0 for i, j in enumerate(top_word_pair_features)]
    return X


def get_syntactic_grammar_feature(sentence_text):
    from feaure_extraction.feature_vector import extract_syntactic_grammar
    trigrams_list = extract_syntactic_grammar(sentence_text)
    top_syntactic_grammar_list = get_top_syntactic_grammar_list()
    X = [i if j in trigrams_list else 0 for i, j in enumerate(top_syntactic_grammar_list)]
    return X


def make_feature_vector(row):
    normalized_sentence = row.normalized_sentence
    sentence = row.sentence_text
    word_feature = get_word_feature(normalized_sentence)
    frequent_word_feature = get_frequent_word_pair_feature(normalized_sentence)
    syntactic_grammar_feature = get_syntactic_grammar_feature(sentence)
    features = word_feature
    features.extend(frequent_word_feature)
    features.extend(syntactic_grammar_feature)
    return features


def main():
    from dataset.read_dataset import get_dataset_dataframe
    df = get_dataset_dataframe()
    X, Y = extract_training_data_from_dataframe(df)
    from sklearn.svm import SVC
    X_train, X_test, y_train, y_test = \
        train_test_split(X, Y, test_size=.2, random_state=42)
    print(df.head())
    print('X: ', (X.shape), 'Y : ', np.array(Y.shape))
    model = SVC(kernel='linear')

    model.fit(X_train, y_train)

    score = model.score(X_test, y_test)

    import pandas as pd

    pd.to_pickle(model, trained_model_pickle_file)

    #classification_report()

    print('Score : ', score)

    y_pred = model.predict(X_test)

    print(classification_report(y_test, y_pred))


def extract_training_data_from_dataframe(df):

    from dataset.read_dataset import get_training_label

    X = df.apply(make_feature_vector, axis=1)

    Y = df.apply(get_training_label, axis=1)

    X = np.array(X.tolist())

    Y = np.array(Y.tolist())

    return X, Y
```

Main Code

Main.py

```
from training.train import main as training_main

from test.predict_on_test_dataset import predict

import pandas as pd

pd.set_option('display.width', 1000)

training_main()

predict()
```

CHAPTER-6
RESULTS

6. RESULTS

6.1 RESULTS OF TRAINING DATA

An Excel file will be created after running train.py

train_dataset_dataframe.csv

S.No	sentence_text	e1	e2	relation_type	normalized_sentence
1	Concurrent administration of a TNF antagonist with ORENCIA has been associated with an increased risk of serious infections and no significant additional efficacy over use of the TNF antagonists alone.	TNF antagonist	ORENCIA	effect	Concurrent_bfadministration_bfTNF_bfantagonist_bf OTHER_DRUG associated_beincreased_berisk_beserious_beinfections_besignificant_beadditional_beefficacy_beuse_beTNF_beantagonists_bealone_be ._be
2	Concurrent therapy with ORENCIA and TNF antagonists is not recommended.	ORENCIA	TNF antagonists	advise	Concurrent_bftherapy_bf DRUG TNF_beantagonists_berecommended_be ._be
3	There is insufficient experience to assess the safety and efficacy of ORENCIA administered	ORENCIA	anakinra	advise	There_bfinsufficient_bfexperience_bfassess_bfsafety_bfefficacy_bf DRUG administered_beconcurrently_be OTHER_DRUG

	concurrently with anakinra, and therefore such use is not recommended.				,_aftherefore_afuse_afrecom mended_af ._af
4	Co-administration of naltrexone with Acamprosate produced a 25% increase in AUC and a 33% increase in the Cmax of acamprosate.	naltrexon e	Acampros ate	mechan ism	Co-administration_bf DRUG OTHER_DRUG produced_af 25%_af increase_afAUC_af 33%_af increase_afCmax_af OTHER_DRUG ._af
5	Patients taking Acamprosate concomitantly with antidepressants more commonly reported both weight gain and weight loss, compared with patients taking either medication alone.	Acampro sate	antidepres sants	effect	Patients_bftaking_bf DRUG concomitantly_be OTHER_DRUG commonly_afreported_afwei ght_afgain_afweight_afloss_a f ,_afcompared_afpatients_afta king_afeither_afmedication_a falone_af ._af
6	Intestinal adsorbents (e. g., charcoal) and digestive enzyme preparations containing carbohydrate-splitting enzymes (e. g., amylase, pancreatin) may reduce the effect of Acarbose and	Intestinal adsorben ts	Acarbose	mechan ism	Intestinal_bfadsorbents_bf (e_bf ._bf g_bf ._bf ,_bf charcoal)_bf digestive_bfenzyme_bfprepar ations_bfcontaining_bf carbohydrate- splitting_bfenzymes_bf (e_bf ._bf g_bf ._bf ,_bf amylase_bf ,_bf pancreatin)_bf may_bfreduce_bfeffect_bf

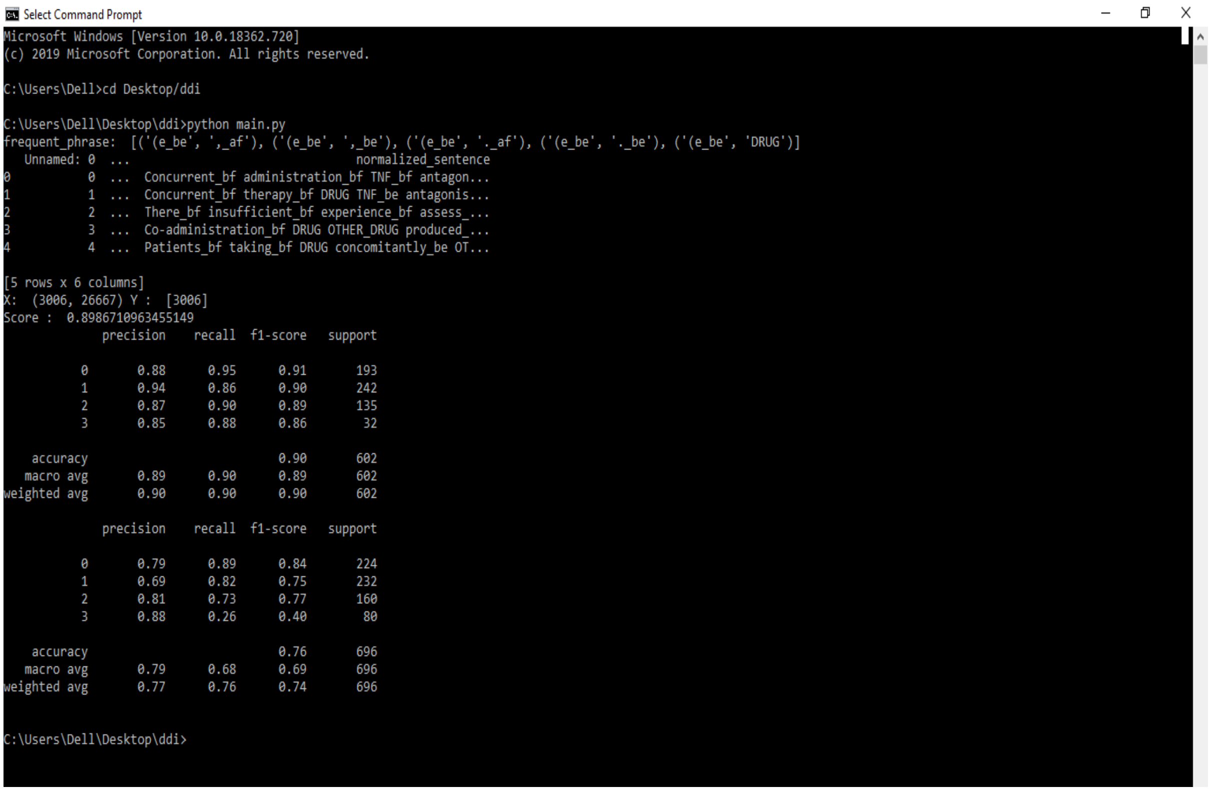
	should not be taken concomitantly.				OTHER_DRUG taken_beconcomitantly_be ._be
7	Intestinal adsorbents (e. g., charcoal) and digestive enzyme preparations containing carbohydrate-splitting enzymes (e. g., amylase, pancreatin) may reduce the effect of Acarbose and should not be taken concomitantly.	charcoal	Acarbose	mechanism	Intestinal_bfadsorbents_bf (e_bf ._bf g_bf ._bf ,_bf charcoal)_bf digestive_bfenzyme_bfpreparations_bfcontaining_bf carbohydrate-splitting_bfenzymes_bf (e_bf ._bf g_bf ._bf ,_bf amylase_bf ,_bf pancreatin)_bf may_bfreduce_bfeffect_bf OTHER_DRUG taken_beconcomitantly_be ._be
8	However, the peak plasma level of metformin was reduced by approximately 20% when taking Acarbose due to a slight delay in the absorption of metformin.	metformin	Acarbose	mechanism	However_bf ,_bf peak_bfplasma_bflevel_bf DRUG reduced_beapproximately_be 20%_be taking_be OTHER_DRUG due_afslight_afdelay_afabsorption_af DRUG ._af
9	Catecholamine-depleting drugs, such as reserpine, may have an additive effect when given with beta-blocking agents.	Catecholamine-depleting drugs	beta-blocking agents	effect	Catecholamine-depleting_bfdrugs_bf ,_bf reserpine_bf ,_bf may_bfadditive_bfeffect_bfgiven_bf beta-blocking_bfagents_bf ._bf

10	DIAMOX modifies phenytoin metabolism with increased serum levels of phenytoin.	DIAMOX	phenytoin	mechanism	DRUG modifies_be OTHER_DRUG metabolism_afincreased_afserum_aflevels_af OTHER_DRUG ._af
11	Acetazolamide may increase the effects of other folic acid antagonists.	Acetazolamide	folic acid antagonists	effect	DRUG may_beincrease_beeffects_be folic_beacid_beantagonists_be ._be
12	Co-administration of probenecid with acyclovir has been shown to increase the mean half-life and the area under the concentration-time curve.	probenecid	acyclovir	mechanism	Co-administration_bf DRUG OTHER_DRUG shown_afincrease_afmean_af half-life_afarea_af concentration-time_afcurve_af ._af
13	Ethanol:Clinical evidence has shown that etretinate can be formed with concurrent ingestion of acitretin and ethanol.	acitretin	ethanol	mechanism	Ethanol:Clinical_bfevidence_bfshown_bfetretinate_bfformed_bfconcurrent_bfingestion_bf DRUG OTHER_DRUG ._af
14	Nafazodone, fluvoxamine, cimetidine (consider Xanax dose reduction).	Nafazodone	Xanax	advise	DRUG ,_be fluvoxamine_be ,_be cimetidine_be (consider_be OTHER_DRUG dose_af reduction)_af ._af

Table 6.1 Output excel file data

6.2 OUTPUT

In command prompt



```
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd Desktop/ddi

C:\Users\Dell\Desktop\ddi>python main.py
frequent_phrase:  [('(e_be', ',_af'), ('(e_be', ',_be'), ('(e_be', '._af'), ('(e_be', '._be'), ('(e_be', 'DRUG')]
   Unnamed: 0  ...                                normalized_sentence
0           0  ...  Concurrent_bf administration_bf TNF_bf antagon...
1           1  ...  Concurrent_bf therapy_bf DRUG TNF_be antagonis...
2           2  ...  There_bf insufficient_bf experience_bf assess_...
3           3  ...  Co-administration_bf DRUG OTHER_DRUG produced_...
4           4  ...  Patients_bf taking_bf DRUG concomitantly_be OT...

[5 rows x 6 columns]
X:  (3006, 26667) Y :  [3006]
Score :  0.8986710963455149
              precision    recall  f1-score   support

           0       0.88      0.95      0.91       193
           1       0.94      0.86      0.90       242
           2       0.87      0.90      0.89       135
           3       0.85      0.88      0.86        32

    accuracy                           0.90       602
   macro avg       0.89      0.90      0.89       602
weighted avg       0.90      0.90      0.90       602

              precision    recall  f1-score   support

           0       0.79      0.89      0.84       224
           1       0.69      0.82      0.75       232
           2       0.81      0.73      0.77       160
           3       0.88      0.26      0.40        80

    accuracy                           0.76       696
   macro avg       0.79      0.68      0.69       696
weighted avg       0.77      0.76      0.74       696


C:\Users\Dell\Desktop\ddi>
```

Fig 6.1 Output

CHAPTER-7 CONCLUSION AND FUTURE IMPLEMENTATION

7. CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

This project aims to extract ADRs from user reviews. In this work, we will propose a novel approach to extracting adverse drug reactions from the user reviews: a combination of a bidirectional LSTM –based recurrent neural network and CRF that operates on the scores extracted by this neural networks. Moreover, future improvements may obtain by extending input embeddings with a character-level model. In future we will combine different approaches to statistical modeling in NLP and we plan to extend our work with other neural models by using deep learning techniques in ADR extraction from free-text reviews.

FUTURE ENHANCEMENT

Future improvements may obtain by extending input embeddings with a character-level model. In future we will combine different approaches to statistical modeling in NLP and we plan to extend our work with other neural models by using deep learning techniques in ADRs extraction from free-text reviews.

REFERENCES

[1] Chang Liu, Chi Yang, Xuyun Zhang, and Jinjun Chen. External integrity verification for outsourced big data in cloud and iot: A big picture. FutureGenerationComputerSystems, 2015.

[2] Katina Michael and Keith W Miller. Big data: New opportunities and new challenges [guest editors' introduction]. Computer, 46(6):22–24, 2013.

[3] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2016.

[4] C.J. McCall, "Big data in healthcare: Hype or hope," A White Paper by Carol J McCall, 2015.

[5] O. Iroju, A. Soriyan, I. Gambo, and J. Olaleke, "Interoperability in healthcare: benefits, challenges and resolutions," International Journal of Innovation and Applied Studies, vol. 3, pp. 262-270, 2013.

[6] O.G. Iroju, and J.O. Olaleke, "A systematic review of natural language processing in healthcare," International Journal of Information Technology and Computer Science, vol.8, pp. 44-48, 2015

[7] M. S. Islam, M. M. Hasan, X. Wang, and H. D. Germack, ''A systematic review on healthcare analytics: Application and theoretical perspectiveof data mining,'' Healthcare, vol. 6, no. 2, p. 54, 2018.

[8] Y. Chen et al., ''Machine-learning-based classification of real-time tissue elastography for hepatic fibrosis in patients with chronic hepatitis B,'' Comput. Biol. Med., vol. 89, pp. 18–23, Jan.2017.

[9] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis,and D. I. Fotiadis, ''Machine learning applications in cancer prognosis and prediction,''Comput.Struct.Biotechnol.J.,vol.13,pp.8–17,Dec.2015.

[10] M.M. Hansen, T. Miron-Shatz, A.Y. Lau, C. Paton, Big data in science and healthcare: a review of recent literature and perspectives, in: Contribution of the IMIA Social Media Working Group, Yearb. Med. Inform. 9 (2014)21–26.

[11] L. Duan, W.N. Street, E. Xu, Healthcare information systems: data mining methods in the creation of a clinical recommender system, Enterp. Inf. Syst. 5 (2011) 169–181.R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., inpress.

[12] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions, IEEE Trans. Knowl. Data Eng. 17 (2005)734–749

[13] A. Ben Abacha, M. F. M. Chowdhury, A. Karanasiou, Y. Mrabet, A. Lavelli, and P. Zweigenbaum, “Text mining for pharmacovigilance: Using machine learning for drug name recognition and drug-drug interaction extraction and classification,” J. Biomed. Inform., vol. 58, pp. 122–132,2015.

[14] S. Liu, B. Tang, Q. Chen, and X. Wang, “Drug-drug Interaction Extraction via Convolutional Neural Networks,” Comput. Math. Methods Med., vol. 2016, pp. 1–8,2016.

[15] Z. Zhao, Z. Yang, L. Luo, H. Lin, and J. Wang, “Drug drug interaction extraction from biomedical literature using syntax convolutional neural network,” Bioinformatics, vol. 32, no. 22, pp. 3444–3453,2016.

[16] S.Kim,H.Liu, L.Yeganova, and W.J.Wilbur, “Extracting drug- drug interactions from literature using a rich feature-based linear kernel approach, ”Journal of Biomedical Informatics, vol. 55, pp. 23–30,2015.

[17] Evaluation scripts of 2013 DDI Extraction challenge, https://www.cs.york.ac.uk/semeval-task9/index.php%3Fid=evaluation.html.

[18] MEDLINE, http://www.nlm.nih.gov/databases/journal.html.

[19] K.He,X.Zhang, S.Ren, and J.Sun, “Delving deep into rectifiers: surpassing human-level performance on imagenet classification,” in Proceedings of the IEEE International Conference on Computer Vision, pp.1026–1034,Santiago,Chile,2015.

[20] E. Tutubalina and S. Nikolenko, “Automated prediction of demographic information from medical user reviews,” Inter- national Conference on Mining Intelligence and Knowledge Exploration, pp. 1–11, Springer,2016.

[21] https://www.askapatient.com