Final Report

**Is it a real piece of cake? Automatic Detection of Idiomatic Language**

Submitted as part of the requirements for:

**CE903 Group Project**

**Group (5) Member Names**

|  |  |
| --- | --- |
| Name | Registration Number |
| Atiwat Onsuwan | 1802514 |
| Junxiao Li | 1807844 |
| Nimit Khanna | 1806776 |
| Omar Seerwani Nooruldeen Meriwani | 1802670 |
| Owen Daynes | 1804656 |
| Ricardo González Martínez | 1802336 |
| Yaowei Lyu | 1802697 |

**Supervisor Name**: Dr. Aline Villavicencio

**Date**: 21-Mar-2019

**Version (1)**  **Date: (**21/03/2019)

# Table of Contents

[Table of Contents 2](#_Toc3727056)

[1. Introduction 3](#_Toc3727057)

[1.1 Problem and project objectives 3](#_Toc3727058)

[1.2 Methodology 3](#_Toc3727059)

[1.3 Platform and Tools for Project Development 4](#_Toc3727060)

[2. System Design 5](#_Toc3727061)

[2.1 Requirements 5](#_Toc3727062)

[2.2 System Architecture 6](#_Toc3727063)

[2.3 System Components 6](#_Toc3727064)

[2.4 Diagrams (Use case, data flow and GUI) 6](#_Toc3727065)

[2.5 Algorithm Design 6](#_Toc3727066)

[3. Implementation 6](#_Toc3727067)

[3.1 Programming Language 6](#_Toc3727068)

[3.2 Implementation details and key components 6](#_Toc3727069)

[3.3 Overview of code listings 6](#_Toc3727070)

[3.4 Tools used for development 6](#_Toc3727071)

[4. Testing 6](#_Toc3727072)

[4.2 Testing Strategy 8](#_Toc3727073)

[4.2.1 Unit Testing 8](#_Toc3727074)

[4.2.2 Integration Testing 8](#_Toc3727075)

[4.2.3 Acceptance Testing 8](#_Toc3727076)

[4.3 Test Results 8](#_Toc3727077)

[5. Conclusions 9](#_Toc3727078)

[5.1 About the System 9](#_Toc3727079)

[5.2 About the Methodologies and Tools Used 9](#_Toc3727080)

[5.3 About Project Management 9](#_Toc3727081)

[6. References 9](#_Toc3727082)

# 1. Introduction

The following document describes the detail and characteristics of the system developed for the project “Idiomatic Expression Identification.” It provides an overview of the motivation to develop the project and an insight into the system by specifying development procedures and structure to illustrate the architecture of the system better.

## 1.1 Problem and project objectives

Natural language engineering has become a recent field of study that has gained traction in the technological area because of its usefulness when handling daily activities. Although several methods have been developed to allow machines to understand better spoken and written language, ambiguity is still one of the biggest challenges this area faces because in the end machines are still unable to understand all of the signals and non-verbal indications that as humans we can perceive.

Based on the previous statement, this project proposes the development of a system that is trained to identify idiomatic language inside of a provided corpus. There are certain verb-noun combinations found within the English language that make up commonly used idioms. However, ambiguity plays a role since this combination is not always used in an idiomatic way and sometimes are rather used to express a literal definition. The system proposed will attempt to differentiate the use of these combinations depending on whether their use is for idiomatic expression or literal meaning.

## 1.2 Methodology

To accomplish the described objectives, a machine learning system will be created which will be trained by using the British National Corpus as well as a verb-noun combination annotated dataset. The system should focus on data pre-processing to extract the relevant training features out of the dataset and use word-embeddings to train a machine learning algorithm which can identify idiomatic from literal use.

Most of the focus of the system will be on testing word-embeddings such as Word2Vec to observe how effective the system is and how performance could be improved by considering additional features of the sentences such as part-of-speech tagging. Additional testing of the model will be carried out by using the model to annotate new sentences which will then be given to a native speaker who can evaluate how good the model is at identifying the idiomatic language.



**Figure:** Methodology of our Developed System

## 1.3 Platform and Tools for Project Development

The main tool for the development of the project will be Python because of the extensive number of external modules that can be used for the realization of the project. The compatibility with word-embedding models such as Word2Vec as well as the machine learning framework sci-kit-learn makes it an ideal option for this project.

Other collaboration tools were used, such as Google Documents, to allow the sharing and creation of files among all team members.

|  |  |  |
| --- | --- | --- |
| **S.No** | **Module/ Event** | **Tools used** |
| 1. | **Pre-processing** | Python, Pycharm, NLTK, Numpy and Pandas as Python Libraries. |
| 2. | **Processing** | NLTK, Machine Learning Libraries like Scikit-learn. |
| 3. | **Designing** | Python Framework |
| 4 | **Testing** | Microsoft Excel as Test Case log recording |

# 2. System Design

## 2.1 Requirements

In this section, we will meet all the system requirement as we stated in our First stage of the Project, i.e., SRS (Software Requirement Specification) [ ]. Now we will see how these requirements are met as per our project requirement.

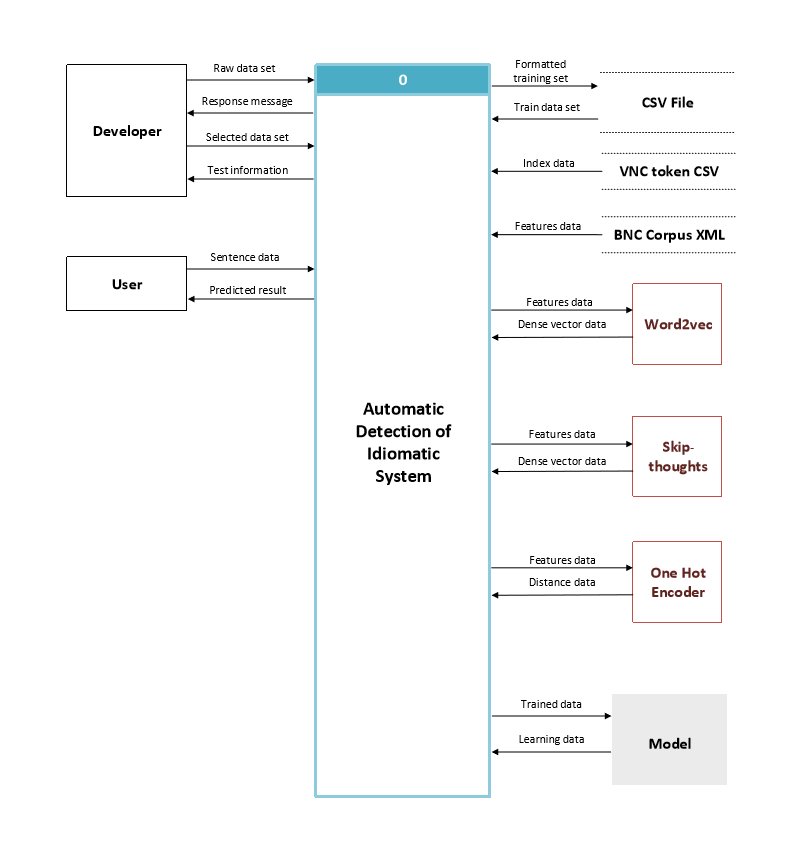
2.1.1 **Feasibility Study:** As we have mentioned in our SRS that we will investigate all the requirements like user, resource and cost related to our project (idiomatic detection). We have successfully archived our feasibility and developed the project.

The various feasibilities we have archived in terms of system design are as follows:

* **Economic Feasibility:** In terms of economic our system is developed on very less cost and operation of our system are also minimum.
* **Operational Feasibility:** In terms of operations our system can operate, and user-friendly without any errors encountered while performing various tasks on our system, the main task of our system is to find idiomatic sentences.
* **Technical Feasibility:** Technicality of our system is meeting all the software and hardware capability and producing the desired goal.

## 2.2 System Architecture

An overview of automatic identification of idiomatic language system starts from the collect corpus then feature extraction to train the machine learning model using the bag of word technique that can be used to detect the input sentence if it is idiomatic or literal.



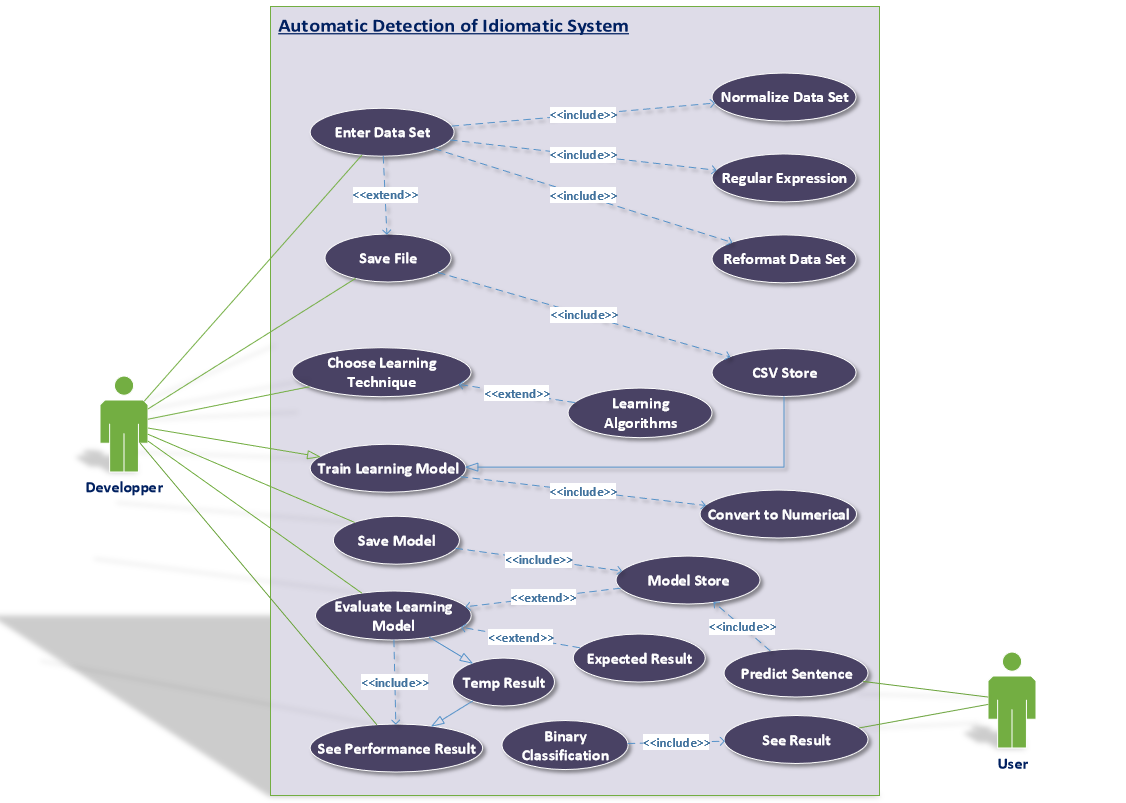
**Figure ?:** System Architecture

## 2.3 System Components

## 2.4 Diagrams (Use case, data flow, and GUI)

2.4.1 **Use Case Diagram**

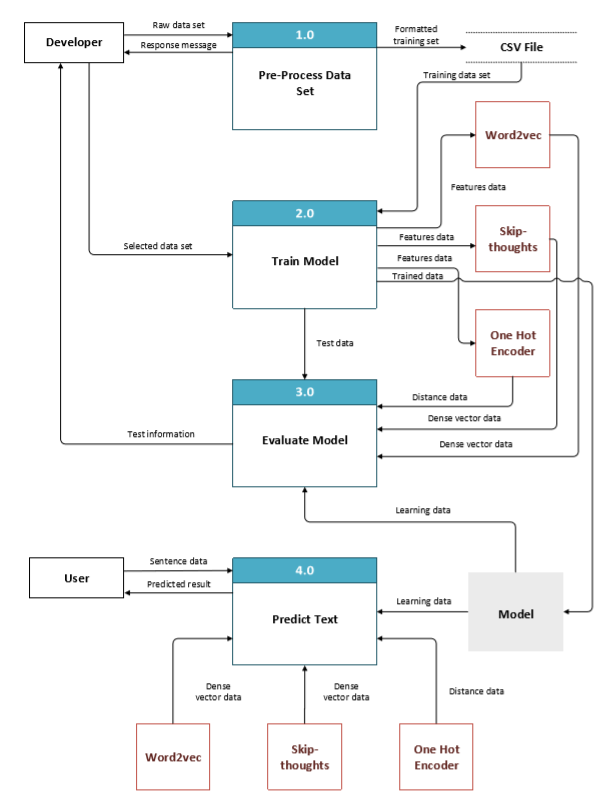
The use case diagram shows a set of action between actors and the system.



**Figure ?:** Use Case Diagram

2.4.2 **Data Flow Diagram Level 0**

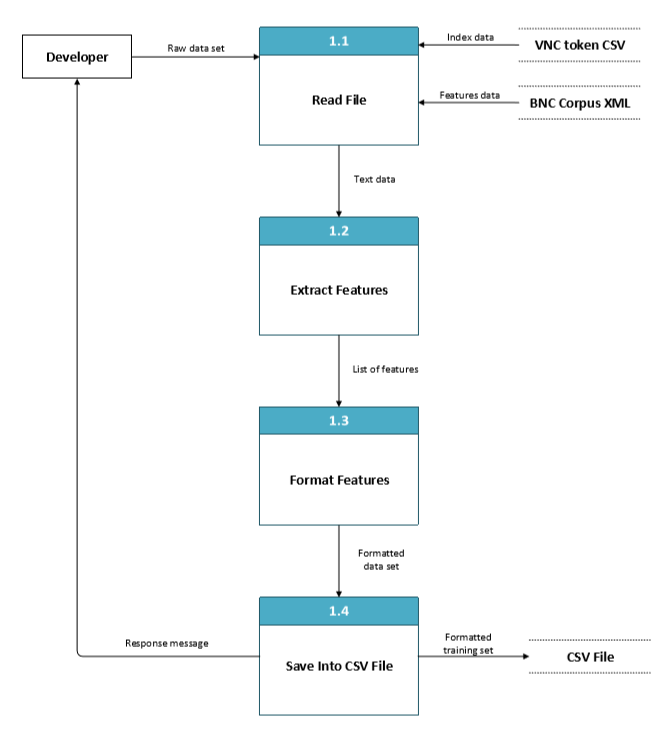
The data flow diagram level 0 indicates overall the flow of data of every process in the system.



**Figure ?:** Data Flow Diagram Level 0

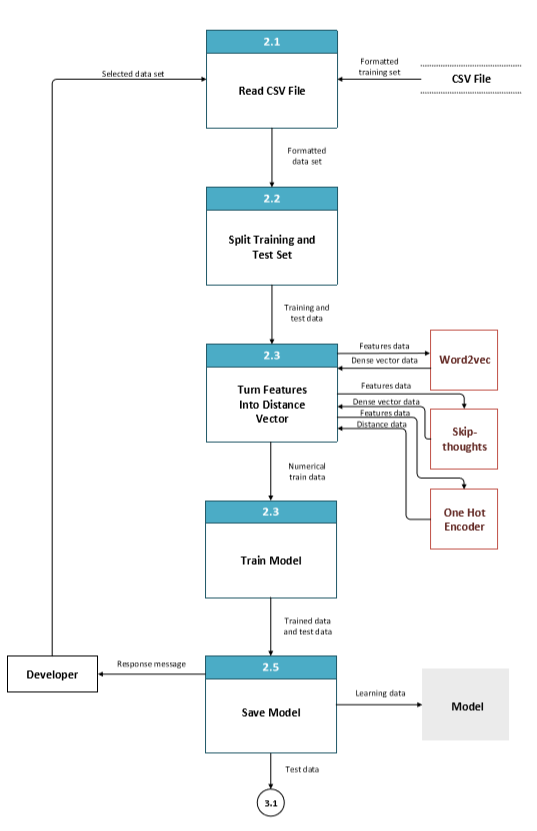
2.4.3 **Data Flow Diagram Level 1 Pre-Process**

The data flow diagram level 1 Pre-Process shows how the system pre-processes the data set.

**Figure ?:** Data Flow Diagram Level 1 Pre-Process

2.4.4 **Data Flow Diagram Level 1 Train Model**

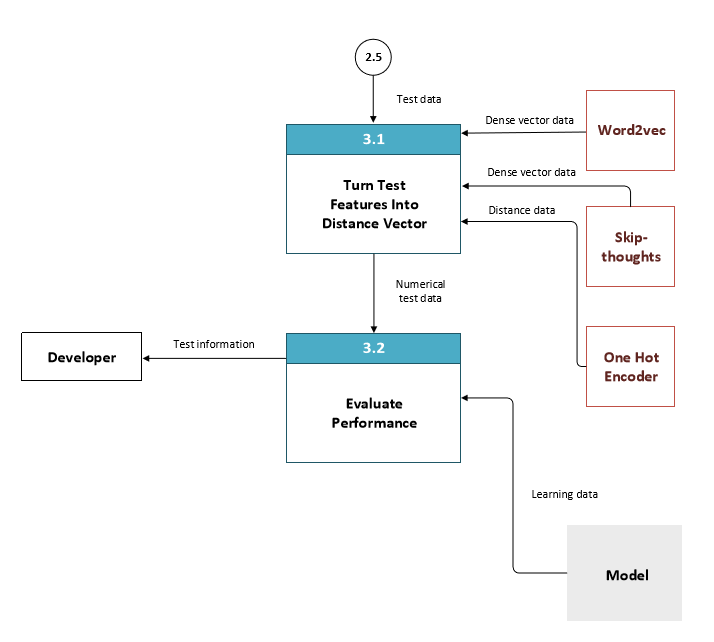
The data flow diagram level 1 Train Model shows how the system trains the bag-of-words model and machine learning model.



**Figure ?:** Data Flow Diagram Level 1 Train Model

2.4.6 **Data Flow Diagram Level 1 Evaluate Model**

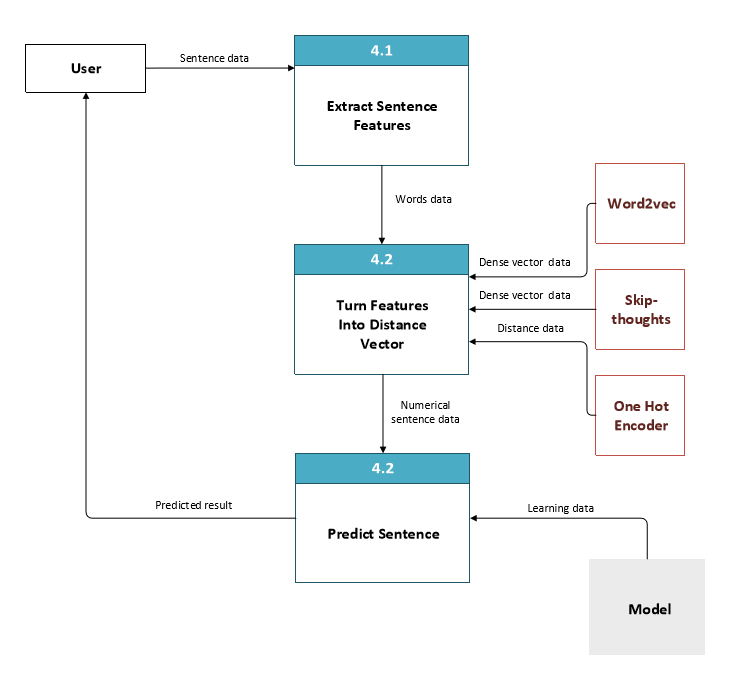
The data flow diagram level 1 Evaluate Model shows how the system evaluates the performance of the learning model.



**Figure ?:** Data Flow Diagram Level 1 Evaluate Model

2.4.7 **Data Flow Diagram Level 1 Predict Text**

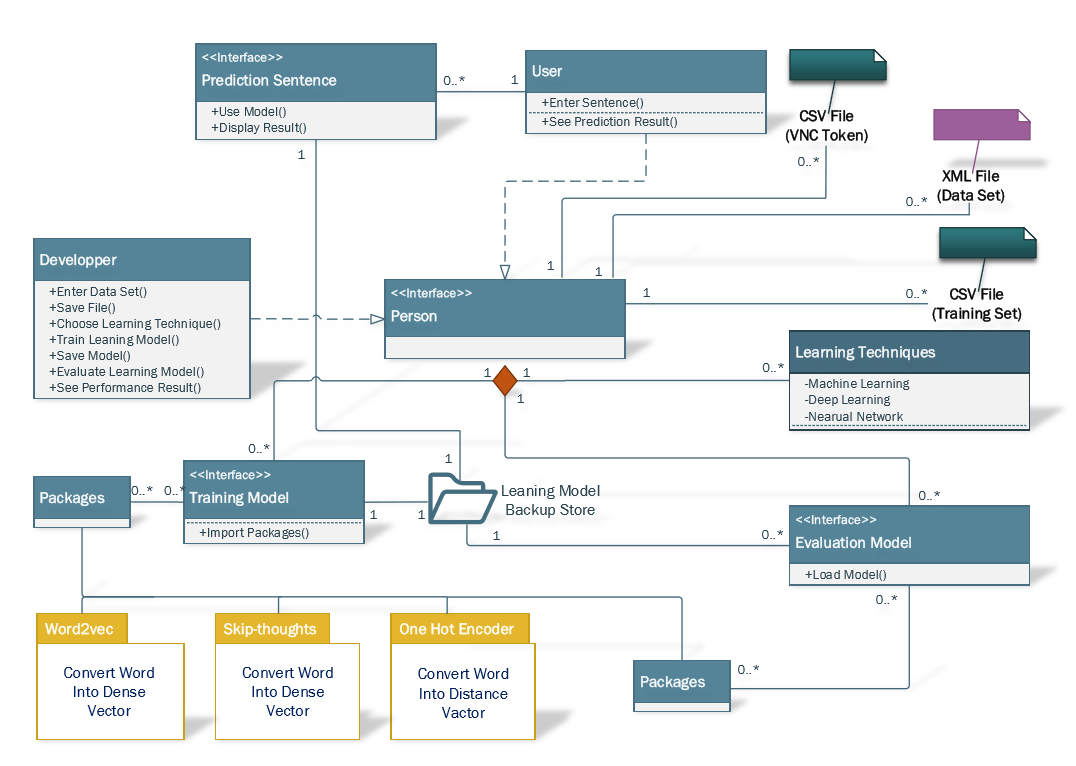
The data flow diagram level 1 Predict Text shows how the system predicts the class of input sentences.



**Figure ?:** Data Flow Diagram Level 1 Predict Text

2.4.8 **Class Diagram**

The class diagram below describes the structure of the system by showing the classes, the attributes, the operation, and the relationship between each object.

****

**Figure ?:** Class Diagram

## 2.5 Algorithm Design

# 3. **Implementation**

## 3.1 **Programming Language**

* In our case, we are going to use the Python3 to implement the tasks the compiler we used is Pycharm and Jupyter notebook.

## 3.2 **Implementation details and key components**

* **Pre-processing**

In this part, first, we read the ‘VNC-tokens.csv ’ by using the pandas which contain the index of the position in the BNC where we can find the sentence. Then we created the CSV file called ‘Dataset ’ which contains the index such as ‘sentences’, ‘idiom’, ‘BetweenPos’, ‘prePos’, ‘prePos2’, ‘postPOS’, ‘postPOS2’, ‘usage’ and other sour features. ‘sentences’ is the sentence we extract from the BNC. ‘BetweenPos’ is the POS\_tag of words between the phrases. ‘prePOS’ point out the first word’s POS\_tag before the phrase and ‘prePOS2’ means the second words. ‘postPOS’ and ‘postPOS2’ mean the POS\_tag of the post words. The final one is ‘usage’ is going to show the usage of the VNC (idiomatic or literal).

* **Train model**

In this part, we need to read the formatted ‘Dataset’ CSV file first by using the Pandas package. After we read the data, we are going to use the train\_test\_split method in the scikilearn package to get the testing and training data set. After that, we are going to convert the features into the vector by using three methods: Word2vec, skip-thought and One Hot Encoder.

* + **Word2vec**

In the Gensim, there is a method called Word2vec which can convert the word into vector. First, we need to put the word into a list. After that we extracted twelve features: Sentence’, ‘Idiom’, ‘NumOfWordsBetween’ , ‘BetweenPOS’, ‘PrePOS’, ‘prePOS2’, ‘postPOS’ , ‘postPOS2’, ‘CountOfSameNounInContext’, ‘SentenceLength’, ‘SentimentsAVG’ and ‘Usage’ of the sentence, then we turn into the dense vector.

* + **One Hot Encoder**

In the scikilearn library we got a method called preprocessing which contains the one hot encoder. We need the dense vector, so the sparse of OneHotEncoder should be false. We have done the same thing as word2vec turns the features of the sentence into the vector.

* + **Skip-thought**

In the skip-thought package, we used the encoder\_manager to turn the features of the sentence into the vector.

**After we turn the feature data into a dense vector, we train the model.**

* + **SVM (Support Vector machine)**

First, we got the train data set by using the train\_test\_split function in scikilearn package. After that, we train the SVM model by using SVM.fit().

* + **Logistic Regression**

First, we got the train data set by using the train\_test\_split function in scikilearn package. Then we train the Logistic Regression model and save it.

* **Evaluation Model**

In this part, we evaluated the model. First, we get the test data by using the train\_test\_split from the ‘Dataset.csv’ file. We also turn feature data into the vector using the word2vec, One Hot Encoder, and skip-thought. Then we got the results of the evaluation.

* + **Accuracy**

In this part, we are going to use the accuracy\_score from the scikilearn library. First, we get the predicted data by using the saved model. Then compare with the test data and get the accuracy score by using the accuracy\_score function.

* + **Kappa Statistic**

We used the cohen\_kappa\_score from the scikilearn. Matrics package. First, we get the predicted data from the train model, then compare with the test data. Finally, we use the cohen\_kappa\_score function to get the kappa statistic.

* + **Confusion Matrix**

The confusion matrix is a method that allows visualization of the performance of an algorithm. In the case, we used the crosstab method which from Pandas package. The x-row of the matrix is the true data and the y-row is the predicted data.

* + **Classification Report**

Classification report is a function of scikilearn package which gives the precision, recall and F-1 scores. First, we get the predicted data and the test data. Then we put in the classification\_report functions, and it gives the results.

* **Prediction Model**

First, we generate a file called Example\_sentences.txt which contains the sentence label is ‘Q’ in the ‘VNC-tokens’ file. After we extracted the sentence from the dataset, the next step is to turn the sentence into the vector by using three methods we already mentioned: word2vec, skip-thought and One Hot Encoder. We can get the results by using the trained model to get the prediction of the sentence is idiomatic or literal.

**(Screenshots)**

## 3.3 **Overview of code listings**

|  |  |
| --- | --- |
| **Module** | **Description** |
| Pre\_process.py | This task is going to preprocess the data and remove the undefine feature and get all the sentences and get the all the useful feature |
| Onehot\_encoder\_SVM.py | This task is going to turn the feature into the vector by using One Hot Encoder and then classifier by SVM |
| Word2Vec\_SVM.py | This task is going to turn the feature into the vector by using word2vec r and then classifier by SVM |
| W2V+Logist\_on\_example\_set.py | This task is going to turn the feature into the vector by using word2vec and then classified by Logistic Regression |
| BNCxmlreader.py | This task is going to read the data from the BNC and get the VNC as the train data |
| st.py | This task is encoder feature data into the vector |
| SkipthoughtsNB.py | This task is going to train the skip-thought model by using the VNC and classifier by using the naïve Bayes |

## 3.4 Tools used for development

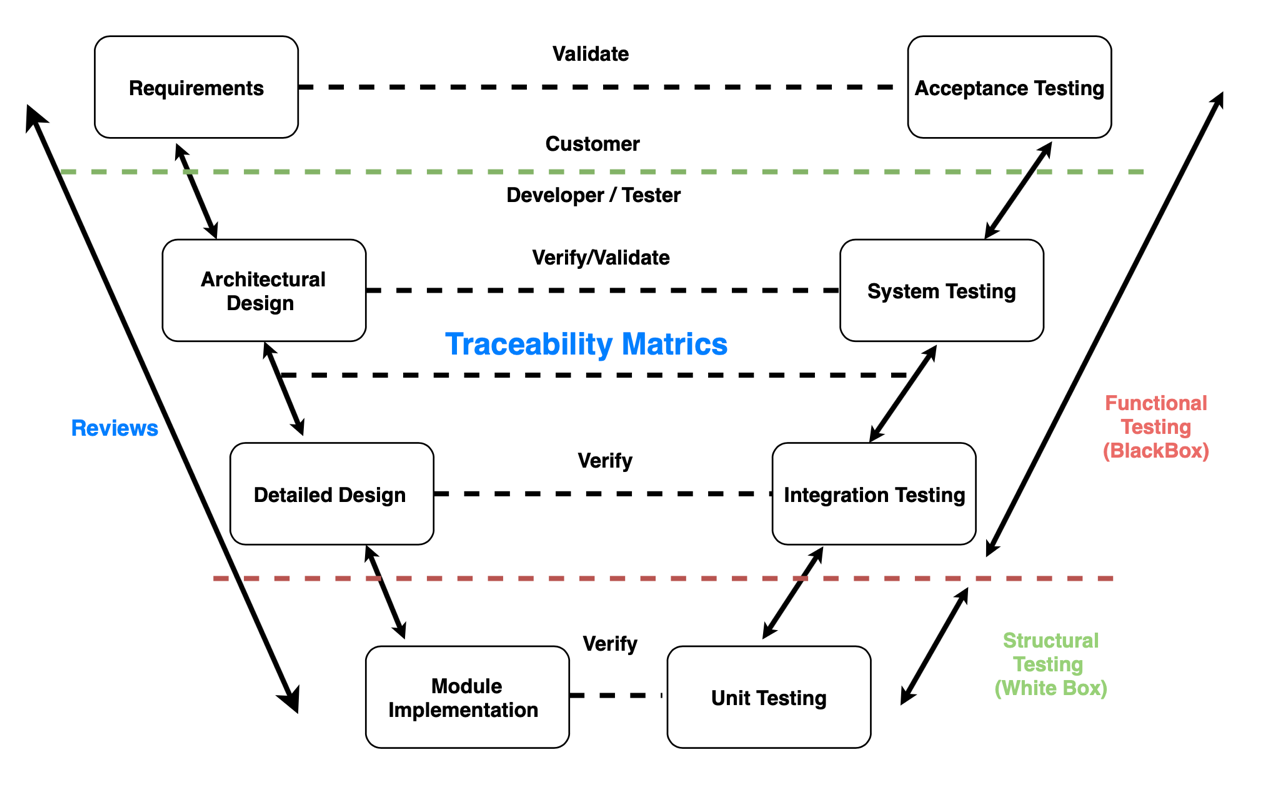
|  |  |  |
| --- | --- | --- |
| **NO.** | **Software used** | **Description** |
| 01 | Operating System | The project will be performed on both the OS i.e. MacOS and Windows for best support and user-friendliness |
| 02 | Anaconda Software | Used to install the various framework for the system i.e. IPython, JupyterNotebook, Pycharm |
| 03 | Python | Python for programming task to perform i.e. Version 3 recommended |
| 04 | Java/JDK | java version "1.8.0\_201" i.e Java8 Java(TM) SE Runtime Environment (build 1.8.0\_201-b09)  Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode) |

# **4**. **Testing**

In this section, we will focus in detail of testing strategies which we have mentioned in our SRS as it is the main stage of our methodology in the waterfall model on which our model is developed. As it is a backbone of our system, so we have focused more on this stage to monitor every stage of the development to detect the difference between the existing and required conditions (that is bugs in the system). ***Testing is performed at every stage of the development phases*** as we have seen in the methodology of our system. We have followed the V&V model for our system.

**4.1** **Verification and Validation Testing**

Our whole system is based on V&V model by this we are focusing how our system is evaluated and how we matched with “Traceability Metrics” with the Development Team Vs Testing Team in this case we all are performing both roles.



**Figure ?:** Verification and Validation Model Reference [ ]

As from the [ ] above Figure. ? which we are giving in our first stage of the project i.e. SRS we stick on this model and answer the following questions:

1. **Did we build the system (Idiomatic Detection System) right (Verification)?**
2. **Did we build the system (Idiomatic Detection System) right (Validation)?**

**Traceability Matrix:** As you can see in the above figure traceability matrix play an important bridge between the Development Team and Testing Team. It’s main role it to trace the requirement specification which we have constructed in the SRS by this we focus more on the functional requirement of the system which we have archived, and which leads to test conditions which we have archived.

In this our traceability matrix we have successfully traced all the user specification with the desired out, i.e. Idiomatic Detection System as our end goal.

**4.1.1 Static Techniques**

In this section, we will see all the static techniques we have used and followed as per our Software requirement specification requirements.

The main Static technique which we used throughout every stage of SDLC is **Review** which saved a lot of time and money and helps us to detect the defects and errors at the early stages of the development phase.

**Review Techniques for our system are as follows:**

1. **Inspection Review:** This technique is the main part of the review technique which we have used throughout our system development which is led by our project Supervisor (**Dr.Aline).**Following are the inspection steps which our supervisor performed at a regular interval of time as shown below:

* Supervisor at a regular interval of time checked the software quality assurance of the project and informed us how to archive the desired quality of the system.
* She also found some problems which we are facing while archiving our system goals with execution.
* She has given us some rules and checklist with some entry and exit criteria to be met for our system.

1. **Walkthrough Review:** In our development stage our group members of the project done a regular scenario check with the dry running of the code so that we archived the desired processes for our project.
2. **Technical Review:** This review technique is led by our Group Members as peer reviewers and technical expert-led by our Supervisor by this we found and detected some fault in the process which we are following to archive our desired output, i.e. *Idiomatic detection system*.

**4.1.2 Dynamic Techniques**

In this technique, we have followed two main methods Black box testing for functional testing and White box testing for Structural testing which helped us to find *fault*, *errors*, *defect* and *failure* in our system and overcome from this. We will see these two methods in more detail below:

**4.1.2.1** **Black Box Testing**

In, this we will monitor the functionality of the system which is ***under test,*** without proper knowledge of the functions for designing and implementation of the system.

The various Black Box testing we performed on our system are as follows:

|  |  |  |
| --- | --- | --- |
| **Input Data** | **Black Box** | **Expected Result** |
| **Features Data** | **Word2vec** | **Dense Vector** |
| **Features Data** | **Skip-thought** | **Dense Vector** |
| **Features Data** | **One-hot encoder** | **Distance Vector** |
| **Test and Train Data** | **Classifiers (Support vector machine , Logistic Regressions, Naïve Bayes)** | **Accuracy and Classification Report (Precision , Recall, F1-score)** |
| **CSV files, Input Array, Test Data, Train Data,** | **Python Libraries (Pandas, Numpy, Scikit-learn)** | **Processed Data for the next stages** |
| **Corpus data** | **NLTK Libraries(Beautiful Soup)** | **Extract feature from XML tag** |

**Figure ? :** Black Box Testing

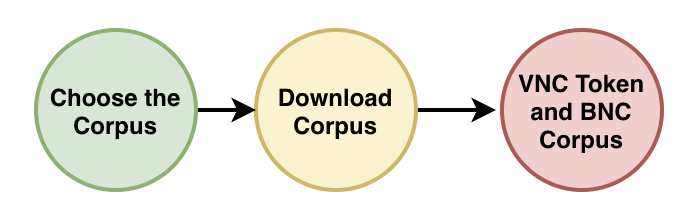
Above Figure shows our Black Box Testing Approach which we have used for testing our test cases from our software requirement specifications with the help of this Functional testing method we have designed our functionality of the unit testing, and after that, we designed the system under test.

The following are the Black Box Techniques used in our project:

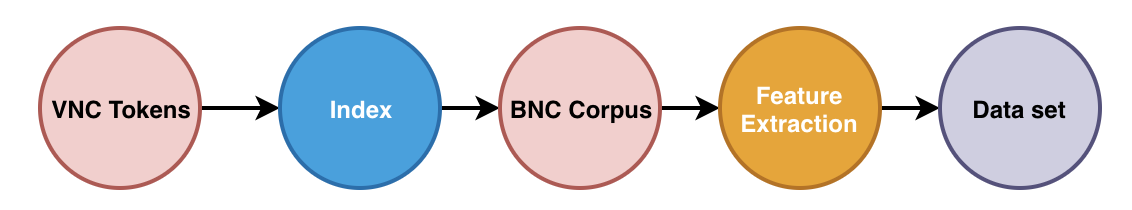
* **State Transition Testing**

In this STT we will apply the BBT to check the functionality based on one state to another state transition diagram or table to show the action to a specific state.

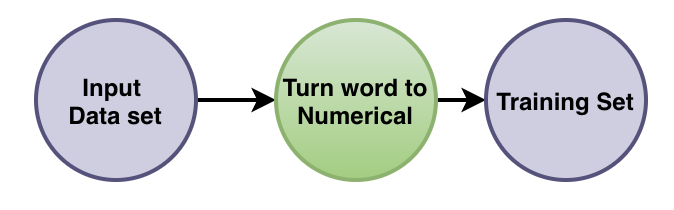
**STT applied to our project:**

****

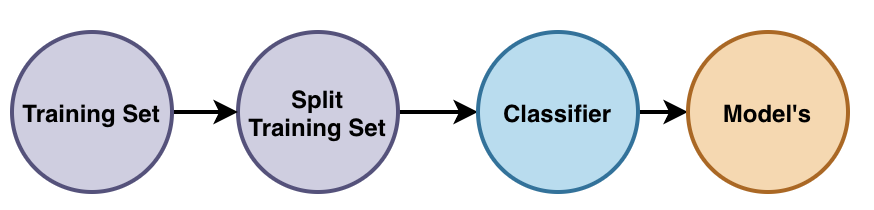
**Figure :** STT for Corpus

****

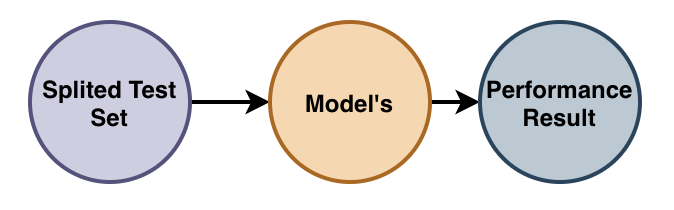
**Figure :** STT for Data set

****

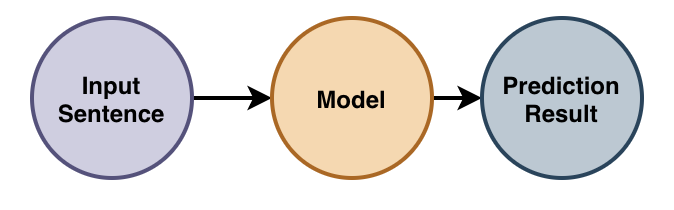
**Figure :** STT for Training Set



**Figure :** STT for Split Training Set



**Figure :** STT for Performance Result

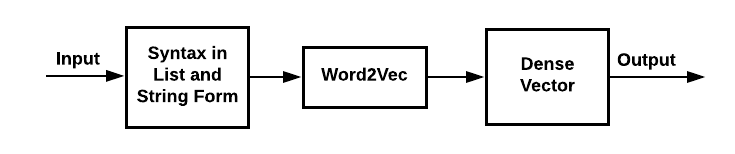
****

**Figure :** STT for Prediction Result

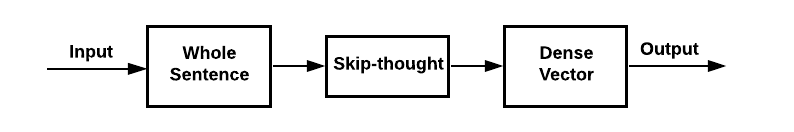
* **Syntax Testing**

In this we applied Syntax Testing , to check that our input data is examined to ensure that the syntax structure of the input is matched with the syntax structure required for the processing of the data we have for development of our system because of large number of test cases the syntax testing is often used as an automated system.

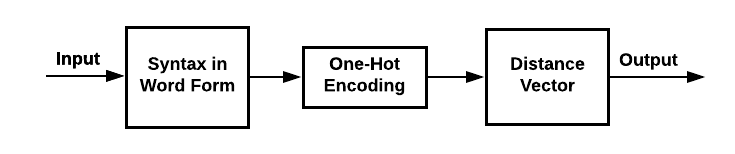
**Below is the following syntax testing which we have performed in our project:**



**Figure :** Syntax Testing for Word2Vec



**Figure :** Syntax Testing for Skip-thought



**Figure :** Syntax Testing for On-hot Encoding

A screenshot of a cell phone

Description automatically generated

**Figure :** Syntax Testing for Classifier (SVM, LR, and NB)

**4.1.2.2 White Box Testing**

As we mention in SRS [ ], it is used for structural testing to design test cases based on the complete knowledge of the structure, detailed knowledge of the design of the system and detailed knowledge of code and system under test. In this, we also need to consider the data set specification, model and use cases, etc to check the desired result of test inputs and expected behavior of the result.

The following are the White Box Techniques used in our project:

* **Statement Testing**

In statement testing, in this, we have tested the line of codes, i.e. the statements or functions which we have used for execution under certain conditions and what the desired outputs we are getting after applying this testing. In this, we also check the path between the various statements and the calling functions for the execution of code to archived the actual results.

|  |  |  |
| --- | --- | --- |
| **Module** | **Line of Code** | **Actual Result** |
| Pre-Processing | 350 LOC | Data set |
| Word2vec | 5 LOC | Dense vector |
| Skip-though | 3 LOC | Dense vector |
| One-Hot Encoding | 3 LOC | Distance vector |
| Support Vector Machine | 7 LOC | Machine Learning Model |
| Logistic Regression | 8 LOC | Machine Learning Model |
| Naïve Bayes | 9 LOC | Machine Learning Model |
| Prediction | 16 LOC | Classification (Literal / Idiomatic ) |

* **Data Flow Testing**

In Data Flow testing, we test the path of the system ( Functions, Model, Statement) and we traced all the states of data objects are accessed and managed throughout a program. We thoroughly check every path of the code with the desired output which we required for our next function of code as we will see below:

**Note:**

**For the Better understanding of the Data Flow Testing, we have used a Test Case as our Data Flow Model to test the detailed flow of paths of the system under test.**

## 4.2 Testing Strategy

### 4.2.1 Unit Testing

### 4.2.2 Integration Testing

### 4.2.3 Acceptance Testing

## 4.3 Test Results

# 5. Conclusions

## 5.1 About the System

## 5.2 About the Methodologies and Tools Used

## 5.3 About Project Management

The project was managed by following a waterfall development methodology given that all of the expected requirements and features were clearly defined since the beginning of the project. This allowed us to create a specific work plan to follow with set milestones and work dates which should, in essence, be considered fixed. Overall, we were able to stay in the schedule with some slack times present in some of the tasks which would later be used in other tasks.

Managing the project was particularly challenging around deadline periods where most of the team members were considerably busier and were therefore unable to work as much time on the project as may have been expected. These deficiencies were made up for in later dates in order to complete the project by the expected end date.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Milestone | Description | Start Date | Expected End Date | Real End Date | Deliverable |
| Requirement elicitation and system design | Defining the main characteristics of the system to be developed by using requirement elicitation techniques as well as system design tools | 17/1/2019 | 7/2/2019 | 7/2/2019 | Complete SRS document |
| Data pre-processing complete | Create a python script to pre-process relevant data and extract features that will be used to train the machine learning model | 7/2/2019 | 14/2/2019 | 12/2/2019 | Python script for data-preprocessing |
| Word2Vec module | Create a python script that implements the Word2Vec library to transform text into vector | 14/2/2019 | 17/2/2019 | 17/2/2019 | Python script using Word2Vec module |
| Skip-thoughts module | Create a python script implementing a skip-thoughts technique for text processing | 17/2/2019 | 23/2/2019 | 24/2/2019 | Python script using a skip-thoughts technique |
| One-hot encoder | Create a python script implementing a one-hot encoder for text processing | 23/2/2019 | 28/2/2019 | 4/3/2019 | Python script implementing a one-hot encoder |
| Model training module | Create a python script to train a machine learning model based on the data obtained from the previously processed text | 4/3/2019 | 15/3/2019 | 12/3/2019 | Python script that generates a trained machine learning model |
| Evaluated machine learning model | Execute a full system to obtain precision, recall, accuracy, and F1-Score on the testing dataset to compare results with the ones stated in the relevant paper | 15/3/2019 | 16/3/2019 | 16/3/2019 | Full system execution has desired performance |
| Final Report | Write the final report describing the design and implementation of the system | 16/3/2019 | 20/3/2019 | 20/3/2019 | Complete final report |

# 6. References