Comparison of various base and ensemble predictors in recognition textual entailment task using sentence extraction

CSE4022

NLP Review 3

Slot: - G2 + TG2

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Team Details :

Team ID- 5

Team mate 1: Mihir Agarwal

Reg No: 18BCE2526

Team mate 2: Shashank Shukla

Reg No. 18BCE2522

Link to code:

https://drive.google.com/file/d/1aaCL_XqDB3K0zBnxIEW03OYICTYF7dyK/view?usp=sharing

Prof. RAJESHKANNAN R

1- Abstract

Recognizing textual entailment (RTE) is a task that predicts whether a text fragment can be inferred from another text fragment. In this project, we tackle the RTE problem using sentence extraction to cover semantic variation and then extracting subject, predicate, and object from each sentence without using external resources like Wordnet. Finally, a similarity function is used to predict entailment relation. In the sentence extraction phase, we used sentence detection, extract sentence in the subordinate clause, prepositional phrase, and passive sentence. Our system has the potential to give accuracy which is comparable to other systems that are not using external resources.

Dataset used is the Third Pascal Recognizing Textual Entailment Challenge (RTE-3) dataset. It has approximately 800 pairs of text (T) and hypothesis (H) with labels as True or False showing whether T entails H or not.

For sentence extraction, the following methodology is followed. The first part is preprocessing, where the parse tree is generated using the Stanford NLP library. After this step part of the speech and parse tree is used for sentence extraction. Then part of the sentence is extracted i.e., subject, predicate, and object using part of speech tag and syntactic parse tree. At the last feature, extraction is done and a classifier is used to predict entailment. In this process, TF-IDF is used for word weighing and a feature table is formed. After feature extraction, any classification algorithm can be used to classify whether the text and hypothesis are entailed or not.

2- Introduction

Recognizing Textual Entailment (RTE) has become an important natural language processing task in recent years. The RTE goal is to detect entailment relation between two snippet text pairs <T (text), H (hypothesis)>. T entails H, if H can be inferred from T using common knowledge.

The following is an example extracted from the first RTE challenge dataset showing Text (T) entails Hypothesis (H).

> T: The body of Satomi Mitarai was found by a teacher after her attacker returned to class in bloody clothes.
> H: Mitarai's body was found by a teacher after her killer returned to their classroom covered in blood.

RTE has been useful in various natural language processing applications to handle variations of semantic expressions, such as information extraction (IE), text summarization, question answering (QA), and machine translation (MT). In-text summarization, textual entailment (TE) can be used to remove sentence redundancy. RTE also can be employed in automatic information credibility assessment tasks. Information is assumed to be more credible if there are other independent sources confirming it and the information is consistent with ground truth information. RTE then can be used to compare information from various sources to check whether it confirms with each other and is consistent with ground truth. Another important factor in information credibility is to assess the sources of information or informant credibility.

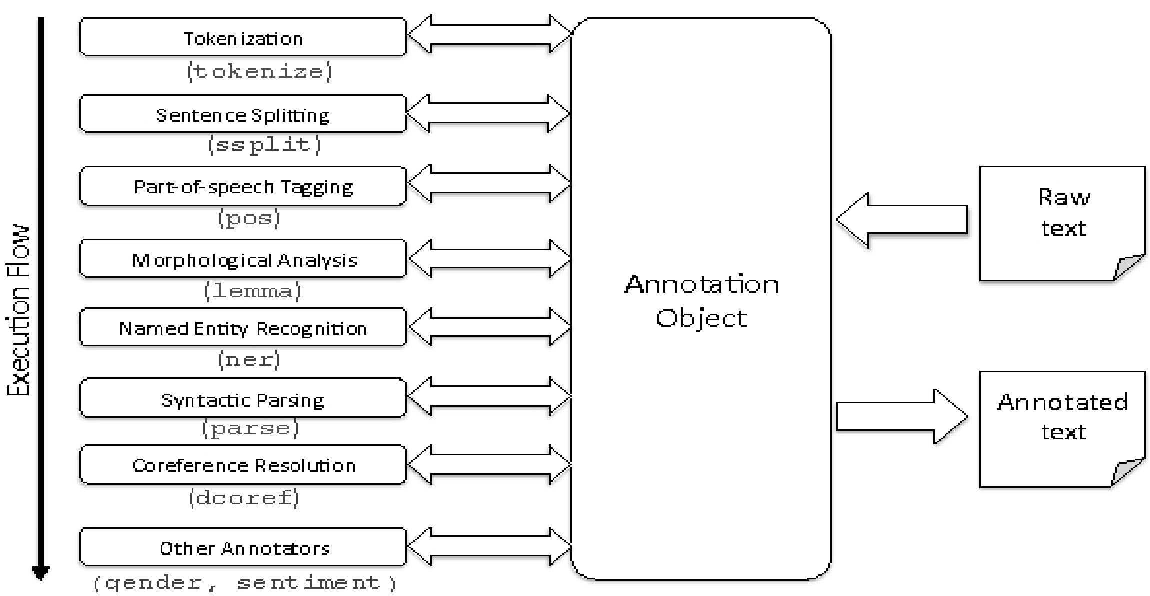
There are three groups of information in the statement map: Focus, Conflict, and Evidence. Focus is a group of information that is related to the query. Conflict contains information that contradicts with Focus. Both Focus and Conflict groups have Evidence groups that contain information to support each group.

Some approaches have been employed to recognize textual entailment automatically, such as 1) lexical similarity and syntactic alignment; 2) logic-based, and 3) combination techniques. Some systems use external lexical databases such as Wordnet, DIRT, Wikipedia, and verb-oriented external databases like VerbNet, FrameNet. Although external resources could increase accuracy, it needs more processing power and time.

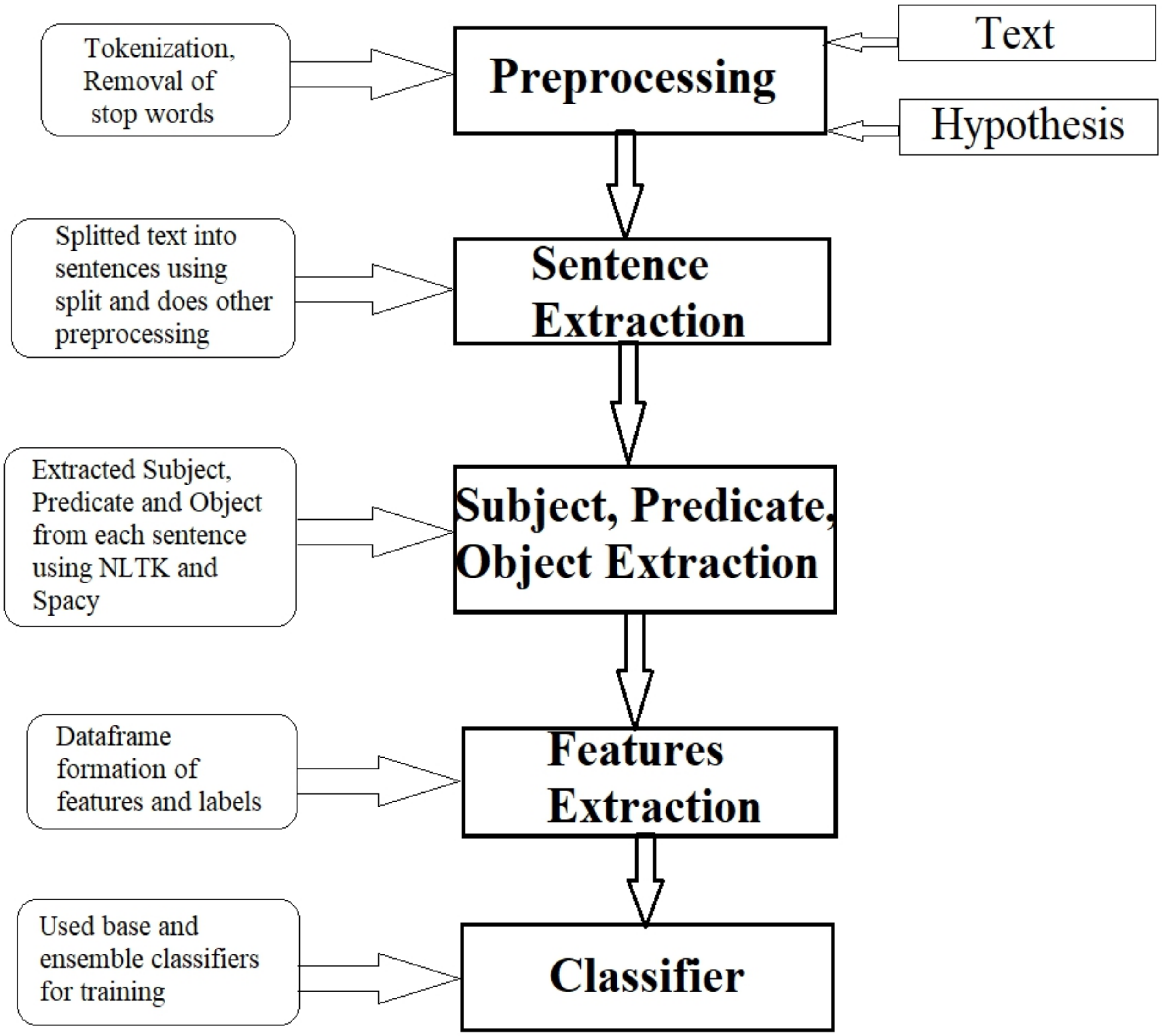
3- Problem Statement

Recognising text entailment has various applications. Nowadays, detection of plagiarism is a common problem. Many students copy answers and answer sheets are highly plagarised. So, textual entailment can be used to see if two texts are related i.e, either one text entails hypothesis or other text. Their are a lot of reviews in amazon and we dont know which are positive and which are negative. So, textual entailment can be used as sentiment analysis by observing if comment entails a positive sentence or not. Many small texts, for example small answers in interviews can be checked at the moment by seing if that answer is entailed by actual answer.

3.1 Architecture Diagram



3.2 Flow Diagram



3.3- Approach

The textual entailment recognizer has five modules, as shown in the above architecture diagram. First, the preprocessing module applies Stanford Parser and transforms T and H into the syntactical structure in the form of a parse tree. Second, sentence extraction applied to H and T that may have more than one sentence or clause. Third, subject, predicate, and object extracted from a sentence. Fourth, features are extracted from the sentence, subject, predicate, and object, and then a classifier used to determine whether H entails T.

In this section, we will discuss each module in more detail.

A. Preprocessing

In the preprocessing step, 3 things are done-

1. Removing symbols like "(",")". '-"
   a) List of h and t are traversed using for loop
   b) .replace function of python is used to replace characters like ',', '-' with whitespace
2. Generation of the syntactic parse tree
   a) The syntactic parse tree is generated using Stanford parser
   b) The parser is loaded in GUI version of it and English is chosen as the language
   c) Text file having required text is loaded and a tree is generated for each text.
3. Generation of part of speech
   a) POS(part of speech) tag is generated using stanza of Stanford NLP library.
   b) It is also extracted from a parse tree generated using the above step.
   c) In the parse tree, each word has its own tag.

B. Sentence Extraction

Sentence extraction is needed because hypothesis (H) can entail with only some part of the text (T), for example:

> T: “At the same time the Italian digital rights group, Electronic Frontiers Italy, has asked the **nation's government to investigate Sony** over its use of anti-piracy software.”
> H: “Italy's government investigates Sony.”

Part-of-speech tag and parse tree are used in the sentence extraction module. Steps in the sentence extraction for T and H are:

1. Sentence detection, split regular sentence which is separated by a period.
2. Extract subordinate clauses. “SBAR” part of speech tag used to extract subordinate clauses.
3. Extract sentences in prepositional phrases. Detected using “S” or “VP” tag which exists in “PP”

After sentence extraction steps, the number of sentences in T or H will be increased.

C. Part of Sentence Extraction: Subject, Predicate, and Object

We extracted subject, predicate, and object for every sentence generated in the sentence extraction module. To extract those parts of a sentence, we used NLP libraries such as NLTK and Spacy to make the dependency graph and took references from Stack Overflow.

D. Feature Extraction and Classifier

Cosine similarity is used for calculating the distance between parts of a sentence (subject, predicate, object) from all H's sentences to all T's sentences.

After all, features extracted from the dataset, we used and compared all machine learning base and ensemble predictors to automatically classify whether T entails H.

3.4- Pseudocode and Code-

```
# -*- coding: utf-8 -*-
"""NLP Review 3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1aaCL_XqDB3K0zBnxIEW03OYICTYF7dyK
"""


"""# Importing Required libraries"""

from bs4 import BeautifulSoup

"""# Mounting Google Drive"""

from google.colab import drive
drive.mount('/content/drive')

"""# Reading Dataset


"""

with open(r'/content/drive/MyDrive/NLP_Review_3/rte1_dev.xml', 'r') as f:
   data = f.read()

data

"""# Passing the stored data inside
# the beautifulsoup parser, storing
# the returned object
"""

Bs_data = BeautifulSoup(data, "xml")
print(Bs_data)

"""## Finding all instances of tag"""

t_array_all = Bs_data.find_all('t')
h_array_all = Bs_data.find_all('h')

t_array = []
h_array = []
for a in t_array_all:
    s = str(a)
    s = s[3:-4]
    s = s.replace(',', '')
    s = s.replace('-', '')
    if(s[-1]!='.'):
        s = s + "."
    s = s + " "
    t_array.append(s)
    print(s)
```

```
#     print(a)

for b in h_array_all:
    s = str(b)
    s = s[3:-4]
    s = s.replace(',', '')
    s = s.replace('-', '')
    if(s[-1]!='.'):
        s = s + "."
    s = s + " "
    h_array.append(s)
    print(s)
    print(b)

# print(h_arrray[1])

"""## Without PreProcessing"""

t_array_all[10]

"""## With PreProcessing"""

t_array[90]

print(h_array[90])

pair_array = Bs_data.find_all('pair')
pair_array[0]

"""# Printing metadata"""

id_array = []
value_array = []
for i in pair_array:
    id = i.get('id')
    value = i.get('value')
    id_array.append(id)
    value_array.append(value)
    print(id)
    print(value)

print(id_array)
print(value_array)

print(len(id_array))
print(len(value_array))
print(len(t_array))
print(len(h_array))

print(id_array[11])
print(value_array[11])
print(t_array[11])
print(h_array[11])

"""# Sentance Extraction"""

l = set(['Mr.', 'Mrs.', 'Ms.', 'Dr.', 'Prof.', 'Rev.', 'Capt.', 'Lt.-Col.',
```

```
'Col.', 'Lt.-Cmdr.', 'The Hon.', 'Cmdr.', 'Flt. Lt.', 'Brgdr.', 'Wng. Cmdr.',
'Group Capt.' ,'Rt.', 'Maj.-Gen.', 'Rear Admrl.', 'Esq.', 'Mx', 'Adv', 'Jr.',

'A.','B.','C.','D.','E.','F.','G.','H.','I.','J.','K.','L.','M.','N.',
         'O.','P.','Q.','R.','S.','T.','U.','V.','W.','X.','Y.','Z.','U.S.'] )
s = 'Iraq became a sovereign country Monday morning but only a dozen
officials were present at the transfer of power ceremony in the heavily
guarded Green Zone of Baghdad. Bremer handed over legal documents that were
accepted by Iraq\'s interim Prime Minister Iyad Allawi. '

def split_at_period(input_str, keywords):
    final = []
    split_l = input_str.split(' ')

    for word in split_l:
        if '.' in word and word not in keywords:
            final.append(word + '\n')
            continue
        final.append(word + ' ')


    return ''.join(final).rstrip()

split_at_period(s, l)

"""# Processing for Text"""

TbeforeSplit = []
for i in t_array:
    TbeforeSplit.append(split_at_period(i,l))

print(TbeforeSplit[51])

TafterSplit = []
for i in TbeforeSplit:
    TafterSplit.append(i.split("\n"))#storing all splitted sentances

t_array[51]

TafterSplit[51]

"""# Extracting Subject, Predicate and Object from the Sentance"""

!pwd

# Commented out IPython magic to ensure Python compatibility.
# %cd "/content/drive/MyDrive/NLP_Review_3"

from subject_verb_object_extract import findSVOs, nlp
TSOP = []#array to fill S,O and p
for FullSentance in TafterSplit:
    SOPFullSentance = []
    for splittedSentance in FullSentance:
        SOPSplittedSentance = []
        tokens = nlp(splittedSentance)
        svos = findSVOs(tokens)
#         print(svos)
```

```
        S=[]
        O=[]
        P=[]
        for eachSOP in svos:
#             print(eachSOP)
            for eachSubject in eachSOP[0].split(","):
                if(len(eachSubject)>0):
                    S.append(eachSubject)
            if(len(eachSOP)>1):
                for eachPredicate in eachSOP[1].split(","):
                    if(len(eachPredicate)>0):
                        P.append(eachPredicate)
            if(len(eachSOP)>2):
                for eachObject in eachSOP[2].split(","):
                    if(len(eachObject)>0):
                        O.append(eachObject)
        SOPSplittedSentance.append(S)
        SOPSplittedSentance.append(P)
        SOPSplittedSentance.append(O)
        SOPFullSentance.append(SOPSplittedSentance)
    TSOP.append(SOPFullSentance)
#         print(SOPFullSentance)
#         print(O)
#         print(P)

print(TSOP)

t_array[10]

TSOP[10]

for i in TSOP:
  print(i)

"""# Processing for Hypothesis

"""

HbeforeSplit = []
for i in h_array:
    HbeforeSplit.append(split_at_period(i,1))

print(HbeforeSplit[51])

h_array[0]

HafterSplit = []
for i in HbeforeSplit:
    HafterSplit.append(i.split("\n"))#storing all splitted sentances

HafterSplit[0]

from subject_verb_object_extract import findSVOs, nlp
HSOP = []#array to fill S,O and p
for FullSentance in HafterSplit:
    SOPFullSentance = []
```

```
    for splittedSentance in FullSentance:
        SOPSplittedSentance = []
        tokens = nlp(splittedSentance)
        svos = findSVOs(tokens)
#         print(svos)

        S=[]
        O=[]
        P=[]
        for eachSOP in svos:
#             print(eachSOP)
            for eachSubject in eachSOP[0].split(","):
                if(len(eachSubject)>0):
                    S.append(eachSubject)
            if(len(eachSOP)>1):
                for eachPredicate in eachSOP[1].split(","):
                    if(len(eachPredicate)>0):
                        P.append(eachPredicate)
            if(len(eachSOP)>2):
                for eachObject in eachSOP[2].split(","):
                    if(len(eachObject)>0):
                        O.append(eachObject)
        SOPSplittedSentance.append(S)
        SOPSplittedSentance.append(P)
        SOPSplittedSentance.append(O)
        SOPFullSentance.append(SOPSplittedSentance)
    HSOP.append(SOPFullSentance)
#         print(SOPFullSentance)
#         print(O)
#         print(P)

print(HSOP)

"""# Calculating Cosine Similarity"""

print(len(t_array))
print(len(h_array))

def cosineSimilarity(t,h):
    X =t
    Y =h

    # tokenization
    X_list = word_tokenize(X)
    Y_list = word_tokenize(Y)

    # sw contains the list of stopwords
    sw = stopwords.words('english')
    l1 =[];l2 =[]

    # remove stop words from the string
    X_set = {w for w in X_list if not w in sw}
    Y_set = {w for w in Y_list if not w in sw}

    # form a set containing keywords of both strings
    rvector = X_set.union(Y_set)
    for w in rvector:
```

```
        if w in X_set: l1.append(1) # create a vector
        else: l1.append(0)
        if w in Y_set: l2.append(1)
        else: l2.append(0)
    c = 0

    # cosine formula
    for i in range(len(rvector)):
            c+= l1[i]*l2[i]
    if(float((sum(l1)*sum(l2))**0.5) ==0):
        return 0
    cosine = c / float((sum(l1)*sum(l2))**0.5)
    return cosine

"""# Cosine Similarity between T and H directly"""

import nltk
nltk.download('punkt')

t_h = []
for i in range (len(t_array)):
    t_h.append(cosineSimilarity(t_array[i],h_array[i]))

print(t_h)

"""# Best Cosine Similarity without Parts of Sentance"""

best_t_h =[]
for indexOfFullSentance in range(len(TafterSplit)):
    best=-1
    for TindexOfSplittedSentance in
range(len(TafterSplit[indexOfFullSentance])):
        for HindexOfSplittedSentance in
range(len(HafterSplit[indexOfFullSentance])):
            a =
cosineSimilarity(TafterSplit[indexOfFullSentance][TindexOfSplittedSentance],H
afterSplit[indexOfFullSentance][HindexOfSplittedSentance])
            if(best<a):
                best =a
    best_t_h.append(best)

TafterSplit[0][0]

best_t_h[0]

for i in best_t_h:
    print(i)
#     print("\n")

print(len(best_t_h))
print(len(t_h))

"""# Best Cosine Similarity with Parts of Sentance"""

best_t_h_spo = []
for indexOfFullSentance in range(len(TSOP)):
    best=-1
```

```
    for TindexOfSplittedSentance in range(len(TSOP[indexOfFullSentance])):
        for HindexOfSplittedSentance in range(len(HSOP[indexOfFullSentance])):
            for i in range(3):

if(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])==0 or
HSOP[indexOfFullSentance][HindexOfSplittedSentance][i]==0):
                    a = 0
                    best=0
                else:
                    for j in
range(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])):
                        for k in
range(len(HSOP[indexOfFullSentance][HindexOfSplittedSentance][i])):
                            a =
cosineSimilarity(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i][j],HS
OP[indexOfFullSentance][HindexOfSplittedSentance][i][k])
                            if(best<a):
                                best =a
    if(best==-1):
        best=0
    best_t_h_spo.append(best)

best_t_h_spo

"""# Average Cosine Similarity with Parts of Sentance"""

avg_t_h_spo = []
for indexOfFullSentance in range(len(TSOP)):
    sum1=0
    count=0
    for TindexOfSplittedSentance in range(len(TSOP[indexOfFullSentance])):
        for HindexOfSplittedSentance in range(len(HSOP[indexOfFullSentance])):
            for i in range(3):


if(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])==0 or
HSOP[indexOfFullSentance][HindexOfSplittedSentance][i]==0):
                    a = 0
                else:
                    for j in
range(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])):
                        for k in
range(len(HSOP[indexOfFullSentance][HindexOfSplittedSentance][i])):
                            a =
cosineSimilarity(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i][j],HS
OP[indexOfFullSentance][HindexOfSplittedSentance][i][k])
                            sum1 =sum1+a
                            count=count+1

    if(count==0):
        best=0
    else:
        best=sum1/count
    avg_t_h_spo.append(best)

avg_t_h_spo
```

```python
"""# Model Training

## Creating Labels array
"""

labels = []
for i in pair_array:
    if (i.get('value')=="TRUE"):
        labels.append(1)
    else:
        labels.append(0)
#     print(i.get('value'))

labels

"""## Creating Dataframe"""

import pandas as pd
df = pd.DataFrame(list(zip(t_h, best_t_h, best_t_h_spo, avg_t_h_spo,labels)),
               columns =['t_h',
'best_t_h','best_t_h_spo','avg_t_h_spo','labels'])
df

df.shape

Xdf = df[['t_h', 'best_t_h','best_t_h_spo','avg_t_h_spo']]
Xdf

X = Xdf.values
y = df.labels.values
print(X)
print(y)

"""
## Splitting X and y into training and testing sets"""

X = Xdf.values
y = df.labels.values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)

xtrain = X_train
ytrain = y_train
xtest=X_test
ytest = y_test

"""# BASE PREDICTIORS"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score,
roc_auc_score
from sklearn.metrics import plot_roc_curve, plot_precision_recall_curve
```

```python
from sklearn.metrics import classification_report,confusion_matrix
from sklearn import metrics

"""# 1- SVM"""

from sklearn.svm import SVC

svm_model=SVC()
svm_model.fit(xtrain,ytrain)

predsvm=svm_model.predict(xtest)
svm_model.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predsvm)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predsvm, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predsvm))
print("\n")
print("   P    N")
print(confusion_matrix(ytest,predsvm))
print("\n")
print(classification_report(ytest,predsvm))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(svm_model, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(svm_model, xtest, ytest)
plt.show()

"""# 2- KNN"""

from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=11)

knn.fit(xtrain,ytrain)

predknn=knn.predict(xtest)
knn.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predknn)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]
```

```python
print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predknn, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predknn))
print("\n")
print(confusion_matrix(ytest,predknn))
print("\n")
print(classification_report(ytest,predknn))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(knn, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(knn, xtest, ytest)
plt.show()

# try K=1 through K=25 and record testing accuracy
k_range = range(1, 15)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 26
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xtrain, ytrain)
    y_pred = knn.predict(xtest)
    scores.append(metrics.accuracy_score(ytest, y_pred))

print(scores)

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')

"""# 3- NAIVE BAYES"""

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

gnb.fit(xtrain,ytrain)

predg=gnb.predict(xtest)
gnb.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predg)
TP=accuracy[0][0]
```

```python
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predg, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predg))
print("\n")
print(confusion_matrix(ytest,predg))
print("\n")
print(classification_report(ytest,predg))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(gnb, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(gnb, xtest, ytest)
plt.show()

c=0
l=len(ytest)
for i in range(0,l):
    if(predg[i]!=ytest[i]):
        c=c+1
print("Number of mislabeled points out of a total %d points : %d" %(l,c))

"""# 4- LOGISTIC REGRESSION"""

from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()

logmodel.fit(xtrain,ytrain)

predlog=logmodel.predict(xtest)
logistic_score=logmodel.score(xtest,ytest)*100
logistic_score

accuracy=confusion_matrix(ytest,predlog)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predlog, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predlog))
```

```
print("\n")
print(confusion_matrix(ytest,predlog))
print("\n")
print(classification_report(ytest,predlog))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(logmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(logmodel, xtest, ytest)
plt.show()

"""# 5- MLP"""

from sklearn.neural_network import MLPClassifier

model=MLPClassifier(hidden_layer_sizes=(20,20),max_iter=2000)
model.fit(xtrain,ytrain)

predn=model.predict(xtest)
model.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predn)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predn, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predn))
print("\n")
print(confusion_matrix(ytest,predn))
print("\n")
print(classification_report(ytest,predn))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(model, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(model, xtest, ytest)
plt.show()

"""# 6- DECISION TREE"""

from sklearn import tree
```

```
tmodel=tree.DecisionTreeClassifier()
tmodel.fit(xtrain,ytrain)

predt=tmodel.predict(xtest)
tmodel.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predt)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predt, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predt))
print("\n")
print(confusion_matrix(ytest,predt))
print("\n")
print(classification_report(ytest,predt))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(tmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(tmodel, xtest, ytest)
plt.show()

"""# ENSEMBLE PREDICTORS

# 1- ADABOOST
"""

from sklearn.ensemble import AdaBoostClassifier

adamodel = AdaBoostClassifier(n_estimators=100)
adamodel.fit(xtrain,ytrain)

predada=adamodel.predict(xtest)
adamodel.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predada)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
```

```
print("\n")
print("F1-score or FM: ", f1_score(ytest, predada, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predada))
print("\n")
print(confusion_matrix(ytest,predada))
print("\n")
print(classification_report(ytest,predada))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(adamodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(adamodel, xtest, ytest)
plt.show()

"""# 2- BAGGING"""

from sklearn.ensemble import BaggingClassifier

bagmodel = BaggingClassifier(base_estimator=None, n_estimators=10)
#default=decision tree, SVC()
bagmodel.fit(xtrain, ytrain)

predbag=bagmodel.predict(xtest)
bagmodel.score(xtest, ytest)*100

accuracy=confusion_matrix(ytest,predbag)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predbag, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predbag))
print("\n")
print(confusion_matrix(ytest,predbag))
print("\n")
print(classification_report(ytest,predbag))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(bagmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(bagmodel, xtest, ytest)
plt.show()
```

```
"""# 3- Extra_Tree_Classifier"""

from sklearn.ensemble import ExtraTreesClassifier

exmodel = ExtraTreesClassifier(n_estimators=100)
exmodel.fit(xtrain, ytrain)

predex=exmodel.predict(xtest)
exmodel.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predex)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predex, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predex))
print("\n")
print(confusion_matrix(ytest,predex))
print("\n")
print(classification_report(ytest,predex))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(exmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(exmodel, xtest, ytest)
plt.show()

"""# 4- Gradient_Boosting_Classifier"""

from sklearn.ensemble import GradientBoostingClassifier

gradmodel = GradientBoostingClassifier()
gradmodel.fit(xtrain,ytrain)

predgrad=gradmodel.predict(xtest)
gradmodel.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predgrad)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
```

```
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predgrad, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predgrad))
print("\n")
print(confusion_matrix(ytest,predgrad))
print("\n")
print(classification_report(ytest,predgrad))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(gradmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(gradmodel, xtest, ytest)
plt.show()

"""# 5- Random_Forest_Classifier"""

from sklearn.ensemble import RandomForestClassifier

randmodel = RandomForestClassifier()
randmodel.fit(xtrain,ytrain)

predrand=randmodel.predict(xtest)
randmodel.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predrand)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predrand, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predrand))
print("\n")
print(confusion_matrix(ytest,predrand))
print("\n")
print(classification_report(ytest,predrand))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(randmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(randmodel, xtest, ytest)
plt.show()
```

```python
"""# 6- Stacking_Classifier"""

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier

estimators = [('rf', RandomForestClassifier(n_estimators=10,
random_state=42)),
              ('svr', make_pipeline(StandardScaler(),
LinearSVC(random_state=42)))]

stmodel = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())
stmodel.fit(xtrain,ytrain)

predst=stmodel.predict(xtest)
stmodel.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predst)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predst, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predst))
print("\n")
print(confusion_matrix(ytest,predst))
print("\n")
print(classification_report(ytest,predst))

x=np.arange(100)*0.01
y=x

disp = plot_roc_curve(stmodel, xtest, ytest)
plt.plot(x,y, '--')
plt.show()

disp = plot_precision_recall_curve(stmodel, xtest, ytest)
plt.show()

"""# 7- Voting_Classifier"""

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier

clf1 = LogisticRegression()
clf2 = RandomForestClassifier()#n_estimators=50, random_state=1)
```

```
clf3 = GaussianNB()
votmodel = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting='hard')
votmodel.fit(xtrain,ytrain)

predvot=votmodel.predict(xtest)
votmodel.score(xtest,ytest)*100

accuracy=confusion_matrix(ytest,predvot)
TP=accuracy[0][0]
FP=accuracy[1][0]
TN=accuracy[1][1]
FN=accuracy[0][1]

print("Accuracy: ",(TP+TN)/(TP+FP+TN+FN)*100)
print("Probability of detection of defect(Recall, pd): ",TN/(TN+FP))
print("Probability of false alarm(pf): ",FP/(TP+FP))
print("Probability of correct detection(Precision): ", TN/(TN+FN))
print("\n")
print("F1-score or FM: ", f1_score(ytest, predvot, average='binary'))
print("AUC value: ",roc_auc_score(ytest, predvot))
print("\n")
print(confusion_matrix(ytest,predvot))
print("\n")
print(classification_report(ytest,predvot))




"""# For given input text"""

l = set(['Mr.', 'Mrs.', 'Ms.', 'Dr.', 'Prof.', 'Rev.', 'Capt.', 'Lt.-Col.', 'Col.', 'Lt.-Cmdr.', 'The Hon.', 'Cmdr.', 'Flt. Lt.', 'Brgdr.', 'Wng. Cmdr.', 'Group Capt.' ,'Rt.', 'Maj.-Gen.', 'Rear Admrl.', 'Esq.', 'Mx', 'Adv', 'Jr.',

'A.','B.','C.','D.','E.','F.','G.','H.','I.','J.','K.','L.','M.','N.',
         'O.','P.','Q.','R.','S.','T.','U.','V.','W.','X.','Y.','Z.','U.S.'] )
s = 'Iraq became a sovereign country Monday morning but only a dozen officials were present at the transfer of power ceremony in the heavily guarded Green Zone of Baghdad. Bremer handed over legal documents that were accepted by Iraq\'s interim Prime Minister Iyad Allawi. '

split_at_period(s, l)

text='In Seoul Han confirmed that all North Korean media on Monday had started referring to Kim Jong Il as "His Excellency" or by the term previously reserved for his father "the Great Leader." Until now the younger Kim has been called "the Dear Leader. '
hypothesis= 'Kim Il Sung\'s son is called Kim Jong Il'

TtestSplit=[]
TtestSplit.append(split_at_period(text, l).split("\n"))
print(TtestSplit)

HtestSplit=[]
HtestSplit.append(split_at_period(hypothesis, l).split("\n"))
```

```
print(HtestSplit)

from subject_verb_object_extract import findSVOs, nlp
TSOPtest = []#array to fill S,O and p
for FullSentance in TtestSplit:
    SOPFullSentance = []
    for splittedSentance in FullSentance:
        SOPSplittedSentance = []
        tokens = nlp(splittedSentance)
        svos = findSVOs(tokens)
#         print(svos)

        S=[]
        O=[]
        P=[]
        for eachSOP in svos:
#             print(eachSOP)
            for eachSubject in eachSOP[0].split(","):
                if(len(eachSubject)>0):
                    S.append(eachSubject)
            if(len(eachSOP)>1):
                for eachPredicate in eachSOP[1].split(","):
                    if(len(eachPredicate)>0):
                        P.append(eachPredicate)
            if(len(eachSOP)>2):
                for eachObject in eachSOP[2].split(","):
                    if(len(eachObject)>0):
                        O.append(eachObject)
        SOPSplittedSentance.append(S)
        SOPSplittedSentance.append(P)
        SOPSplittedSentance.append(O)
        SOPFullSentance.append(SOPSplittedSentance)
    TSOPtest.append(SOPFullSentance)
#         print(SOPFullSentance)
#         print(O)
#         print(P)

print(TSOPtest)

from subject_verb_object_extract import findSVOs, nlp
HSOPtest = []#array to fill S,O and p
for FullSentance in HtestSplit:
    SOPFullSentance = []
    for splittedSentance in FullSentance:
        SOPSplittedSentance = []
        tokens = nlp(splittedSentance)
        svos = findSVOs(tokens)
#         print(svos)

        S=[]
        O=[]
        P=[]
        for eachSOP in svos:
#             print(eachSOP)
            for eachSubject in eachSOP[0].split(","):
                if(len(eachSubject)>0):
                    S.append(eachSubject)
```

```
            if(len(eachSOP)>1):
                for eachPredicate in eachSOP[1].split(","):
                    if(len(eachPredicate)>0):
                        P.append(eachPredicate)
            if(len(eachSOP)>2):
                for eachObject in eachSOP[2].split(","):
                    if(len(eachObject)>0):
                        O.append(eachObject)
        SOPSplittedSentance.append(S)
        SOPSplittedSentance.append(P)
        SOPSplittedSentance.append(O)
        SOPFullSentance.append(SOPSplittedSentance)
    HSOPtest.append(SOPFullSentance)
#          print(SOPFullSentance)
#          print(O)
#          print(P)

print(HSOPtest)

t_array=['In Seoul Han confirmed that all North Korean media on Monday had 
started referring to Kim Jong Il as "His Excellency" or by the term 
previously reserved for his father "the Great Leader." Until now the younger 
Kim has been called "the Dear Leader. ']
h_array=['Kim Il Sung\'s son is called Kim Jong Il']

t_h = []
for i in range (len(t_array)):
    t_h.append(cosineSimilarity(t_array[i],h_array[i]))
print(t_h)

TafterSplit=TtestSplit
HafterSplit=HtestSplit
TSOP=TSOPtest
HSOP=HSOPtest

best_t_h =[]
for indexOfFullSentance in range(len(TafterSplit)):
    best=-1
    for TindexOfSplittedSentance in 
range(len(TafterSplit[indexOfFullSentance])):
        for HindexOfSplittedSentance in 
range(len(HafterSplit[indexOfFullSentance])):
            a = 
cosineSimilarity(TafterSplit[indexOfFullSentance][TindexOfSplittedSentance],H
afterSplit[indexOfFullSentance][HindexOfSplittedSentance])
            if(best<a):
                best =a
    best_t_h.append(best)
print(best_t_h)

best_t_h_spo = []
for indexOfFullSentance in range(len(TSOP)):
    best=-1
    for TindexOfSplittedSentance in range(len(TSOP[indexOfFullSentance])):
        for HindexOfSplittedSentance in range(len(HSOP[indexOfFullSentance])):
            for i in range(3):
```

```
if(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])==0 or
HSOP[indexOfFullSentance][HindexOfSplittedSentance][i]==0):
                    a = 0
                    best=0
                else:
                    for j in
range(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])):
                        for k in
range(len(HSOP[indexOfFullSentance][HindexOfSplittedSentance][i])):
                            a =
cosineSimilarity(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i][j],HS
OP[indexOfFullSentance][HindexOfSplittedSentance][i][k])
                            if(best<a):
                                best =a
    if(best==-1):
        best=0
    best_t_h_spo.append(best)
print(best_t_h_spo)

avg_t_h_spo = []
for indexOfFullSentance in range(len(TSOP)):
    sum1=0
    count=0
    for TindexOfSplittedSentance in range(len(TSOP[indexOfFullSentance])):
        for HindexOfSplittedSentance in range(len(HSOP[indexOfFullSentance])):
            for i in range(3):


if(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])==0 or
HSOP[indexOfFullSentance][HindexOfSplittedSentance][i]==0):
                    a = 0
                else:
                    for j in
range(len(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i])):
                        for k in
range(len(HSOP[indexOfFullSentance][HindexOfSplittedSentance][i])):
                            a =
cosineSimilarity(TSOP[indexOfFullSentance][TindexOfSplittedSentance][i][j],HS
OP[indexOfFullSentance][HindexOfSplittedSentance][i][k])
                            sum1 =sum1+a
                            count=count+1

    if(count==0):
        best=0
    else:
        best=sum1/count
    avg_t_h_spo.append(best)
print(avg_t_h_spo)

labels=[0]

df = pd.DataFrame(list(zip(t_h, best_t_h, best_t_h_spo, avg_t_h_spo,labels)),
               columns =['t_h',
'best_t_h','best_t_h_spo','avg_t_h_spo','labels'])
df

Xdf = df[['t_h', 'best_t_h','best_t_h_spo','avg_t_h_spo']]
```

```
Xdf

x_test = Xdf.values
y_test = df.labels.values

print(x_test, y_test)

print(svm_model.predict(x_test))
```

4- Experiment and Results

4.1. Dataset (sample with explanation)

We used a dataset from the Third Pascal Recognizing Textual Entailment Challenge (RTE-3). RTE-3 has two datasets: development set and test set. Each consists of 800 pairs of text (T) and hypothesis (H), all manually annotated. RTE-3 dataset is the last RTE challenge dataset available for direct download without needing special permission.

RTE-3 pairs were taken from various sources from the web and were reviewed by three human judges. The average agreement between judges is 87.8% with a Kappa level 0.75.

Given below is a screenshot of one group in the dataset which is in XML format. It has pair id, value as True or False which shows that hypothesis is entailed from text or not, and task as QA which stands for Question Answering. It has two tags as t and h. T is for Text and H is for Hypothesis.

```
<pair id="625" value="FALSE" task="QA">

   <t>This year, however, the contest has taken on a new urgency as the Clinton
Administration, moving to block the Pyongyang government &apos;s bid to build
a nuclear arsenal, has rekindled some of the passion in North Korea&apos;s
defiance of the West.</t>

   <h>Pyongyang is the capital of North Korea.</h>

</pair>
```

4.2. Output

4.2.1- Output of Part A (Preprocessing) -

Removing symbols like “(“,”)”. ‘-”-

Without PreProcessing

```
In [252]: t_array_all[10]
Out[252]: <t>Iraqi militants said Sunday they would behead Kim Sun-Il, a 33-year-old translator, within 24 hours unless plans to dispatch
          thousands of South Korean troops to Iraq were abandoned.</t>
```

With PreProcessing

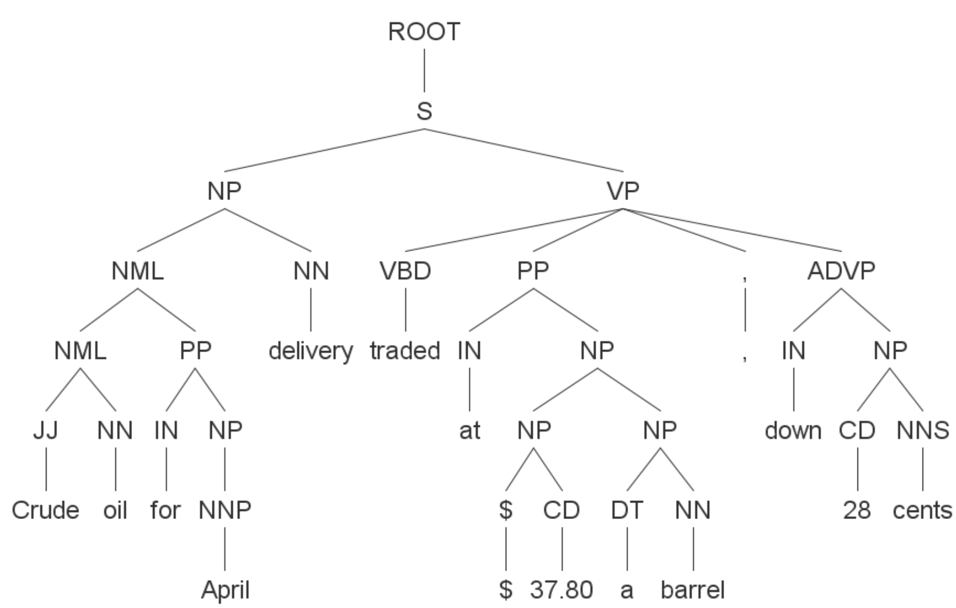
```
In [251]: t_array[10]
Out[251]: 'Iraqi militants said Sunday they would behead Kim SunIl a 33yearold translator within 24 hours unless plans to dispatch thousa
          nds of South Korean troops to Iraq were abandoned. '
```

4.2.2- Output of Part B (Sentence Extraction) -

Generation of the syntactic parse tree

Sentance - Crude oil for April delivery traded at $37.80 a barrel, down 28 cents

Generated Parse tree –



(ROOT (S (NP (NML (NML (JJ Crude) (NN oil)) (PP (IN for) (NP (NNP April)))) (NN delivery)) (VP (VBD traded) (PP (IN at) (NP ($ $) (CD 37.80))) (NP (NP (DT a) (NN barrel)) (ADVP (IN down) (NP (CD 28) (NNS cents))))) (. .)))

Using steps in part B, we extracted the following text into six sentences

"I just hope I don't become so blissful I become boring. Nirvana leader Kurt Cobain giving meaning to his "Teen Spirit" coda a denial. "

Sentence	Step
I just hope I don\'t become so blissful I become boring.	Sentence detection
Nirvana leader Kurt Cobain giving meaning to his "Teen Spirit" coda "a denial	Sentence detection

Screenshot-

```
t_array[51]
'"I just hope I don\'t become so blissful I become boring."  Nirvana leader Kurt Cobain giving meaning to his "Teen Spirit" coda "a denial". '
```

```
[294] TafterSplit[51]
['"I just hope I don\'t become so blissful I become boring."',
 ' Nirvana leader Kurt Cobain giving meaning to his "Teen Spirit" coda "a denial".']
```

4.2.3- Output of Part C (Parts of Sentence Extraction: Subject, Predicate and Object)-

Example- Iraqi militants said Sunday they would behead Kim SunIl a 33yearold translator within 24 hours unless plans to dispatch thousands of South Korean troops to Iraq were abandoned.

Screenshot -

```
[304] t_array[10]
'Iraqi militants said Sunday they would behead Kim SunIl a 33yearold translator within 24 hours unless plans to dispatch thousands of South Korean troops to Iraq were abandoned. '
```

```
[305] TSOP[10]
[[['Iraqi militants',
   'Kim SunIl',
   'a 33yearold translator',
   '24 hours',
   'thousands of Korean troops',
   'Iraq',
   'plans'],
  ['said', 'behead', 'behead', 'behead', 'dispatch', 'dispatch', 'abandoned'],
  ['they', 'they', 'they', 'plans', 'plans']]]
```

```
[301] t_array[11]

'Two Turkish engineers and an Afghan translator kidnapped in December were freed Friday. '
```

```
TSOP[11]

[[['December', 'Two Turkish engineers', 'an Afghan translator'],
  ['kidnap', 'freed', 'freed'],
  ['an Afghan translator']]]
```

Sentence	Subject	Object	Predicate
Iraqi militants said Sunday they would behead Kim SunIl a 33yearold translator within 24 hours unless plans to dispatch thousands of South Korean troops to Iraq were abandoned. of children.	'Iraqi militants', 'Kim SunIl', 'a 33yearold translator', '24 hours', 'thousands of Korean troops', 'Iraq', 'plans'	'they', 'they', 'they', 'plans', 'plans'	'said', 'behead', 'behead', 'behead', 'dispatch', 'dispatch', 'abandoned'
Two Turkish engineers and an Afghan translator kidnapped in December were freed Friday.	'December', 'Two Turkish engineers', 'an Afghan translator'	'kidnap', 'freed', 'freed'	'an Afghan translator'

4.2.4- Output of part D (Feature Extraction and Classifier)-

After sentence extraction, each T and H could be transformed into several sentences. Cosine similarity is used for calculating distance between parts of sentence (subject, predicate, object) from all H's sentences to all T's sentences.

We assumed value of cosine similarity between T and H is proportional with entailment relationship between them. We also use cosine similarity to calculate distance between T and H sentences directly and not using part of sentence.

Table shows complete features.

Feature	Description
best_t_h_spo	Best cosine similarity between all T's and H's sentence using part of sentence(subject-predicate-object)
best_t_h	Best cosine similarity without part of sentence
avg_t_h_spo	Average cosine similarity using part of sentence
t_h	Cosine similarity between H and T (not using sentence extraction)

Screenshot -

	t_h	best_t_h	best_t_h_spo	avg_t_h_spo	labels
0	0.603023	0.617213	0.000000	0.288675	0
1	0.547723	0.547723	0.000000	0.000000	0
2	0.676123	0.676123	0.000000	0.000000	1
3	0.267261	0.267261	1.000000	0.023810	0
4	0.095346	0.095346	0.000000	0.000000	1
...	...	...	...	...	...
1762	0.396059	0.396059	1.000000	0.155192	0
1763	0.606780	0.606780	1.000000	0.325192	0
1764	0.400000	0.400000	1.000000	0.181818	0
1765	0.534522	0.534522	0.408248	0.081650	0
1766	0.462910	0.462910	0.000000	0.000000	0

1767 rows × 5 columns

4.2.5 - Output of Various Ensemble and Base Machine Learning Models

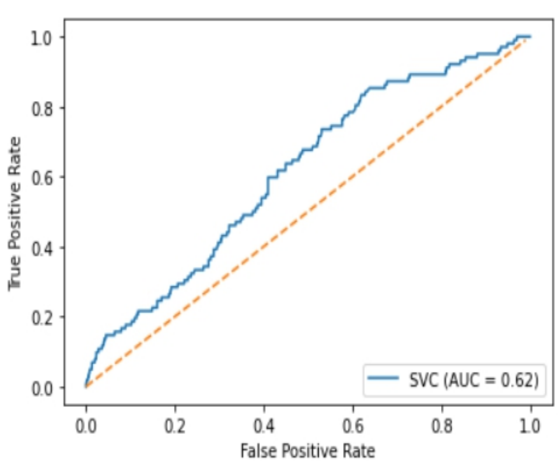
4.2.5.1 - Base models

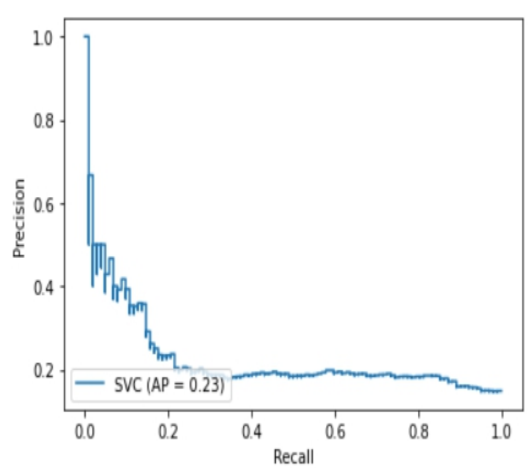
1)SVM –

Confusion Matrix -

```
print(confusion_matrix(ytest,predsvm))
print("\n")
print(classification_report(ytest,predsvm))
```

```
Accuracy:  85.57284299858557
Probability of detection of defect(Recall, pd):  0.0
Probability of false alarm(pf):  0.14427157001414428
Probability of correct detection(Precision):  nan


F1-score or FM:  0.0
AUC value:  0.5


   P    N
[[605   0]
 [102   0]]

              precision    recall  f1-score   support

           0       0.86      1.00      0.92       605
           1       0.00      0.00      0.00       102

    accuracy                           0.86       707
   macro avg       0.43      0.50      0.46       707
weighted avg       0.73      0.86      0.79       707
```



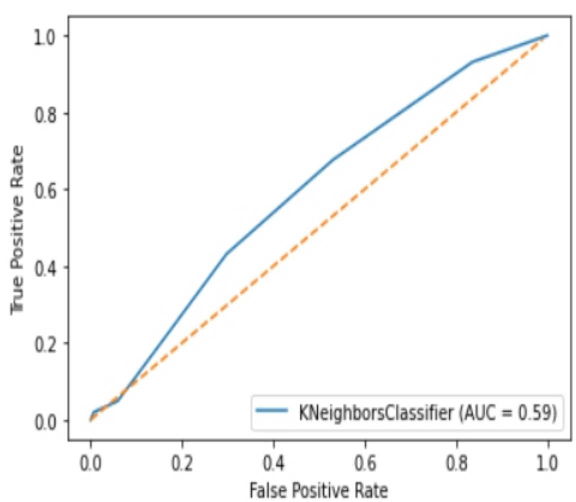


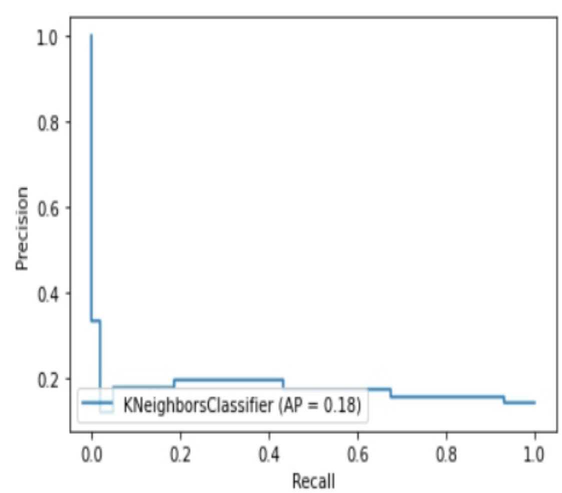
2 – KNN

Confusion Matrix -

```
print(classification_report(ytest,predknn))
```

```
Accuracy:  85.2899575671853
Probability of detection of defect(Recall, pd):  0.0196078431372549
Probability of false alarm(pf):  0.14265335235378032
Probability of correct detection(Precision):  0.3333333333333333


F1-score or FM:  0.037037037037037035
AUC value:  0.5064981364446605


[[601   4]
 [100   2]]

              precision    recall  f1-score   support

           0       0.86      0.99      0.92       605
           1       0.33      0.02      0.04       102

    accuracy                           0.85       707
   macro avg       0.60      0.51      0.48       707
weighted avg       0.78      0.85      0.79       707
```

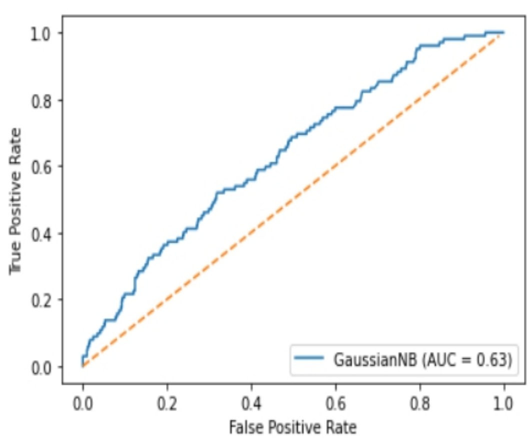


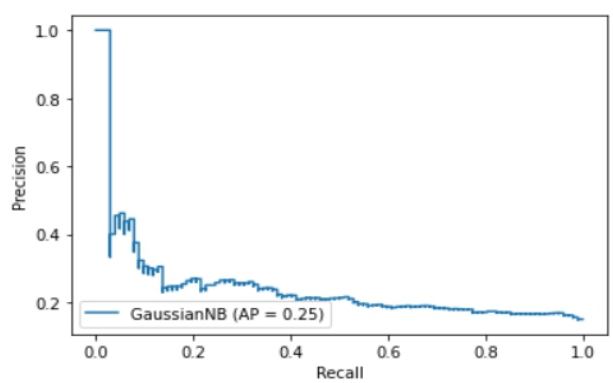


3 -NAIVE BAYES

Confusion Matrix -

```
Accuracy:  82.03677510608203
Probability of detection of defect(Recall, pd):  0.13725490196078433
Probability of false alarm(pf):  0.1345565749235474
Probability of correct detection(Precision):  0.2641509433962264


F1-score or FM:  0.1806451612903226
AUC value:  0.5363960460217144


[[566  39]
 [ 88  14]]


              precision    recall  f1-score   support

           0       0.87      0.94      0.90       605
           1       0.26      0.14      0.18       102

    accuracy                           0.82       707
   macro avg       0.56      0.54      0.54       707
weighted avg       0.78      0.82      0.80       707
```



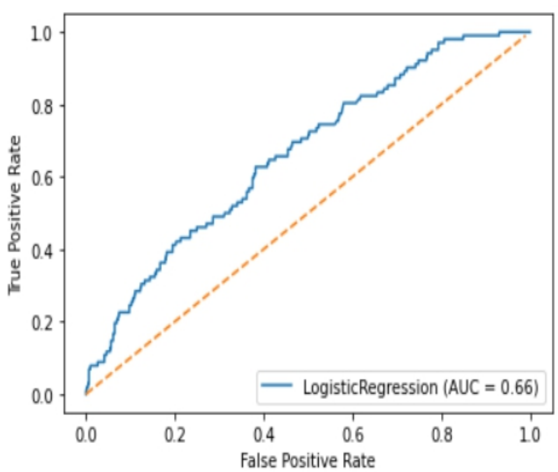


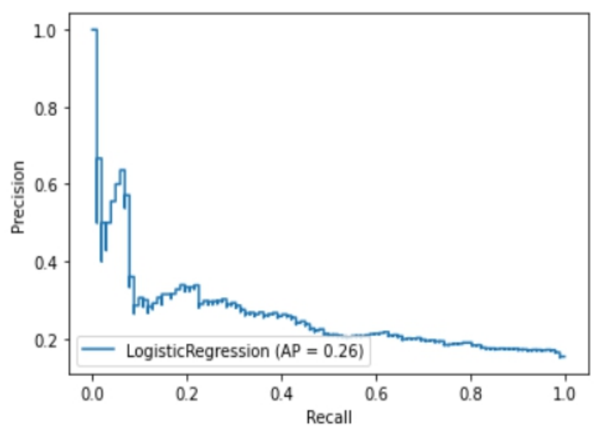
4 -LOGISTIC REGRESSION

Confusion Matrix -

```
print(confusion_matrix(ytest,predlog))
print("\n")
print(classification_report(ytest,predlog))
```

```
Accuracy:  85.57284299858557
Probability of detection of defect(Recall, pd):  0.0
Probability of false alarm(pf):  0.14427157001414428
Probability of correct detection(Precision):  nan


F1-score or FM:  0.0
AUC value:  0.5


[[605   0]
 [102   0]]

              precision    recall  f1-score   support

           0       0.86      1.00      0.92       605
           1       0.00      0.00      0.00       102

    accuracy                           0.86       707
   macro avg       0.43      0.50      0.46       707
weighted avg       0.73      0.86      0.79       707
```



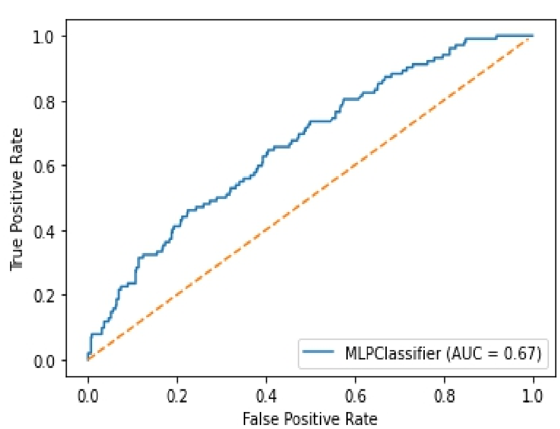


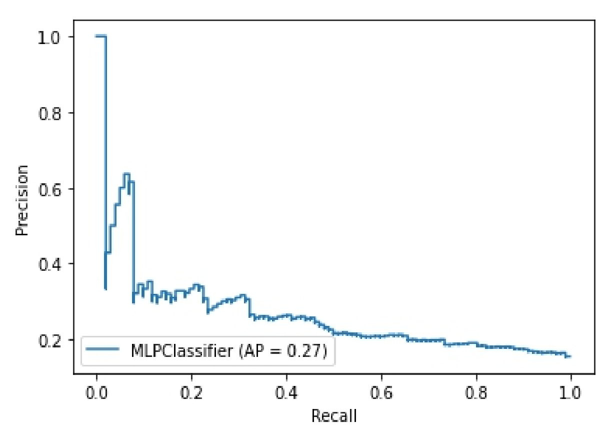
5- MLP

<u>Confusion Matrix -</u>

```
print(confusion_matrix(ytest,predn))
print("\n")
print(classification_report(ytest,predn))
```

```
Accuracy:  85.57284299858557
Probability of detection of defect(Recall, pd):  0.0
Probability of false alarm(pf):  0.14427157001414428
Probability of correct detection(Precision):  nan


F1-score or FM:  0.0
AUC value:  0.5


[[605   0]
 [102   0]]

              precision    recall  f1-score   support

           0       0.86      1.00      0.92       605
           1       0.00      0.00      0.00       102

    accuracy                           0.86       707
   macro avg       0.43      0.50      0.46       707
weighted avg       0.73      0.86      0.79       707
```



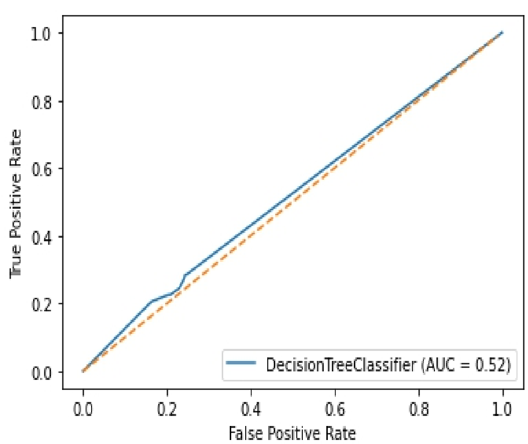


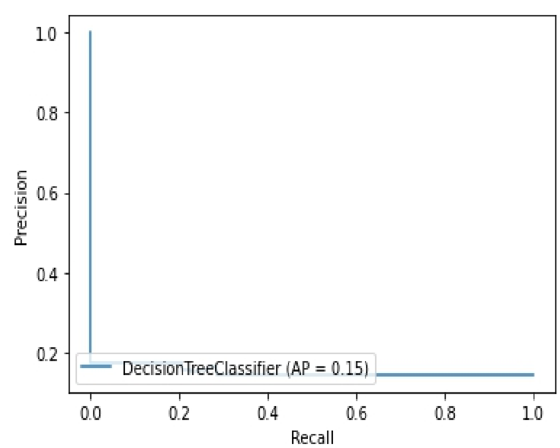
6- DECISION TREE

Confusion Matrix -

```
Accuracy:  74.54031117397454
Probability of detection of defect(Recall, pd):  0.20588235294117646
Probability of false alarm(pf):  0.13798977853492334
Probability of correct detection(Precision):  0.175


F1-score or FM:  0.18918918918918917
AUC value:  0.5211229946524064


[[506  99]
 [ 81  21]]
```

	precision	recall	f1-score	support
0	0.86	0.84	0.85	605
1	0.17	0.21	0.19	102
accuracy			0.75	707
macro avg	0.52	0.52	0.52	707
weighted avg	0.76	0.75	0.75	707



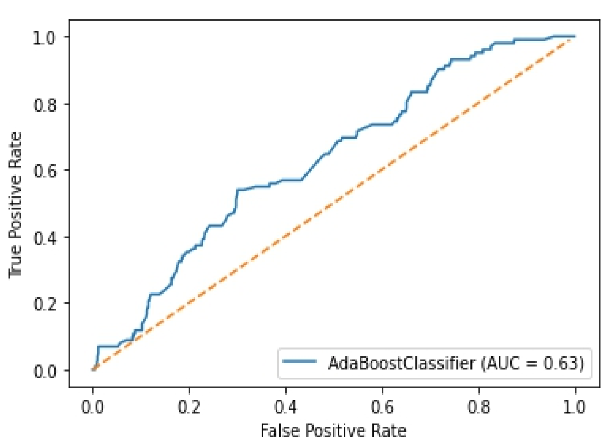


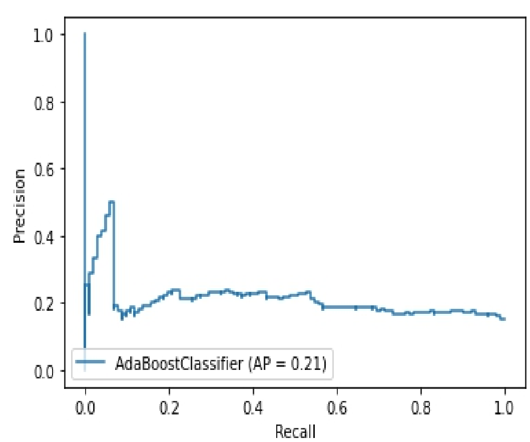
4.2.5.1 – Ensemble Predictors

1- ADABOOST

Confusion Matrix -

```
Accuracy:  85.14851485148515
Probability of detection of defect(Recall, pd):  0.029411764705882353
Probability of false alarm(pf):  0.14183381088825214
Probability of correct detection(Precision):  0.3333333333333333


F1-score or FM:  0.05405405405405406
AUC value:  0.5097472046669907


[[599   6]
 [ 99   3]]


              precision    recall  f1-score   support

           0       0.86      0.99      0.92       605
           1       0.33      0.03      0.05       102

    accuracy                           0.85       707
   macro avg       0.60      0.51      0.49       707
weighted avg       0.78      0.85      0.79       707
```

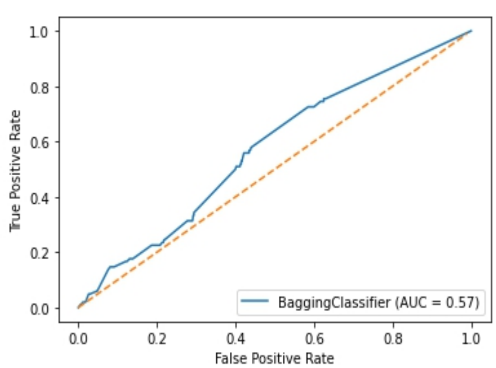


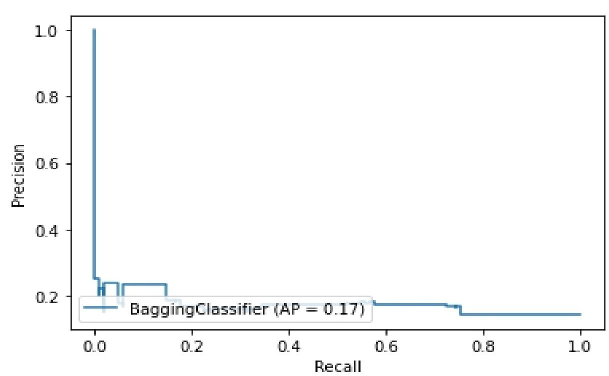


2- BAGGING

Confusion Matrix -

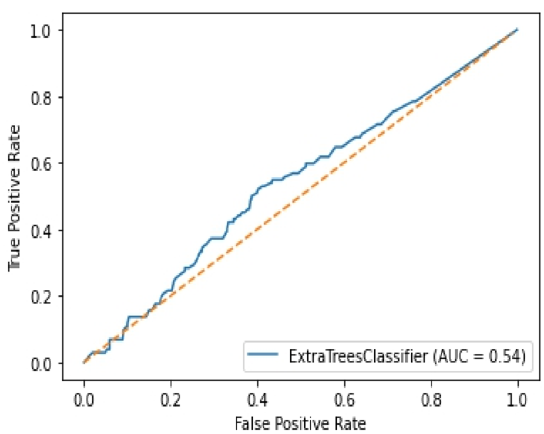
```
Accuracy:  79.9151343705799
Probability of detection of defect(Recall, pd):  0.14705882352941177
Probability of false alarm(pf):  0.13657770800627944
Probability of correct detection(Precision):  0.21428571428571427


F1-score or FM:  0.1744186046511628
AUC value:  0.5280748663101604


[[550  55]
 [ 87  15]]

              precision    recall  f1-score   support

           0       0.86      0.91      0.89       605
           1       0.21      0.15      0.17       102

    accuracy                           0.80       707
   macro avg       0.54      0.53      0.53       707
weighted avg       0.77      0.80      0.78       707
```

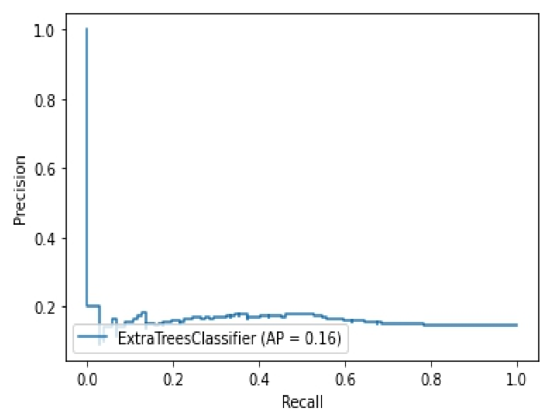




3- Extra_Tree_Classifier

```
Accuracy:  78.07637906647807
Probability of detection of defect(Recall, pd):  0.13725490196078433
Probability of false alarm(pf):  0.14057507987220447
Probability of correct detection(Precision):  0.1728395061728395


F1-score or FM:  0.15300546448087432
AUC value:  0.5132555501539459


[[538  67]
 [ 88  14]]

              precision    recall  f1-score   support

           0       0.86      0.89      0.87       605
           1       0.17      0.14      0.15       102

    accuracy                           0.78       707
   macro avg       0.52      0.51      0.51       707
weighted avg       0.76      0.78      0.77       707
```

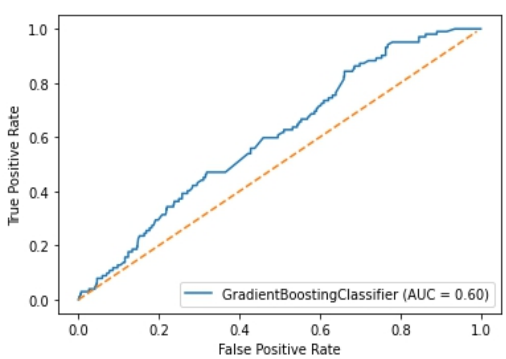


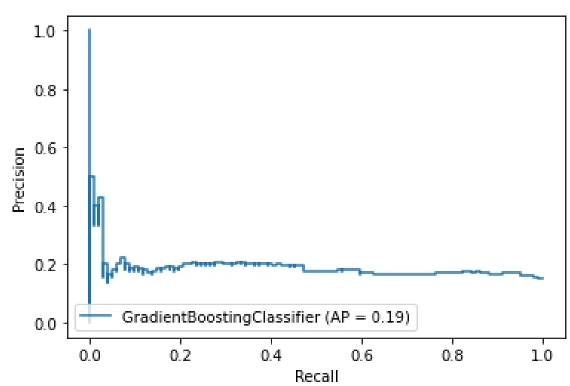


4- Gradient_Boosting_Classifier

Confusion Matrix -

```
Accuracy:  84.86562942008487
Probability of detection of defect(Recall, pd):  0.029411764705882353
Probability of false alarm(pf):  0.14224137931034483
Probability of correct detection(Precision):  0.2727272727272727


F1-score or FM:  0.05309734513274336
AUC value:  0.5080943121050072


[[597   8]
 [ 99   3]]

              precision    recall  f1-score   support

           0       0.86      0.99      0.92       605
           1       0.27      0.03      0.05       102

    accuracy                           0.85       707
   macro avg       0.57      0.51      0.49       707
weighted avg       0.77      0.85      0.79       707
```

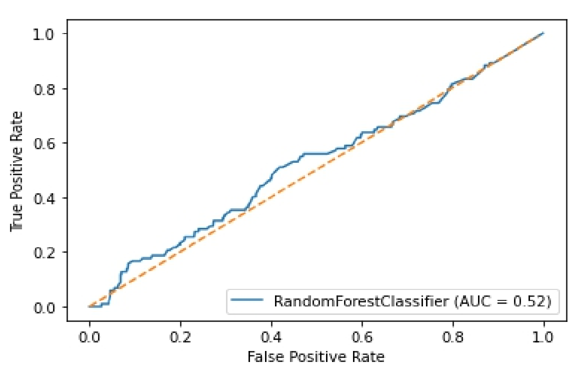


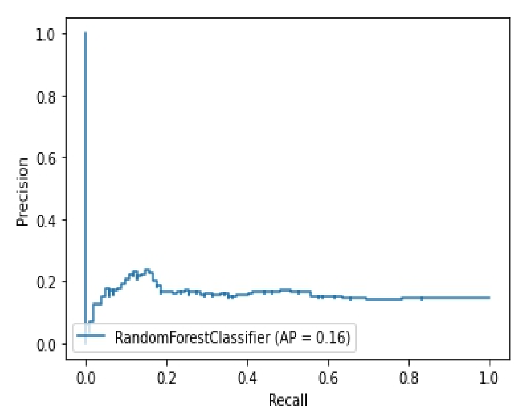


5- Random_Forest_Classifier

Confusion Matrix -

```
Accuracy:  80.33946251768033
Probability of detection of defect(Recall, pd):  0.12745098039215685
Probability of false alarm(pf):  0.13819875776397517
Probability of correct detection(Precision):  0.20634920634920634


F1-score or FM:  0.15757575757575756
AUC value:  0.5224031761464917


[[555  50]
 [ 89  13]]

              precision    recall  f1-score   support

           0       0.86      0.92      0.89       605
           1       0.21      0.13      0.16       102

    accuracy                           0.80       707
   macro avg       0.53      0.52      0.52       707
weighted avg       0.77      0.80      0.78       707
```

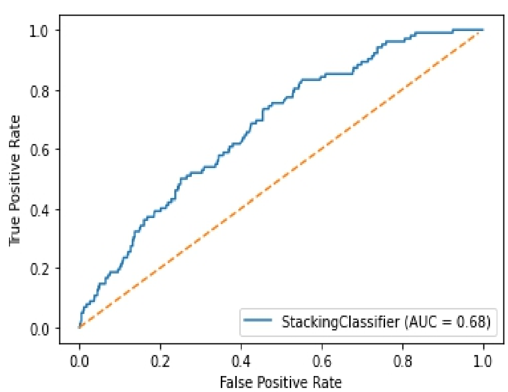


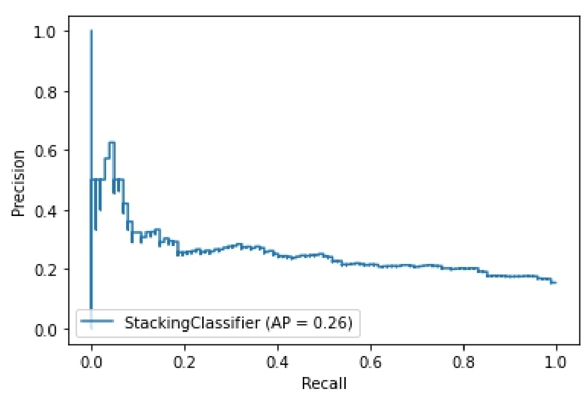


6- Stacking_Classifier

Confusion Matrix -

```
Accuracy:  85.57284299858557
Probability of detection of defect(Recall, pd):  0.0
Probability of false alarm(pf):  0.14427157001414428
Probability of correct detection(Precision):  nan


F1-score or FM:  0.0
AUC value:  0.5


[[605   0]
 [102   0]]


              precision    recall  f1-score   support

           0       0.86      1.00      0.92       605
           1       0.00      0.00      0.00       102

    accuracy                           0.86       707
   macro avg       0.43      0.50      0.46       707
weighted avg       0.73      0.86      0.79       707
```





7- Voting_Classifier

Confusion Matrix -

```
Accuracy:  85.85572842998586
Probability of detection of defect(Recall, pd):  0.0392156862745098
Probability of false alarm(pf):  0.1398002853067047
Probability of correct detection(Precision):  0.6666666666666666


F1-score or FM:  0.07407407407407407
AUC value:  0.5179549505752714


[[603   2]
 [ 98   4]]

              precision    recall  f1-score   support

           0       0.86      1.00      0.92       605
           1       0.67      0.04      0.07       102

    accuracy                           0.86       707
   macro avg       0.76      0.52      0.50       707
weighted avg       0.83      0.86      0.80       707
```

Final Result

S.No.	Algorithm	Accuracy
Base Predictors		
1	SVM	85.57
2	KNN	85.28
3	NAIVE BAYES	82.03
4	LOGISTIC REGRESSION	85.57
5	MLP	85.57
6	Decision Tree	74.54
Ensemble Predictors		
1	AdaBoost	85.14%
2	Bagging	79.91%
3	Extra Tree Classifier	78.07%
4	Gradient Boosting	84.86%
5	Random Forest	80.33%
6	Stacking	85.57%
7	Voting	85.85%

5 - Conclusion

From the above implementation it can be concluded that, to analyze entailment relation between sentences, these steps to be followed are preprocessing, sentence splitting, Extracting Subject, Object and Predicate, Calculating Cosine Similarity and running different models based on the feature extracted are the main steps. In the presented methodology, syntactic parse tree and part of speech of tags are extracted and the sentences are splited, the subject, object and predicate of all the splited sentences are calculated and different models are run using Cosine similarities as features for the given dataset.

In this method, we present our method that employs sentence extraction and part of speech extraction (subject, predicate, object) in recognition textual entailment task. For sentence extraction, we use sentence detection to extract subordinate clause, sentences in prepositional phrases. We have then compared the outputs of various Base and ensemble predictors and it could be seen that it is not necessary that ensemble predictors will give better accuracy, but it can be true in some cases. In our final results we observed that Voting an ensemble algorithm gave best accuracy of 85.87.

Different base algorithms like SVM(Support Vector Machine), KNN(K Nearest Neighbor) etc. are used, total of 6 algorithms. Various ensemble learners like Adaboost, Bagging, Stacking, Voting etc. are used, total of 7 algorithms. All there results are compared in tables, graphs and results of best performing algorithms are shown in required section for each dataset. From the table and graphs it is clearly visible that in all datasets, ensemble learners performs better than base algorithms. For all datasets, FM and AUC values are better for ensemble algorithms. Metrics used for result comparison are FM(F-measure) value, AUC(Area Under ROC curve) value, Recall, Precision value, PF(Probability of False Alarm). All these values for different algorithms are shown in result section. Graph includes ROC(receiver operating characteristic curve) and Precision-Recall curve. A precision recall curve shows the relationship between precision (= positive predictive value= TP / (TP + FP)) and recall(= sensitivity = TP / (TP + FN)). An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification threshold.

6 References

1.Bar-Haim, Roy et al., "Semantic inference at the lexical-syntactic level", PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, vol. 22, no. 1, 2007.

2.Bos, Johan and Katja Markert, "Combining shallow and deep NLP methods for recognizing textual entailment", Proceedings of the First PASCAL Challenges Workshop on Recognising Textual Entailment, 2005.

3.Dan Klein and Christopher D. Manning, "Accurate Unlexicalized Parsing", Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430, 2003.

4.Giampiccolo, Danilo et al., "The third pascal recognizing textual entailment challenge", Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing, 2007.

5.Harabagiu, Sanda and Hickl Andrew, "Methods for using textual entailment in open-domain question answering", Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, 2006.

6.Iftene, Adrian and Balahur-Dobrescu Alexandra, "Hypothesis transformation and semantic variability rules used in recognizing textual entailment", Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. Association for Computational Linguistics, 2007.

7.Lloret, Elena et al., "A Text Summarization Approach under the Influence of Textual Entailment", NLPCS, 2008.

8.Magnini, Bernardo et al., "Entailment Graphs for Text Analytics in the Excitement Project", Text Speech and Dialogue, 2014.

9."Ministry of Defence", Joint Doctrine Publication 2–00 Understanding and Intelligence Support to Join Operations, 2011.

10.Snow, Rion, Lucy Vanderwende and Arul Menezes, "Effectively using syntax for recognizing false entailment", Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, 2006.

11.Tatu, Marta and Dan Moldovan, "Cogex at RTE3", Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. Association for Computational Linguistics, 2007.

12.Murakami, Koji et al., "Statement map: assisting information crediblity analysis by visualizing arguments", Proceedings of the 3rd workshop on Information credibility on the web, 2009.

13.Wang, Rui and Gunter Neumann, "Recognizing textual entailment using sentence similarity based on dependency tree skeletons", Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. Association for Computational Linguistics, 2007.

14.I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 2000.

15.Li, Baoli et al., "Machine learning based semantic inference: Experiments and Observations at RTE-3", Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. Association for Computational Linguistics, 2007.

16.Malakasiotis, Prodromos and Ion Androutsopoulos, "Learning textual entailment using SVMs and string similarity measures", Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. Association for Computational Linguistics, 2007.

17.Marsi, Erwin, Krahmer Emiel and Bosma Wauter, "Dependency-based paraphrasing for recognizing textual entailment", Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. Association for Computational Linguistics, 2007.