Sentiment analysis - Proposed

by By Turnitin

Submission date: 23-Apr-2022 02:55AM (UTC-0400)
Submission ID: 1817981517
File name: Sentiment_analysis_-_Proposed_2.pdf (1.17M)
Word count: 2607
Character count: 15104

Sentiment analysis - Proposed

April 22, 2022

[1]:
```
"""'Sentiment evaluation - This is taken into consideration an emerging subject␣
↪matter these days . This research paper targets to reap a dataset of␣
↪tweetsnand practice extraordinary device studying algorithms to analyze and␣
↪classify texts. This researchnpaper explored textual content classification␣
↪accuracy even as using distinct classifiers for classifyingnbalanced and␣
↪unbalanced datasets.'"""
```

[1]:
```
"'Sentiment evaluation - This is taken into consideration an emerging subject
matter these days . This research paper targets to reap a dataset of tweetsnand
practice extraordinary device studying algorithms to analyze and classify texts.
This researchnpaper explored textual content classification accuracy even as
using distinct classifiers for classifyingnbalanced and unbalanced datasets.'"
```

1 Importing Liberaries

[2]:
```
## Basic Liberaries
import numpy as np
""" For reading CSV """
import pandas as pd
import multiprocessing as mp
import string

## preprocessing liberaries

import nltk                                                    # import␣
↪nltk for natural language processing

## Data visualising liberaries

from matplotlib import pyplot as plt                           # import␣
↪pyplot for visulaising the data
import re
from matplotlib import patches as mpatches
%matplotlib inline
import seaborn as sns                                          # import␣
↪seaborn for data visusalising
```

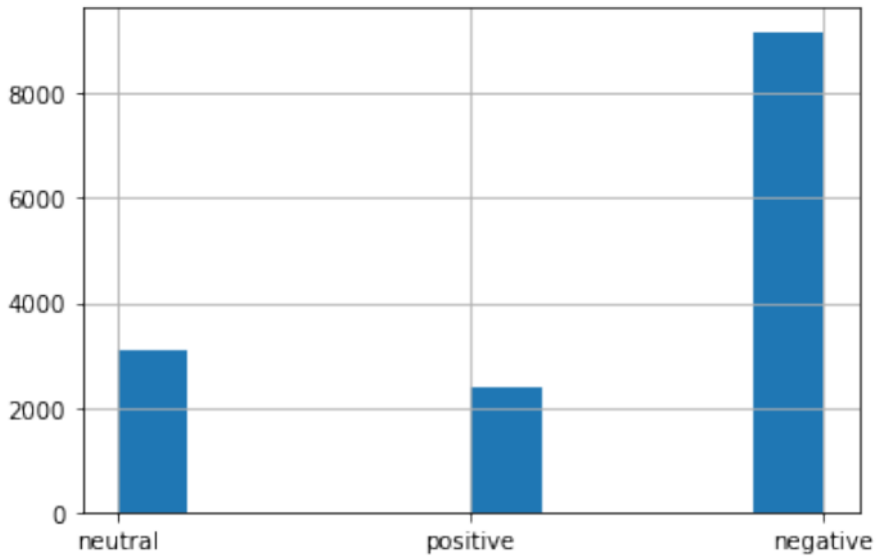
```
## classification liberaries -- Model Importing

from sklearn.ensemble import RandomForestClassifier                    # import
↪Random Forest Classifier from sklearn -- classification model
from sklearn.naive_bayes import GaussianNB                             # import
↪Naive Bayes from sklearn
from sklearn.model_selection import train_test_split                   # for train
↪test split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
↪f1_score, roc_auc_score, confusion_matrix, classification_report
from sklearn.naive_bayes import MultinomialNB                          # Import
↪multinomialNB -- It is a classification model
from sklearn.feature_extraction.text import TfidfVectorizer            # import
↪tfidf vectorisation from sklear FOr Vectorisation Process
from sklearn.linear_model import LogisticRegression                    # Logistic
↪Regression
from sklearn.svm import LinearSVC
from sklearn.preprocessing import OrdinalEncoder                       # For
↪encodeing the text into number
```

2 Data Loading and Visualising

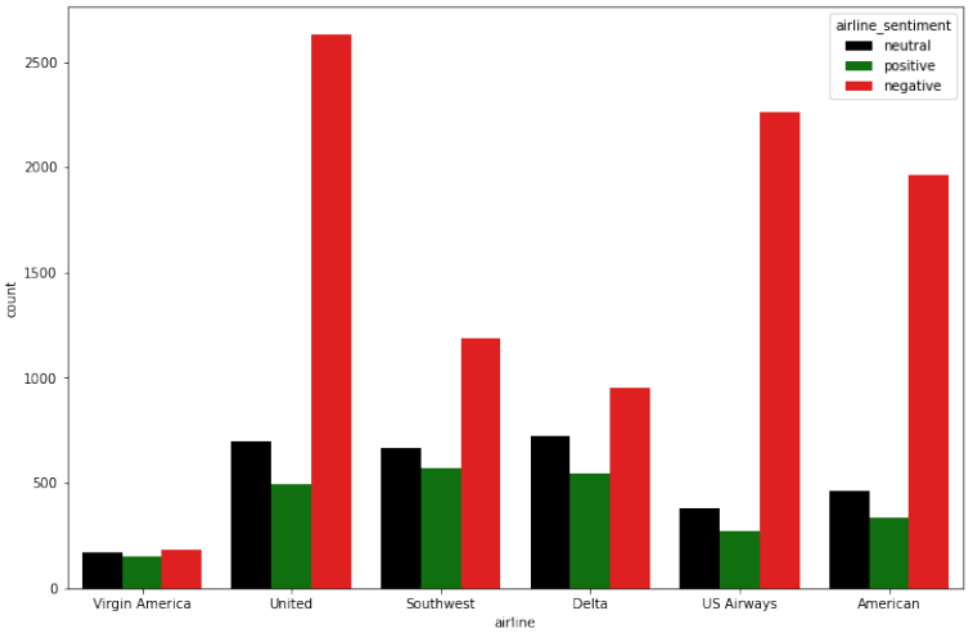
[3]:
```
tweets = pd.read_csv('Tweets.csv') ## Reading the data

dataset = tweets.copy()
```

[4]:
```
dataset['airline_sentiment'].hist() ## Comparing the sentiments of tweets

## Here the number of negative tweets is more than the number of neutral and
↪Positive
```

[4]: <AxesSubplot:>



[5]:
```
# No. of positive , neutral and negative tweets in dataset per airline

plt.rcParams["figure.figsize"] = (12,8)

colors ={"positive": "green" ,"neutral": "Black", "negative": "red"}
ax = sns.countplot(data = dataset, x ="airline", hue = "airline_sentiment",␣
 ↪palette=colors)
```



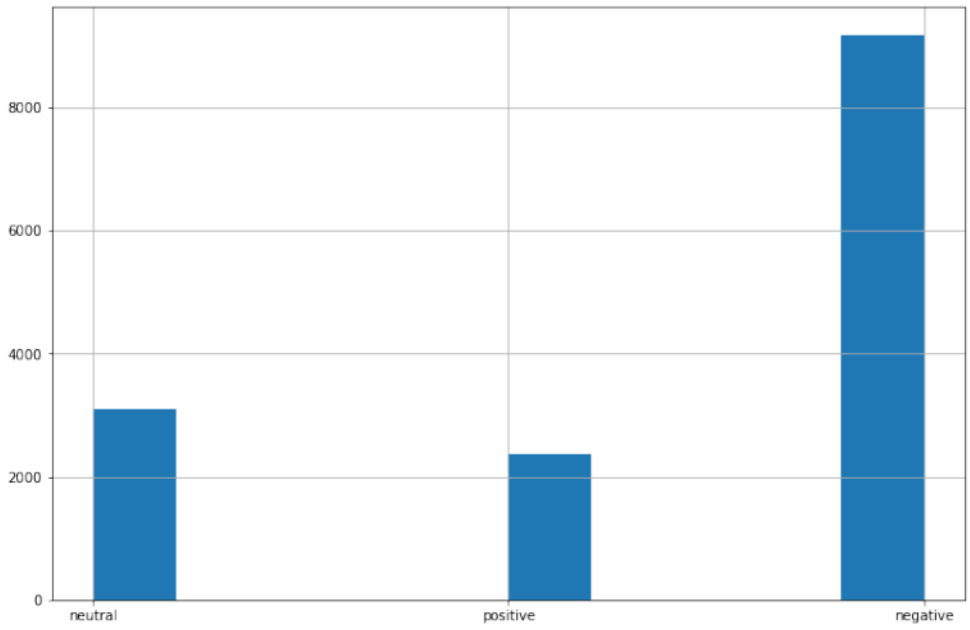
[6]:
```
## Removing the un-used value from Dataframe

dataset.drop(['tweet_location' ,'tweet_id' , 'airline_sentiment_confidence' ,␣
↪'negativereason' , 'user_timezone', 'negativereason_gold' , 'name'  ,␣
↪'tweet_created' , 'tweet_coord' ,'negativereason_gold', 'retweet_count' ,␣
↪'negativereason_confidence',  'airline_sentiment_gold'], axis=1 , inplace =␣
↪True)
```

[7]:
```
dataset.shape
```

[7]: (14640, 3)

[8]:
```
dataset["airline_sentiment"].hist()
```

[8]: <AxesSubplot:>



3 Preprocessing

3.1 Removing Puntuation

[9]:
```
def remove_punctuation(txt):
    ## removing Puntuation as they are not that important for sentimental␣
 ↪analysis
    text_nopun = "".join([chars for chars in txt if chars not in string.
 ↪punctuation])
    text_lower = "".join([chars.lower() for chars in text_nopun])
    return text_lower

dataset['data_no_Puntuation'] = dataset['text'].apply(lambda x:␣
 ↪remove_punctuation(x))
```

3.2 Tokeniseing

[10]:
```
def tokenizing(txt):
    tokens_txt = re.split('\W+' , txt)
    return tokens_txt

dataset['text_tokenised'] = dataset['data_no_Puntuation'].apply(lambda x :␣
 ↪tokenizing(x))
```

3.3 Removing stopword

[11]:
```
stopwords = nltk.corpus.stopwords.words('english')
def remove_stopWord(texts_tokenised):
    text_cleaned = [words for words in texts_tokenised if words not in␣
 ↪stopwords]
    return text_cleaned

dataset['text_no_SW'] = dataset['text_tokenised'].apply(lambda x :␣
 ↪remove_stopWord(x))
```

3.4 Removing Based on length

[12]:
```
def remove_based_on_length(text_no_SW):
    text_lengh_based = [wrd for wrd in text_no_SW if len(wrd) in range(3 , 21)]
    return text_lengh_based

dataset['text_lengh_based'] = dataset['text_no_SW'].apply(lambda x :␣
 ↪remove_based_on_length(x))
```

3.5 Stemming list

[13]:
```
"""Stemming Process for converting words to its base word"""
ps = nltk.PorterStemmer()

def stemming(txt_lengh_based):
    text = [ps.stem(wrd) for wrd in txt_lengh_based]
    return text

dataset['text_stemized'] = dataset['text_lengh_based'].apply(lambda y :␣
 ↪stemming(y))
```

3.6 Removing First word as it is flight name only

[14]:
```
flightNames = ["virginamerica" , "united" , "delta" , "southwestair" ,␣
 ↪"usairways" , "americanair"]
def remove_first_word(text_stemized):
    first_Remove = [words for words in text_stemized if words not in␣
 ↪flightNames]
    return first_Remove

dataset['first_Remove'] = dataset['text_stemized'].apply(lambda z:␣
 ↪remove_first_word(z))
```

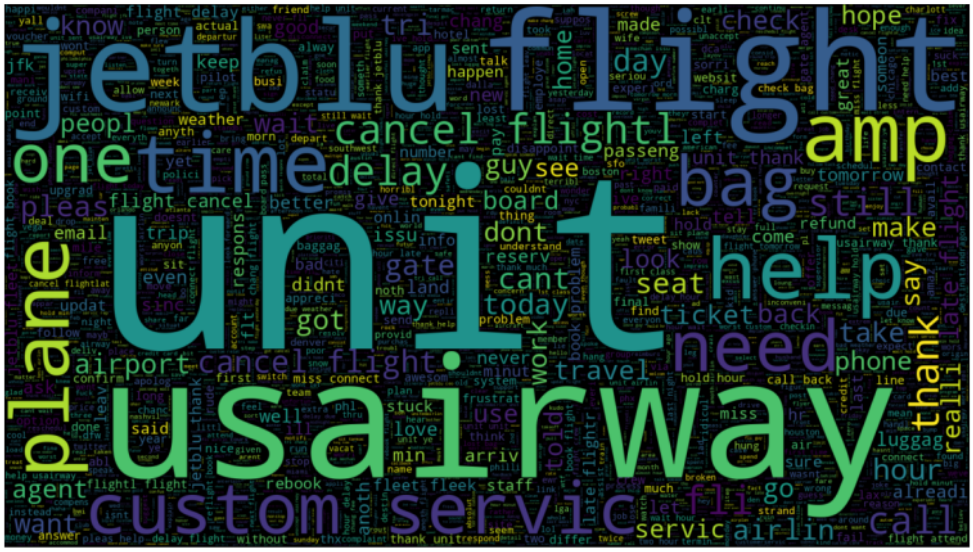
3.7 Handeling airline_sentiments

[15]:
```
from sklearn.preprocessing import LabelEncoder # For Encoding the airline␣
 ↪sentiments as
"""
## 0 --> Negative
## 1 --> Neutral
## 2 --> Positive

"""
label_enc = LabelEncoder()
dataset['airline_sentiment_encoded'] = label_enc.
 ↪fit_transform(dataset['airline_sentiment'])
# dataset.head()

result = dataset['airline_sentiment_encoded']
```

[16]:
```
def detokenise(first_Remove):
    text = ' '.join([str(word) for word in first_Remove])
    return text

dataset['detokenise_sentance'] = dataset['first_Remove'].apply(lambda x :␣
 ↪detokenise(x))
```

4 Visualize the Maximum Repeated Words

[17]:
```
from wordcloud import WordCloud

text = " ".join(dataset['detokenise_sentance'].values)
plt.figure(figsize = (24,12))
wordcloud_figure = WordCloud(min_font_size = 2,  max_words = 3300 , width =␣
 ↪1600 , height = 900).generate(text)

plt.imshow(wordcloud_figure, interpolation='bilinear')
"""# Removig axis """

plt.axis("off")
plt.show()
```



5 APPLYING VECTORISATION

[18]:
```
 ## Applying TFIDF vectorisatoin
```

8

```
from sklearn.feature_extraction.text import TfidfVectorizer as tfidf
 ↪          ## Applying TFIDF vectorisatoin
tfidf_vect = tfidf(use_idf=True , lowercase=False)

corpus = dataset['detokenise_sentance']

X = tfidf_vect.fit(corpus)

X = tfidf_vect.transform(corpus)
```

15

```
# df = pd.DataFrame(X.toarray(), columns = tfidf_vect.get_feature_names())
# df
```

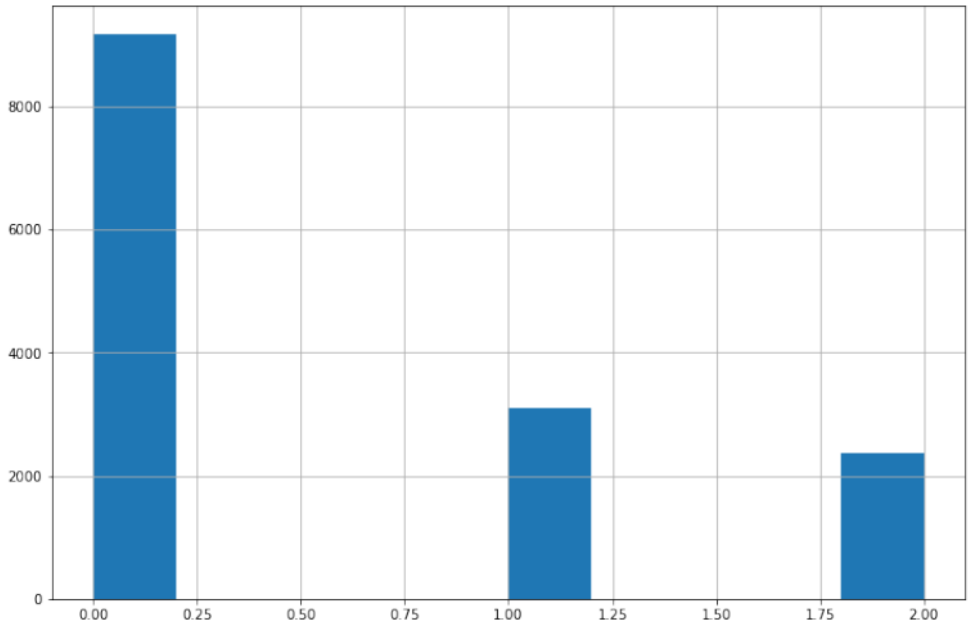
6 Handle Imbalanced Dataset

Negative data has been down sampled to balance the proportion of negative and positive tweets

[19]:
```
## 0 --> Negative
## 1 --> Neutral
## 2 --> Positive
```

```
Y_final = dataset['airline_sentiment_encoded']

dataset['airline_sentiment_encoded'].hist()
```

[19]: <AxesSubplot:>



[20]:
```
# Handling Imbalanced
from imblearn.over_sampling import SMOTE

smote = SMOTE()
X_sm,Y_sm = smote.fit_resample(X,Y_final)
```

7 Train - Test split

[21]:
```
training1 , testing1 , training2 , testing2 = train_test_split(X_sm , Y_sm ,
↪test_size=0.33,random_state=42)
```

8 Report Generator

[22]:
```
def report_gen(testing2 , prediction):
    # defining the report Generation function
    print(" ")
    sns.heatmap(confusion_matrix(testing2, prediction) , fmt = 'd' ,annot =True)
    print(classification_report(testing2, prediction))
    print(" ")
    accuracy = accuracy_score(testing2, prediction)
    print(accuracy)
    return accuracy
```

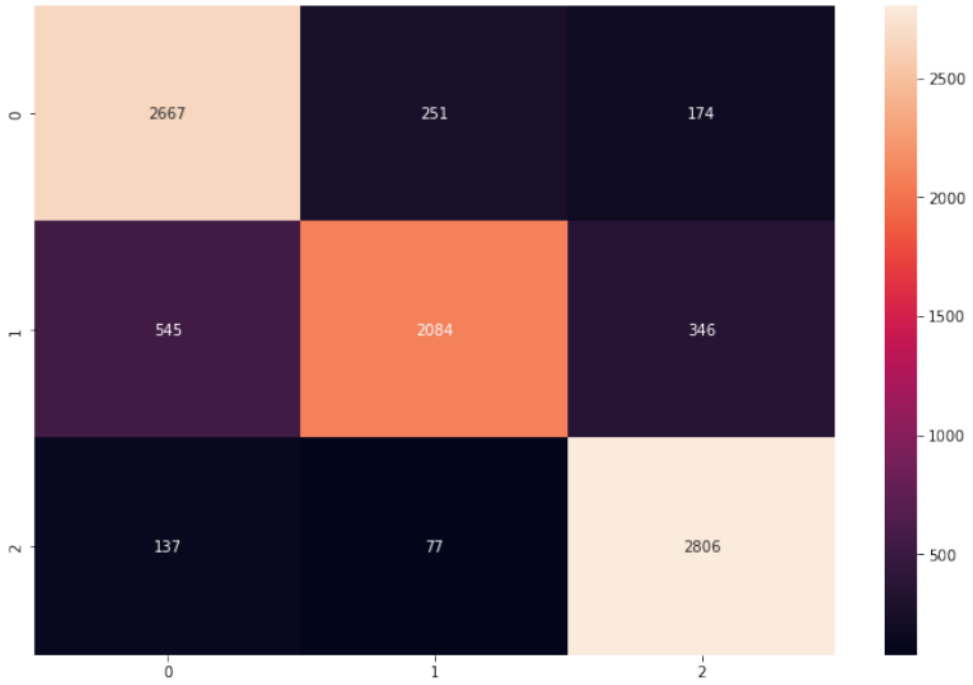
9 Applying Models

9.1 Naive bayes

[23]:
```
naive_bayes = MultinomialNB(alpha=.7)   #try gridsearch
naive_bayes.fit(training1, training2)

naive_bayes_pred = naive_bayes.predict(testing1)
```

[24]:
```
accuracy_nb = report_gen(testing2 , naive_bayes_pred)
```

```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83      3092
           1       0.86      0.70      0.77      2975
           2       0.84      0.93      0.88      3020

    accuracy                           0.83      9087
   macro avg       0.83      0.83      0.83      9087
weighted avg       0.83      0.83      0.83      9087


0.8316275998679432
```



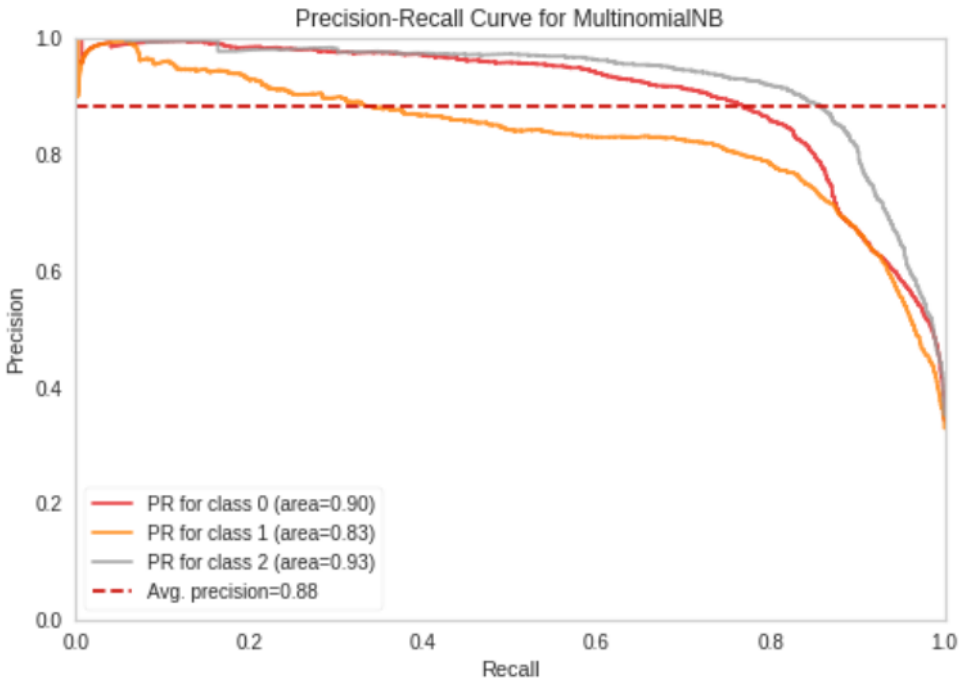
9.1.1 Precesion-Recall Curve for naive bayes

[25]:
```
from yellowbrick.classifier import PrecisionRecallCurve
```

[26]:
```
def precision_recall_curve(curve_model, trained_model, model_name):
    # Defining precision recall curve function
    curv = PrecisionRecallCurve(
        curve_model,
        classes=trained_model.classes_,
        per_class=True,
        cmap="Set1"
    )
    curv.fit(training1,training2)
    # Fitting our data
    curv.score(testing1, testing2)
    # Checking Score for the test data
#     reports_dict[model_name]["Precision-Recall Curve"] = curve.
score_["negative"]
    curv.show();
```

[27]:
```
precision_recall_curve(MultinomialNB(), naive_bayes, "Naive Bayes")
                        # Precession recall curve for naive bayes
```

```
/home/ar/.local/lib/python3.9/site-
packages/yellowbrick/classifier/prcurve.py:254: YellowbrickWarning: micro=True
is ignored;specify per_class=False to draw a PR curve after micro-averaging
  warnings.warn(
```



9.2 Logestic Regression

[28]:
```
logistic_regression = LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,intercept_scaling=1, l1_ratio=None,
max_iter=2000,multi_class='auto', n_jobs=None,
penalty='l2',random_state=7824, solver='lbfgs', tol=0.00001,
verbose=0,warm_start=False)
logistic_regression.fit(training1, training2)
```

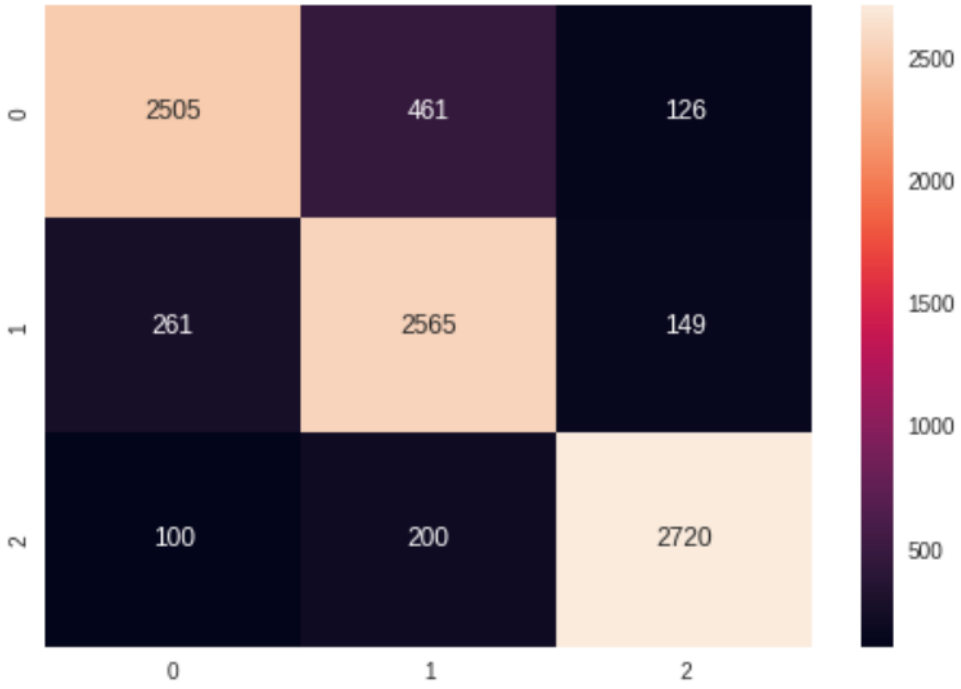
[28]:
```
LogisticRegression(max_iter=2000, random_state=7824, tol=1e-05)
```

[29]:
```
Y_predict_logistic_regression = logistic_regression.predict(testing1)
```

[30]:
```
accuracy_logestic = report_gen(testing2 , Y_predict_logistic_regression)
```

	precision	recall	f1-score	support
0	0.87	0.81	0.84	3092
1	0.80	0.86	0.83	2975
2	0.91	0.90	0.90	3020
accuracy			0.86	9087
macro avg	0.86	0.86	0.86	9087
weighted avg	0.86	0.86	0.86	9087

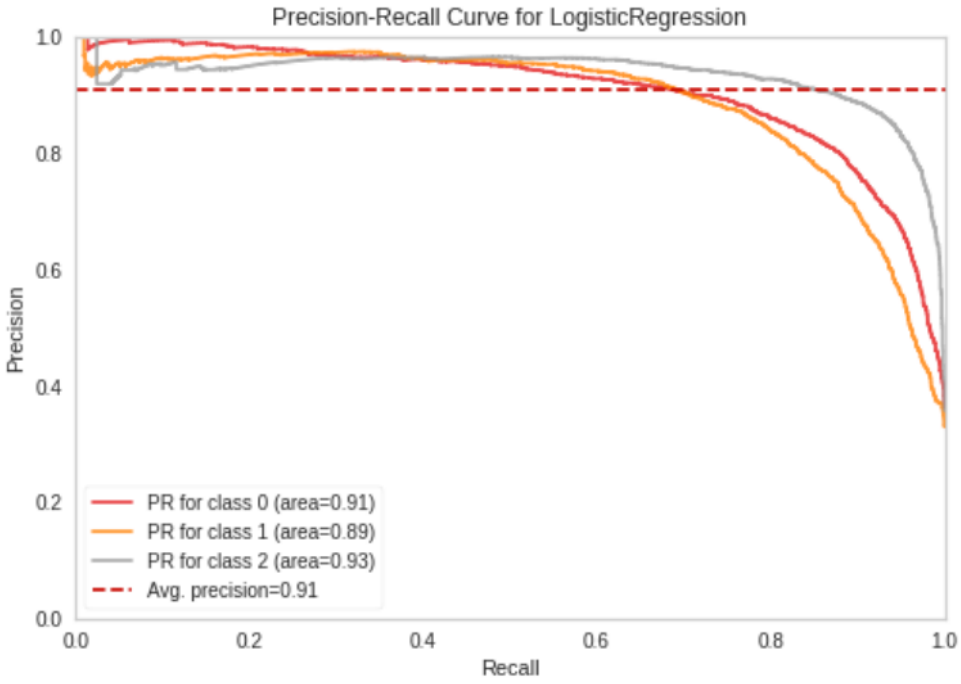
0.8572686255089689



9.2.1 Precesion-Recall Curve for Logestic regression

[32]:
```
precision_recall_curve(LogisticRegression(C=1.0, class_weight=None, dual=False,␣
↪fit_intercept=True,intercept_scaling=1, l1_ratio=None,␣
↪max_iter=1000,multi_class='auto', n_jobs=None,␣
↪penalty='l2',random_state=7823, solver='lbfgs', tol=0.0001,␣
↪verbose=0,warm_start=False), logistic_regression, "Logestic Regression")
```

```
/home/ar/.local/lib/python3.9/site-
packages/yellowbrick/classifier/prcurve.py:254: YellowbrickWarning: micro=True
is ignored;specify per_class=False to draw a PR curve after micro-averaging
  warnings.warn(
```



9.3 Random Forest

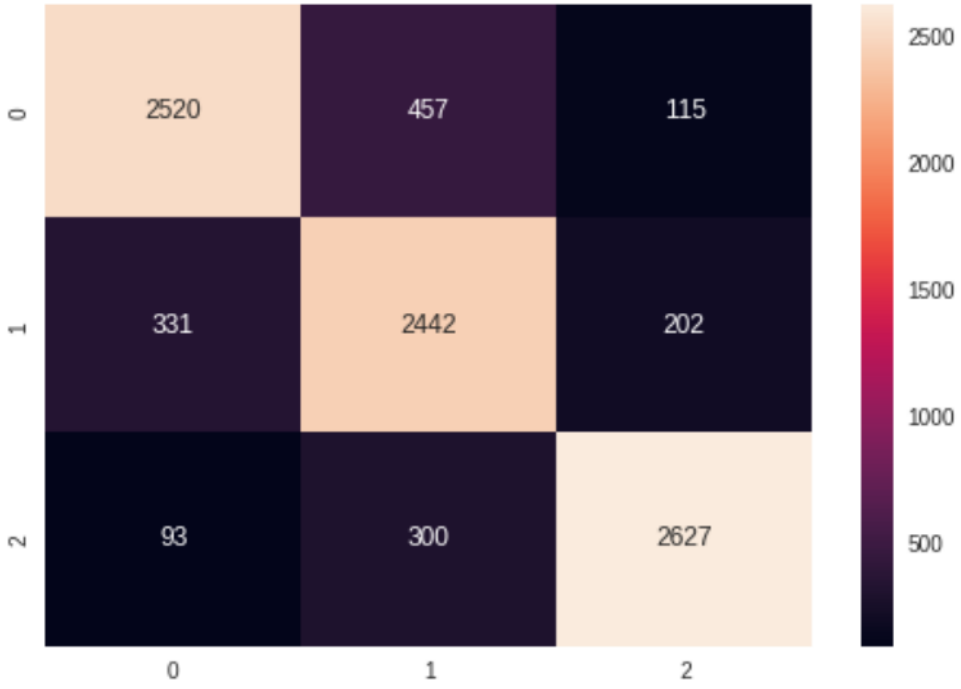
[33]:
```
random_forest = RandomForestClassifier(100,
                           max_depth=40,
                           random_state=0,
                           n_jobs=-1)
random_forest.fit(training1, training2)

random_forest_predict = random_forest.predict(testing1)
```

[34]:
```
accuracy_rf = report_gen(testing2 , random_forest_predict)
```

	precision	recall	f1-score	support
0	0.86	0.82	0.83	3092
1	0.76	0.82	0.79	2975
2	0.89	0.87	0.88	3020

```
    accuracy                           0.84      9087
   macro avg       0.84      0.84      0.84      9087
weighted avg       0.84      0.84      0.84      9087
```

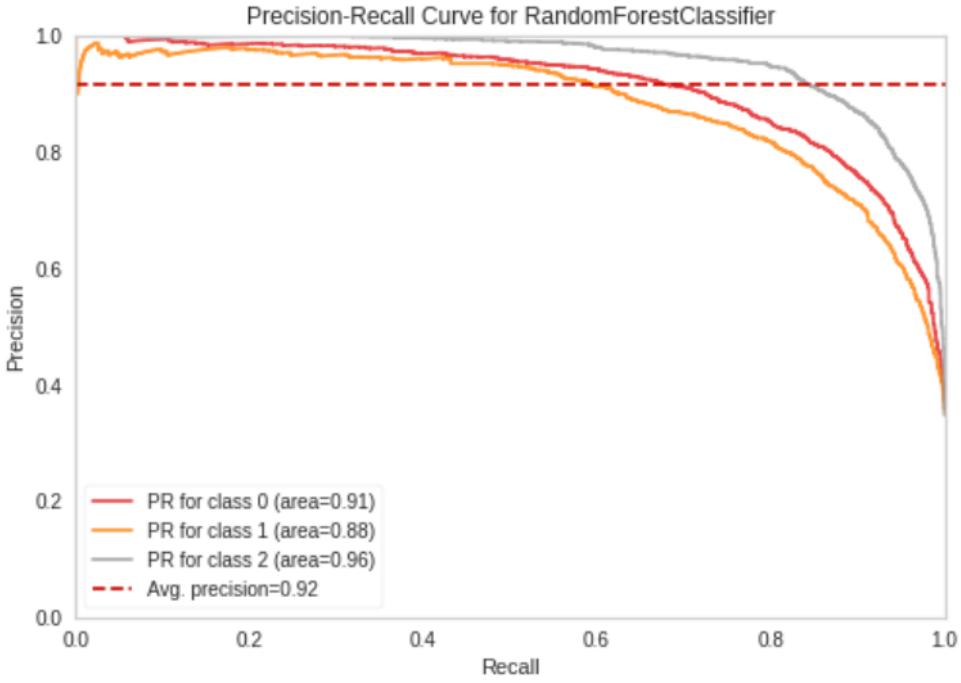
```
0.8351491141190712
```



9.3.1 Precesion-Recall Curve for Random Forest

[35]:
```
precision_recall_curve(RandomForestClassifier(100,␣
↪max_depth=40,random_state=0,n_jobs=-1),random_forest,"Random Forest",)      ␣
↪                          # Precession recall curve for Random  Forest
```

```
/home/ar/.local/lib/python3.9/site-
packages/yellowbrick/classifier/prcurve.py:254: YellowbrickWarning: micro=True
is ignored;specify per_class=False to draw a PR curve after micro-averaging
  warnings.warn(
```

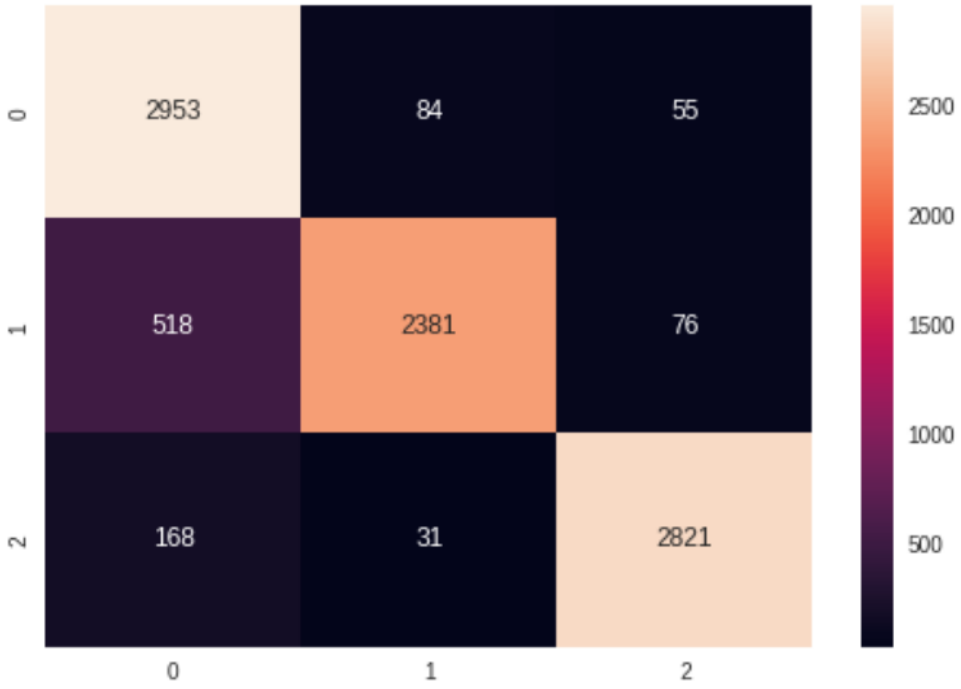


10 Using SVM for better result

[36]:
```
from sklearn import svm
  ↪ # importing SVM

classf = svm.SVC(probability=True)
classf.fit(training1, training2)

classf_pred_svc = classf.predict(testing1)

accuracy_svm = report_gen(testing2 , classf_pred_svc)
```

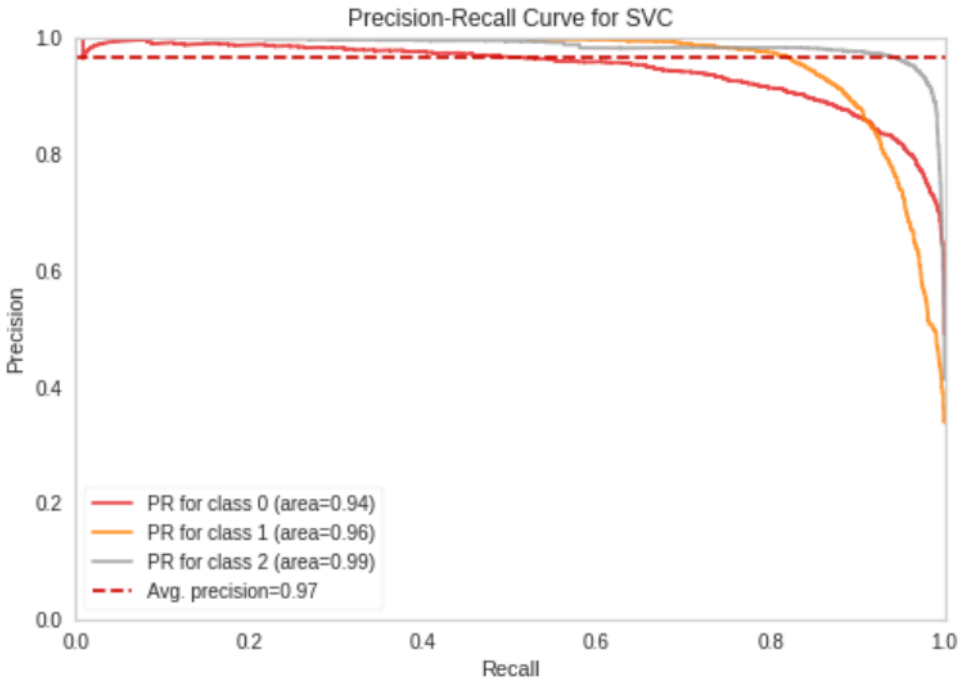
	precision	recall	f1-score	support
0	0.81	0.96	0.88	3092
1	0.95	0.80	0.87	2975
2	0.96	0.93	0.94	3020
accuracy			0.90	9087
macro avg	0.91	0.90	0.90	9087

```
weighted avg       0.91      0.90      0.90      9087

0.8974358974358975
```



[37]:
```
precision_recall_curve(svm.SVC(probability=True),classf,"SVM",)
 ↪                    # Precession recall curve for SVM
```

```
/home/ar/.local/lib/python3.9/site-
packages/yellowbrick/classifier/prcurve.py:254: YellowbrickWarning: micro=True
is ignored;specify per_class=False to draw a PR curve after micro-averaging
  warnings.warn(
```



10.1 SVM with gridsearchCV

[38]:
```
from sklearn.model_selection import GridSearchCV                    # """Using
 ↪gridsrearchCV + SVM to get better Results"""

""" # Tuning the hyperparameters """

parametr ={
    "C":[0.1,1,10],
    "kernel":['linear', 'rbf', 'sigmoid'],
    "gamma":['scale', 'auto']
}

grids = GridSearchCV(estimator=svm.SVC(probability=True), param_grid = parametr
 ↪, cv = 2, verbose=2)
grids.fit(training1,training2)
```

```
Fitting 2 folds for each of 18 candidates, totalling 36 fits
[CV] C=0.1, gamma=scale, kernel=linear …

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] … C=0.1, gamma=scale, kernel=linear, total=  46.5s
```

```
[CV] C=0.1, gamma=scale, kernel=linear …
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   46.5s remaining:    0.0s
[CV] … C=0.1, gamma=scale, kernel=linear, total=  48.4s
[CV] C=0.1, gamma=scale, kernel=rbf …
[CV] … C=0.1, gamma=scale, kernel=rbf, total=  52.5s
[CV] C=0.1, gamma=scale, kernel=rbf …
[CV] … C=0.1, gamma=scale, kernel=rbf, total=  48.8s
[CV] C=0.1, gamma=scale, kernel=sigmoid …
[CV] … C=0.1, gamma=scale, kernel=sigmoid, total=  53.7s
[CV] C=0.1, gamma=scale, kernel=sigmoid …
[CV] … C=0.1, gamma=scale, kernel=sigmoid, total= 1.0min
[CV] C=0.1, gamma=auto, kernel=linear …
[CV] … C=0.1, gamma=auto, kernel=linear, total= 1.0min
[CV] C=0.1, gamma=auto, kernel=linear …
[CV] … C=0.1, gamma=auto, kernel=linear, total=  56.6s
[CV] C=0.1, gamma=auto, kernel=rbf …
[CV] … C=0.1, gamma=auto, kernel=rbf, total= 1.3min
[CV] C=0.1, gamma=auto, kernel=rbf …
[CV] … C=0.1, gamma=auto, kernel=rbf, total= 1.3min
[CV] C=0.1, gamma=auto, kernel=sigmoid …
[CV] … C=0.1, gamma=auto, kernel=sigmoid, total= 1.4min
[CV] C=0.1, gamma=auto, kernel=sigmoid …
[CV] … C=0.1, gamma=auto, kernel=sigmoid, total= 1.3min
[CV] C=1, gamma=scale, kernel=linear …
[CV] … C=1, gamma=scale, kernel=linear, total=  39.8s
[CV] C=1, gamma=scale, kernel=linear …
[CV] … C=1, gamma=scale, kernel=linear, total=  37.1s
[CV] C=1, gamma=scale, kernel=rbf …
[CV] … C=1, gamma=scale, kernel=rbf, total=  46.8s
[CV] C=1, gamma=scale, kernel=rbf …
[CV] … C=1, gamma=scale, kernel=rbf, total=  52.8s
[CV] C=1, gamma=scale, kernel=sigmoid …
[CV] … C=1, gamma=scale, kernel=sigmoid, total=  46.7s
[CV] C=1, gamma=scale, kernel=sigmoid …
[CV] … C=1, gamma=scale, kernel=sigmoid, total=  43.1s
[CV] C=1, gamma=auto, kernel=linear …
[CV] … C=1, gamma=auto, kernel=linear, total=  43.4s
[CV] C=1, gamma=auto, kernel=linear …
[CV] … C=1, gamma=auto, kernel=linear, total=  39.7s
[CV] C=1, gamma=auto, kernel=rbf …
[CV] … C=1, gamma=auto, kernel=rbf, total= 1.5min
[CV] C=1, gamma=auto, kernel=rbf …
[CV] … C=1, gamma=auto, kernel=rbf, total= 1.2min
[CV] C=1, gamma=auto, kernel=sigmoid …
[CV] … C=1, gamma=auto, kernel=sigmoid, total= 1.2min
[CV] C=1, gamma=auto, kernel=sigmoid …
[CV] … C=1, gamma=auto, kernel=sigmoid, total= 1.3min
```

```
[CV] C=10, gamma=scale, kernel=linear …
[CV] … C=10, gamma=scale, kernel=linear, total=  30.9s
[CV] C=10, gamma=scale, kernel=linear …
[CV] … C=10, gamma=scale, kernel=linear, total=  31.8s
[CV] C=10, gamma=scale, kernel=rbf …
[CV] … C=10, gamma=scale, kernel=rbf, total=  53.2s
[CV] C=10, gamma=scale, kernel=rbf …
[CV] … C=10, gamma=scale, kernel=rbf, total=  46.1s
[CV] C=10, gamma=scale, kernel=sigmoid …
[CV] … C=10, gamma=scale, kernel=sigmoid, total=  23.0s
[CV] C=10, gamma=scale, kernel=sigmoid …
[CV] … C=10, gamma=scale, kernel=sigmoid, total=  22.9s
[CV] C=10, gamma=auto, kernel=linear …
[CV] … C=10, gamma=auto, kernel=linear, total=  31.2s
[CV] C=10, gamma=auto, kernel=linear …
[CV] … C=10, gamma=auto, kernel=linear, total=  32.1s
[CV] C=10, gamma=auto, kernel=rbf …
[CV] … C=10, gamma=auto, kernel=rbf, total= 1.2min
[CV] C=10, gamma=auto, kernel=rbf …
[CV] … C=10, gamma=auto, kernel=rbf, total= 1.3min
[CV] C=10, gamma=auto, kernel=sigmoid …
[CV] … C=10, gamma=auto, kernel=sigmoid, total= 1.1min
[CV] C=10, gamma=auto, kernel=sigmoid …
[CV] … C=10, gamma=auto, kernel=sigmoid, total= 1.2min

[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed: 32.7min finished
```

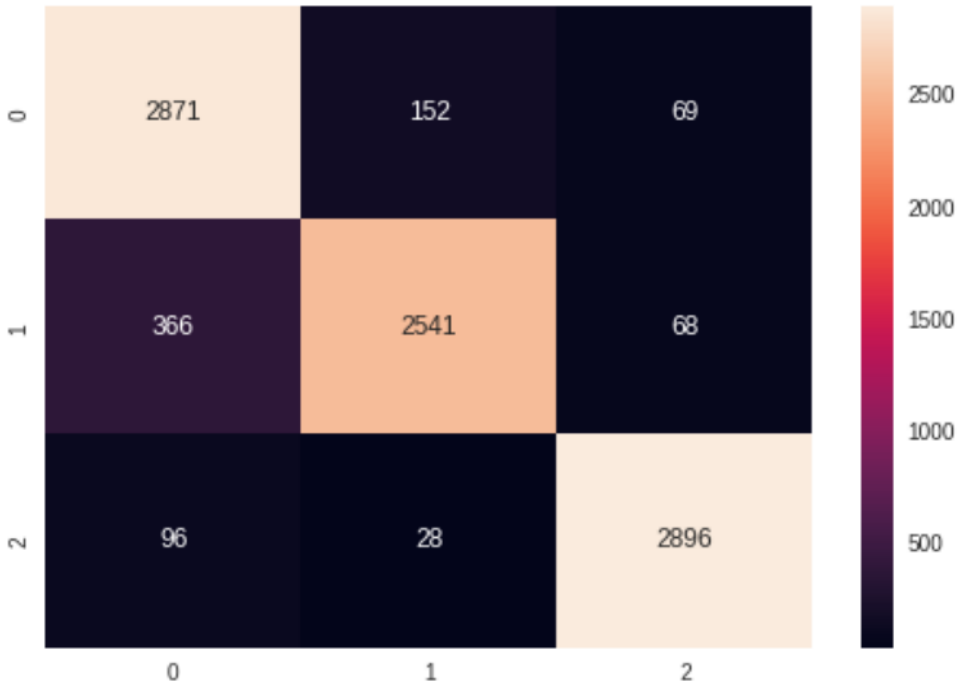
[38]:
```
GridSearchCV(cv=2, estimator=SVC(probability=True),
             param_grid={'C': [0.1, 1, 10], 'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'rbf', 'sigmoid']},
             verbose=2)
```

[39]:
```
predicted_class=grids.predict(testing1)
```

[40]:
```
accuracy_svm_grids = report_gen(testing2 , predicted_class)
```

	precision	recall	f1-score	support
0	0.86	0.93	0.89	3092
1	0.93	0.85	0.89	2975
2	0.95	0.96	0.96	3020
accuracy			0.91	9087
macro avg	0.92	0.91	0.91	9087
weighted avg	0.92	0.91	0.91	9087

```
0.9142731374491031
```



11 Accuracy Result Using Above Models

[41]:
```
print("------------------- Accuracy Score ---------------------")
print(" ")
print(f"Naive Bayes          --  {accuracy_nb * 100}")
print(f"Logestic Regression  --  {accuracy_logestic * 100}")
print(f"Random Forest        --  {accuracy_rf * 100}")

print("--------------------------------------------------------")
print(" ")
print(f"Accuracy using SVM                 : {accuracy_svm * 100}")
print(f"Accuracy using SVM with Gridsearch : {accuracy_svm_grids * 100}")
```

```
------------------- Accuracy Score ---------------------

Naive Bayes          --  83.16275998679433
Logestic Regression  --  85.72686255089688
Random Forest        --  83.51491141190712
--------------------------------------------------------

Accuracy using SVM                 : 89.74358974358975
```

```
Accuracy using SVM with Gridsearch : 91.42731374491031
```

Sentiment analysis - Proposed

ORIGINALITY REPORT

23%	22%	14%	21%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to RMIT University Student Paper	11%
2	htmtopdf.herokuapp.com Internet Source	1%
3	www.datatechnotes.com Internet Source	1%
4	repository.seeu.edu.mk Internet Source	1%
5	github.com Internet Source	1%
6	www.coursehero.com Internet Source	1%
7	Submitted to University of Glamorgan Student Paper	1%
8	medium.com Internet Source	1%
9	Submitted to Middlesex University Student Paper	1%

10	Submitted to University of Queensland Student Paper	1 %
11	mohitagr18.github.io Internet Source	1 %
12	software-data-mining.com Internet Source	1 %
13	machinelearningmastery.com Internet Source	<1 %
14	www.datasklr.com Internet Source	<1 %
15	Jay M. Patel. "Getting Structured Data from the Internet", Springer Science and Business Media LLC, 2020 Publication	<1 %
16	Submitted to University College London Student Paper	<1 %
17	nsing.tistory.com Internet Source	<1 %
18	Dipanjan Sarkar. "Text Analytics with Python", Springer Science and Business Media LLC, 2019 Publication	<1 %
19	Dipanjan Sarkar, Raghav Bali, Tushar Sharma. "Practical Machine Learning with Python",	<1 %

Springer Science and Business Media LLC, 2018

Publication

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		