MACHINE LEARNING

CASE STUDY REPORT

TOPIC: SVM-Support Vector Machines

Submitted by-

Ananya Parida (3)

Moulesh M (12)

Nyshadham Tharak Yogindra (15)

TABLE OF CONTENTS

SUMMARY OF REPORT :

In this case study,

we will look into the nitty-gritty details of SVM and how it is implemented in the real world to solve any business problem. As well the report covers about the different forms of SVM and how it is able to solve any linear or non linear problem. We will understand how hyperparameter tuning is done in SVM and also a 'text' dataset is taken from Kaggle in which Support vector classifier is trained and results are analyzed. Finally, possible alternatives of SVM are also discussed.

INTRODUCTION:

A Support Vector Machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labelled training data for each category, they're able to categorize new text.

This algorithm is most commonly used for solving classification problems and is also referred to as Support Vector Classification. There is also a subset of SVM called SVR which stands for Support Vector Regression which uses the same principles to solve regression problems. SVM also supports the kernel method also called the kernel SVM which allows us to tackle non-linearity.

Support Vector Machines are fast and dependable classification algorithm that performs very well with a limited amount of data to analyse.

Types of SVM:

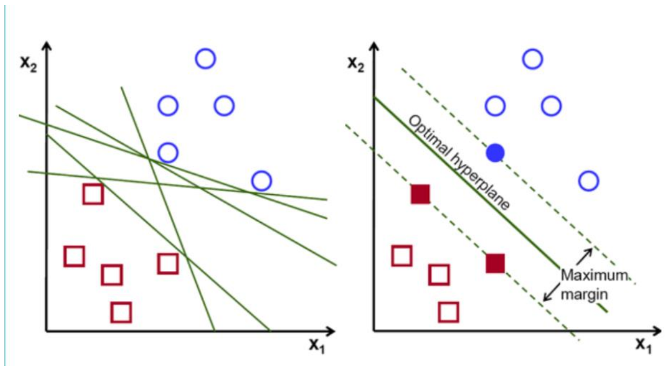
I. **Linear SVM:** Linear SVM is used for data that are linearly separable i.e., for a dataset that can be categorized into two categories by utilizing a single straight line. Such data points are termed as linearly separable data, and the classifier is used described as a Linear SVM classifier.

SVM USED FOR CLASSIFICATION

Algorithm

Taking a binary classification problem, the task is to efficiently classify a test point in either of the classes as accurately as possible. Following are the steps involved in the SVM process:

Firstly, set of points belonging to the two classes are plotted and visualized as shown below. In a 2-d space by just applying a straight line, we can efficiently divide these two classes. But there can be many lines that can classify these classes. There are a set of lines or hyperplanes (green lines) to choose from.



To know the line out of all these lines that is suitable for classification, the hyper-plane is selected which separates the two classes better. This is done by maximizing the distance between the closest data point and the hyper-plane.

In a nutshell, the optimal hyperplane has equation w.x+b = 0. The left support vector has equation w.x+b=-1 and the right support vector has w.x+b=1.
Thus, the distance d between two parallel liens Ay = Bx + c1 and Ay = Bx + c2 is given by d = |C1–C2|/√A^2 + B^2. With this formula in place, we have the distance between the two support vectors as 2/||w||.

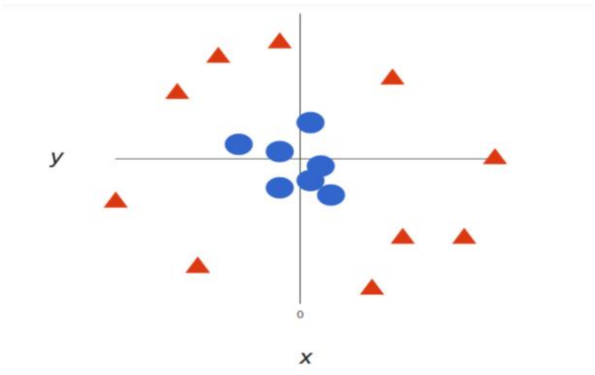
The cost function for SVM looks the like the equation below:

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\left[\frac{1}{N}\sum_{i}^{n} \max\left(0,\ 1 - y_i * (\mathbf{w}\cdot x_i + b)\right)\right]$$

SVM loss function
In the cost function equation above, the λ parameter denotes that a larger λ provides a broader margin, and a smaller λ would yield a smaller margin. Furthermore, the gradient of the cost function is calculated and the weights are updated in the direction that lowers the lost function.

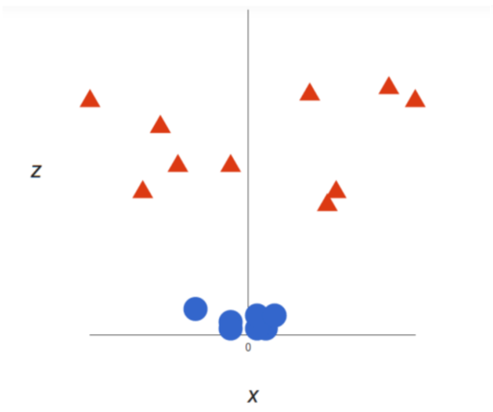
II. **Non-linear SVM:** Non-Linear SVM is used for data that are non-linearly separable data i.e., a straight line cannot be used to classify the dataset. For this, we use something known as a kernel trick that sets data points in a higher dimension where they can be separated using planes or other mathematical functions. Such data points are termed as non-linear data, and the classifier used is termed as a Non-linear SVM classifier.

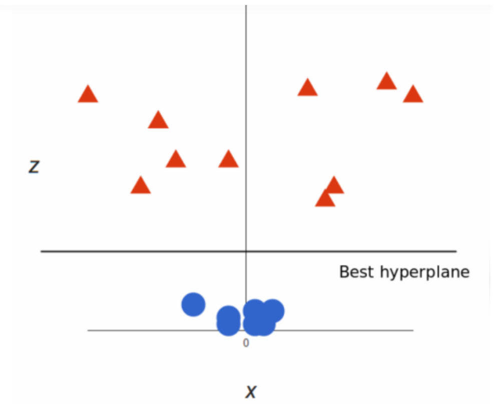
Algorithm
For Non-Linear SVM, algorithm has a method called the kernel trick.
The SVM kernel function takes in low dimensional input space and converts it to a higher-dimensional space. In simple words, it converts the non-separable problem to a separable problem. It performs complex data transformations based on the labels or outputs that define them.



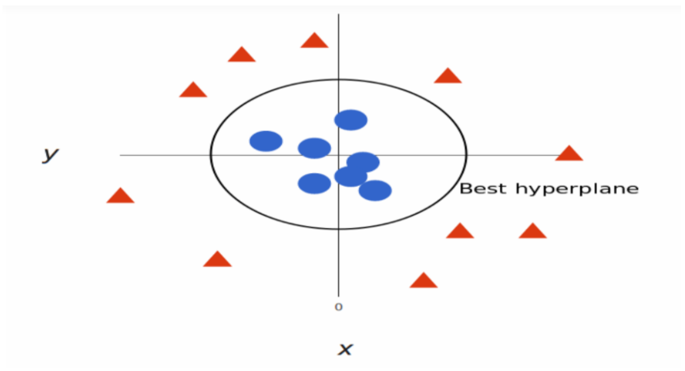
To separate non linearly separable data points, we have to add an extra dimension. For linear data, two dimensions have been used, that is, x and y. For these data points, we add a third dimension, say z. For the example below let $z=x^2+y^2$.

This z function or the added dimensionality transforms the sample space and the above image will become as the following:





It is clear that the above data points can be separated using a straight-line function that is either parallel to the x axis or is inclined at an angle. Different types of kernel functions are present — linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.



Hyperparameters of the Support Vector Machine (SVM) Algorithm

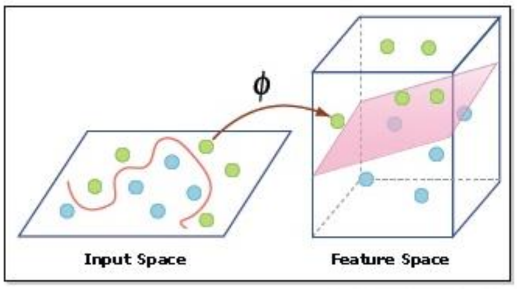
What is **hyperparameter tuning**?

Hyper parameters are [SVC (gamma="scale")] the things in brackets when we are defining a classifier or a regressor or any algo.

Hyperparameters are properties of the algorithm that help classify or regress the dataset when you increase or decrease them for example.

There are a few important parameters of SVM that you should be aware of before proceeding further:

- **Kernel:** A kernel helps us find a hyperplane in the higher dimensional space without increasing the computational cost. Usually, the computational cost will increase if the dimension of the data increases. This increase in dimension is required when we are unable to find a separating hyperplane in a given dimension and are required to move in a higher dimension:



- **Hyperplane:** This is basically a separating line between two data classes in SVM. But in Support Vector Regression, this is the line that will be used to predict the continuous output.
- **Decision Boundary**: A decision boundary can be thought of as a demarcation line (for simplification) on one side of which lie positive examples and on the other side lie the negative examples. On this very line, the examples may be classified as either positive or negative. This same concept of SVM will be applied in Support Vector Regression as well.

Choosing Kernel:

A nice method to determine which kernel is the most suitable is to make various models with varying kernels, then estimate each of their performance, and ultimately compare the outcomes. Then you pick the kernel with the best results. Be particular to estimate the model's performance on unlike observations by using K-Fold Cross-Validation and consider different metrics like Accuracy, F1 Score, etc.

<u>IMPLEMENTATION OF SVM</u>:

About dataset:-

This dataset is contributed by NLP community which deals with various emotions like sadness, joy, anger ,fear ,love and surprise . Also this dataset can be used for multiclass classification tasks with some primary text cleaning & preprocessing following the modelling.

Lets see how SVM is able to model this dataset.

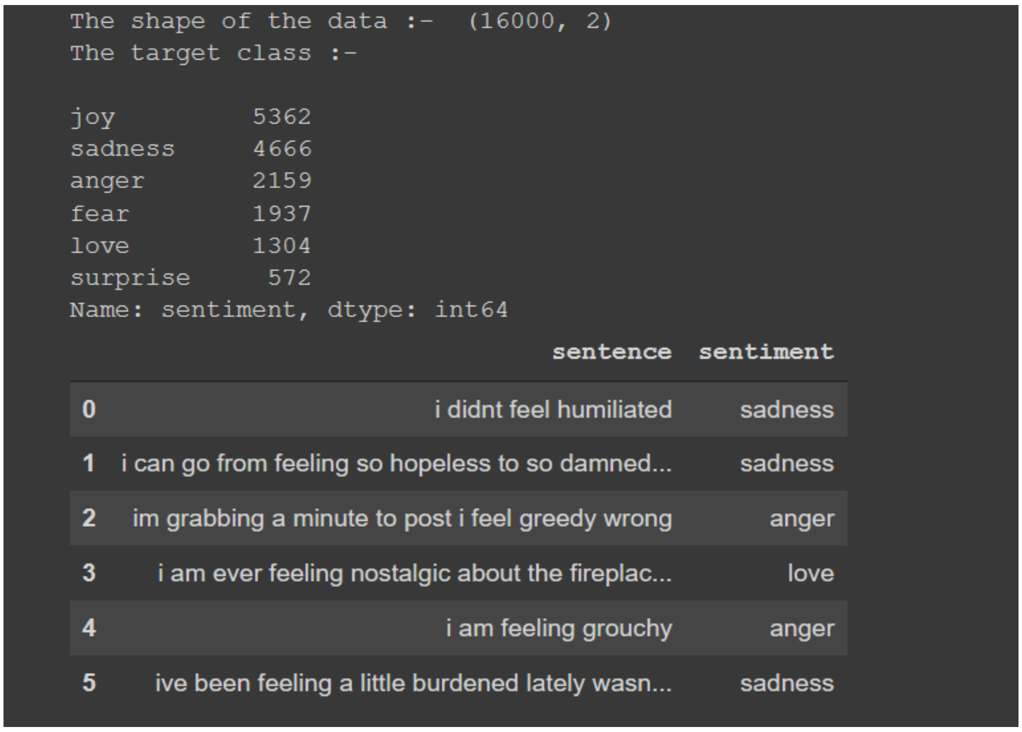
A glimpse of my dataset:-

I am really worried about my exams - - - > emotes **FEAR**

Just get lost from here - - - >emotes **ANGER**

I am more excited for this event - - - > emotes **joy**

Display dataset :-



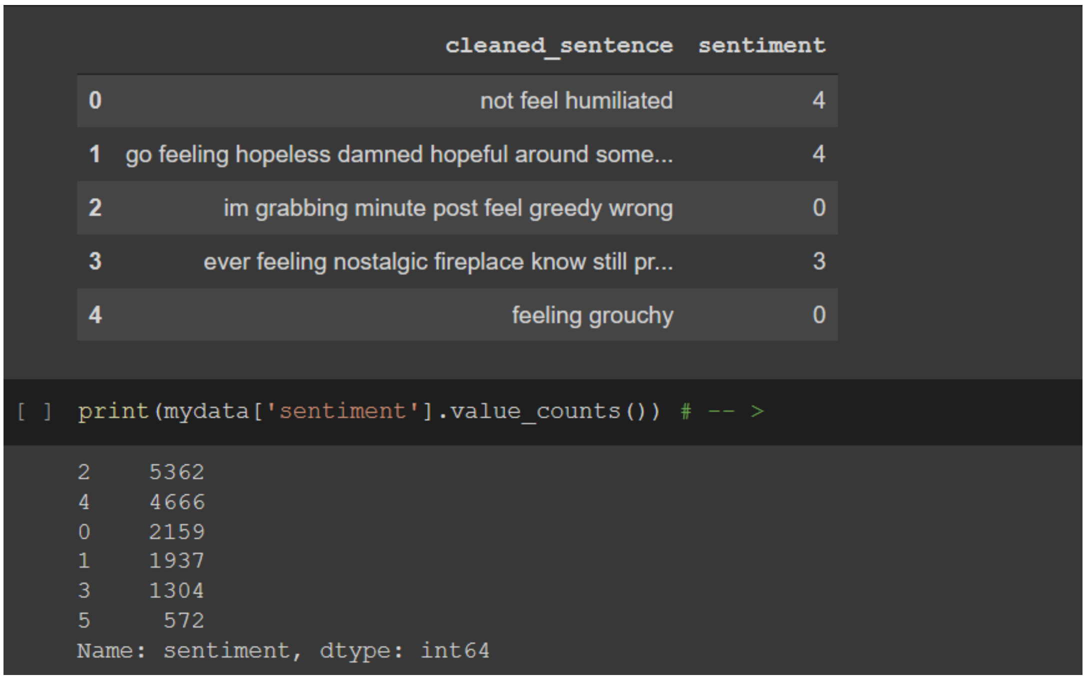
```
The shape of the data :-  (16000, 2)
The target class :-

joy         5362
sadness     4666
anger       2159
fear        1937
love        1304
surprise     572
Name: sentiment, dtype: int64
```

	sentence	sentiment
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger
5	ive been feeling a little burdened lately wasn...	sadness

Observation :-

Here, We can clearly observe that the dataset has target column with '6' classes.

After text cleaning & preprocessing :-



	cleaned_sentence	sentiment
0	not feel humiliated	4
1	go feeling hopeless damned hopeful around some...	4
2	im grabbing minute post feel greedy wrong	0
3	ever feeling nostalgic fireplace know still pr...	3
4	feeling grouchy	0

```
[ ]  print(mydata['sentiment'].value_counts()) # -- >
```

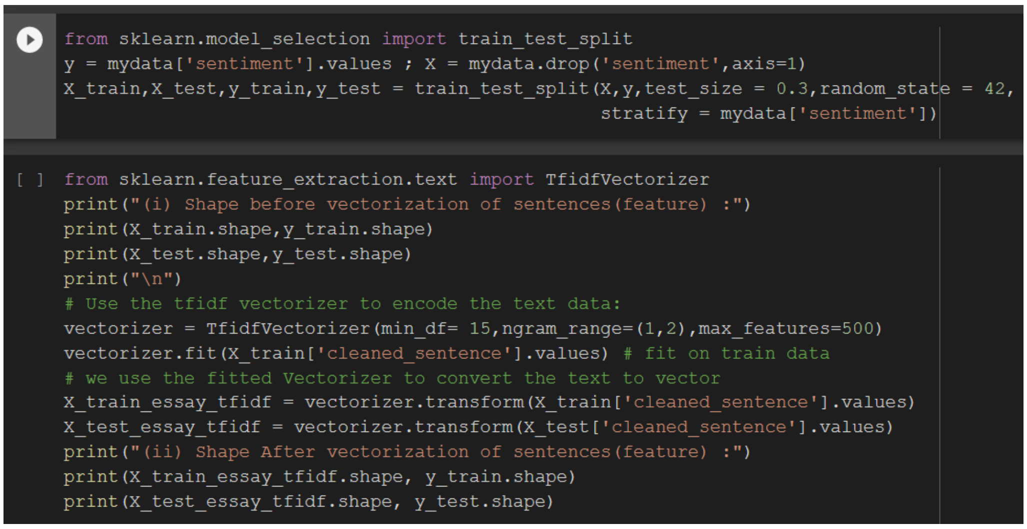
```
2    5362
4    4666
0    2159
1    1937
3    1304
5     572
Name: sentiment, dtype: int64
```

Observation :-

Here, After cleaning & preprocessing we can **clearly observe that the stopwords, punctuations and digits** are removed. As well the **target column** is transformed with the numerical labels as

0 - - > anger ; 1 - - > fear ; 2 - - > joy ; 3 - - > love ; 4 - - > sadness ; 5 - - > surprise

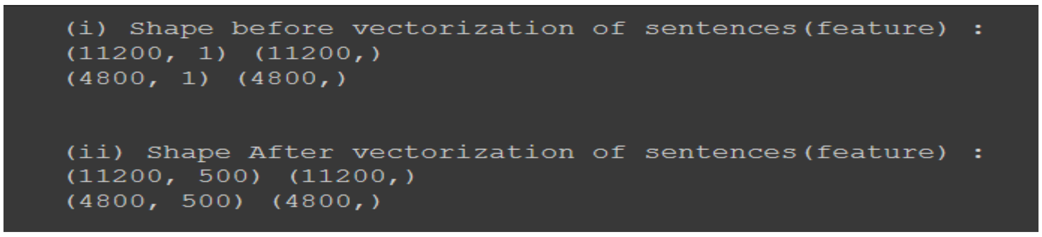
After Text featurization :-



```
from sklearn.model_selection import train_test_split
y = mydata['sentiment'].values ; X = mydata.drop('sentiment',axis=1)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.3,random_state = 42,
                                                 stratify = mydata['sentiment'])
```

```
[ ]  from sklearn.feature_extraction.text import TfidfVectorizer
     print("(i) Shape before vectorization of sentences(feature) :")
     print(X_train.shape,y_train.shape)
     print(X_test.shape,y_test.shape)
     print("\n")
     # Use the tfidf vectorizer to encode the text data:
     vectorizer = TfidfVectorizer(min_df= 15,ngram_range=(1,2),max_features=500)
     vectorizer.fit(X_train['cleaned_sentence'].values) # fit on train data
     # we use the fitted Vectorizer to convert the text to vector
     X_train_essay_tfidf = vectorizer.transform(X_train['cleaned_sentence'].values)
     X_test_essay_tfidf = vectorizer.transform(X_test['cleaned_sentence'].values)
     print("(ii) Shape After vectorization of sentences(feature) :")
     print(X_train_essay_tfidf.shape, y_train.shape)
     print(X_test_essay_tfidf.shape, y_test.shape)
```

Dimensions:-

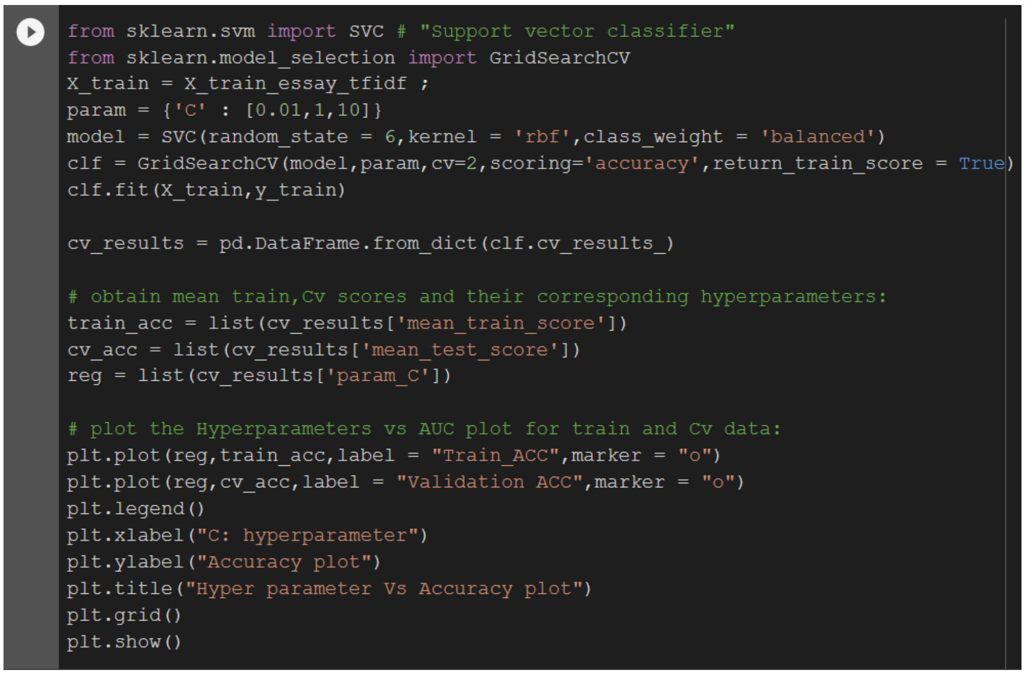


```
(i) Shape before vectorization of sentences(feature) :
(11200, 1) (11200,)
(4800, 1) (4800,)


(ii) Shape After vectorization of sentences(feature) :
(11200, 500) (11200,)
(4800, 500) (4800,)
```

Observations:-

Here, after text featurization on **train[70%] & test[30%]** data we can observe that the column dimensions are 500 (i.e we have 500 features now) with its **tfidf values**.

Apply svm on train & cross validation data with hyperparameter tuning:-

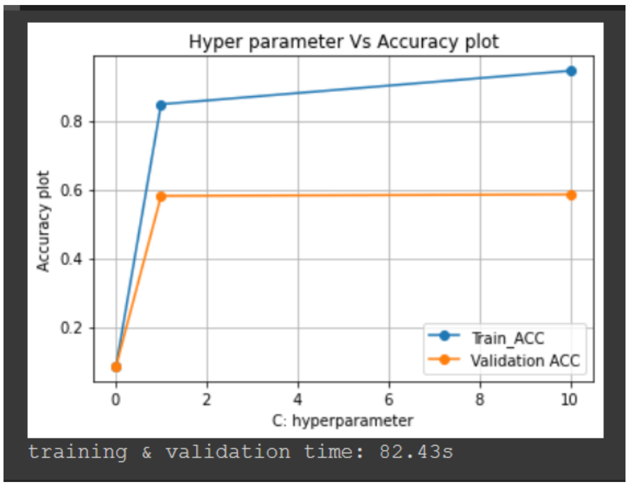


```
from sklearn.svm import SVC # "Support vector classifier"
from sklearn.model_selection import GridSearchCV
X_train = X_train_essay_tfidf ;
param = {'C' : [0.01,1,10]}
model = SVC(random_state = 6,kernel = 'rbf',class_weight = 'balanced')
clf = GridSearchCV(model,param,cv=2,scoring='accuracy',return_train_score = True)
clf.fit(X_train,y_train)

cv_results = pd.DataFrame.from_dict(clf.cv_results_)

# obtain mean train,Cv scores and their corresponding hyperparameters:
train_acc = list(cv_results['mean_train_score'])
cv_acc = list(cv_results['mean_test_score'])
reg = list(cv_results['param_C'])

# plot the Hyperparameters vs AUC plot for train and Cv data:
plt.plot(reg,train_acc,label = "Train_ACC",marker = "o")
plt.plot(reg,cv_acc,label = "Validation ACC",marker = "o")
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("Accuracy plot")
plt.title("Hyper parameter Vs Accuracy plot")
plt.grid()
plt.show()
```

Observations :-

Here, we have applied SVM on train data with internal cross validation = 2 using gridsearchCV that is present in Sklearn library. And we have done hyperparameter tuning on the regularization parameter "C" which helps in controlling overfitting & underfitting. We also plot the train vs CV accuracy plot for different values of hyperparameter and pick the optimal one.

Output :



```
training & validation time: 82.43s
```

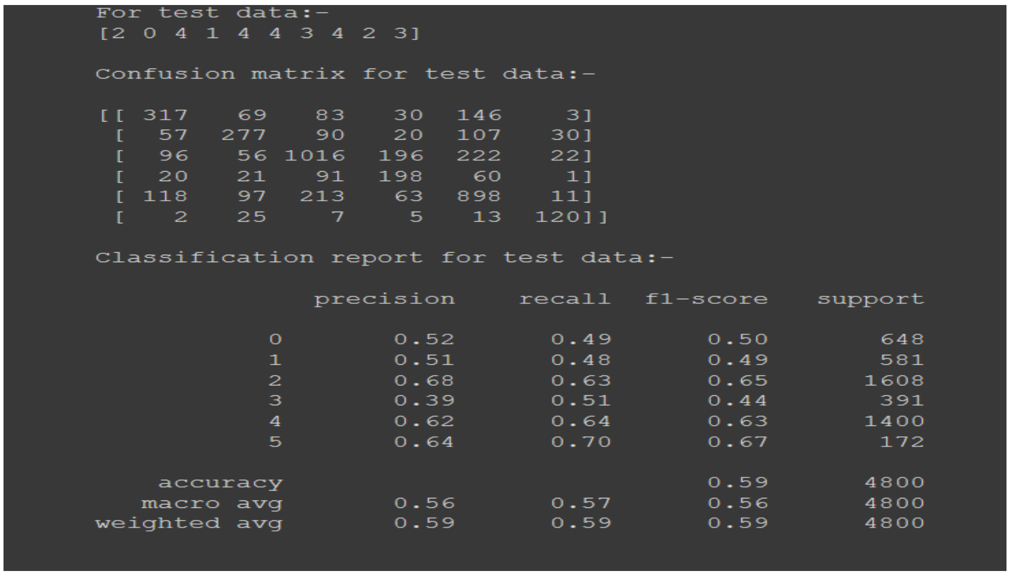
Observation:-

The key inference is that the optimal hyperparameter is C = 1 - - > where the

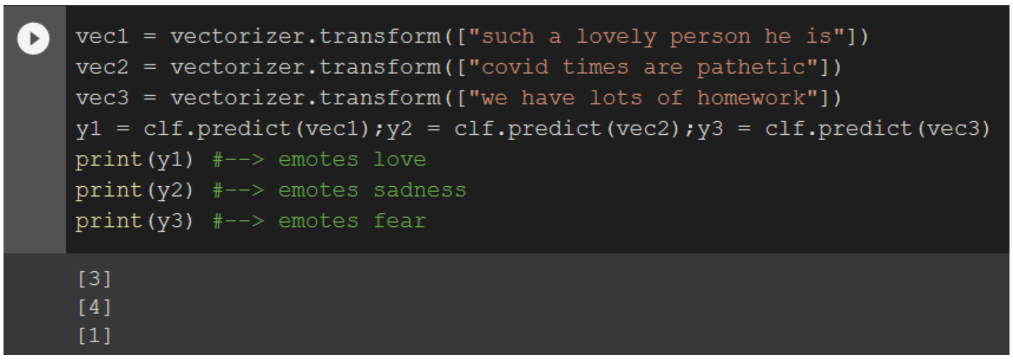
[train accuracy – cv accracy] = 20%

Hence, this hyperparameter ensures that model didn't overfit or underfit.

Results on test data:-



```
For test data:-
[2 0 4 1 4 4 3 4 2 3]

Confusion matrix for test data:-

[[ 317   69   83   30  146    3]
 [  57  277   90   20  107   30]
 [  96   56 1016  196  222   22]
 [  20   21   91  198   60    1]
 [ 118   97  213   63  898   11]
 [   2   25    7    5   13  120]]

Classification report for test data:-

              precision    recall  f1-score   support

           0       0.52      0.49      0.50       648
           1       0.51      0.48      0.49       581
           2       0.68      0.63      0.65      1608
           3       0.39      0.51      0.44       391
           4       0.62      0.64      0.63      1400
           5       0.64      0.70      0.67       172

    accuracy                           0.59      4800
   macro avg       0.56      0.57      0.56      4800
weighted avg       0.59      0.59      0.59      4800
```



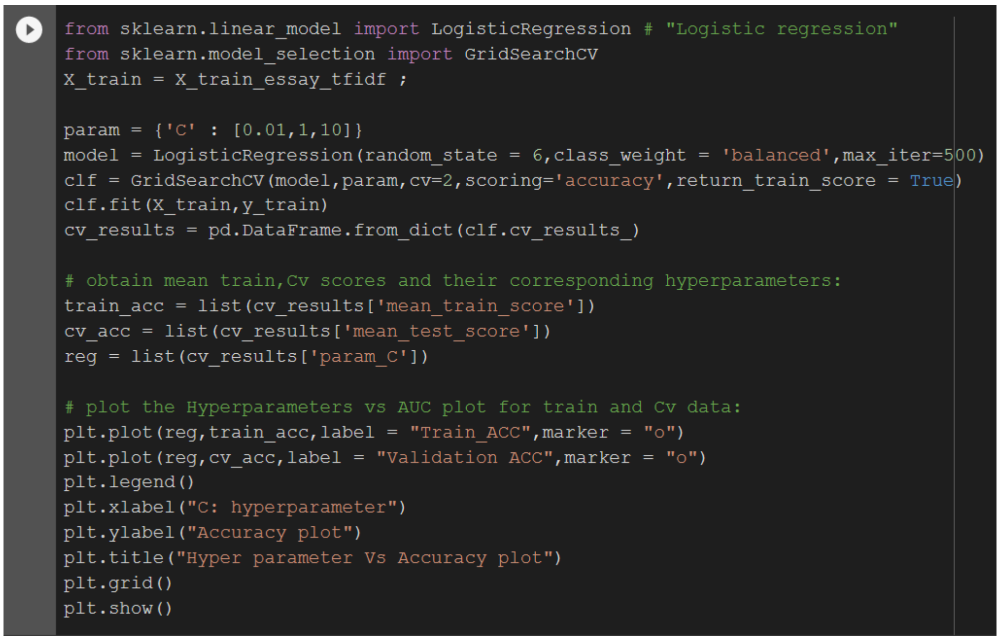
```
vec1 = vectorizer.transform(["such a lovely person he is"])
vec2 = vectorizer.transform(["covid times are pathetic"])
vec3 = vectorizer.transform(["we have lots of homework"])
y1 = clf.predict(vec1);y2 = clf.predict(vec2);y3 = clf.predict(vec3)
print(y1) #--> emotes love
print(y2) #--> emotes sadness
print(y3) #--> emotes fear

[3]
[4]
[1]
```

Observations:-

Here, we use the test data to check the model performance where we infer that accuracy on test data is - - > 59% which is not very bad. And also model is sensible.

POSSIBLE ALTERNATIVES FOR THE ABOVE PROBLEM :

Applying LOGISTIC REGRESSION :

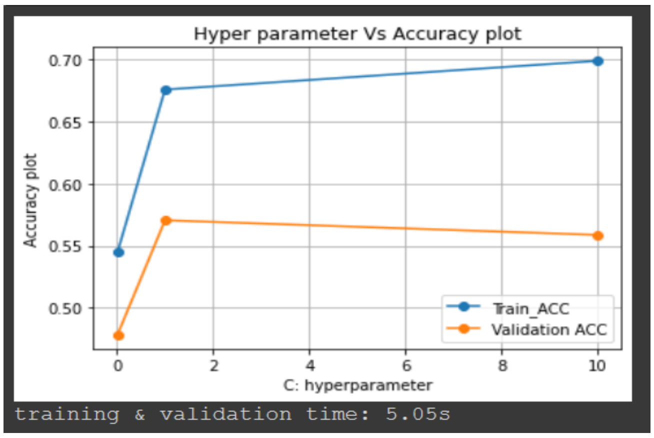


```
from sklearn.linear_model import LogisticRegression # "Logistic regression"
from sklearn.model_selection import GridSearchCV
X_train = X_train_essay_tfidf ;

param = {'C' : [0.01,1,10]}
model = LogisticRegression(random_state = 6,class_weight = 'balanced',max_iter=500)
clf = GridSearchCV(model,param,cv=2,scoring='accuracy',return_train_score = True)
clf.fit(X_train,y_train)
cv_results = pd.DataFrame.from_dict(clf.cv_results_)

# obtain mean train,Cv scores and their corresponding hyperparameters:
train_acc = list(cv_results['mean_train_score'])
cv_acc = list(cv_results['mean_test_score'])
reg = list(cv_results['param_C'])

# plot the Hyperparameters vs AUC plot for train and Cv data:
plt.plot(reg,train_acc,label = "Train_ACC",marker = "o")
plt.plot(reg,cv_acc,label = "Validation ACC",marker = "o")
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("Accuracy plot")
plt.title("Hyper parameter Vs Accuracy plot")
plt.grid()
plt.show()
```

Observation :

Here, we have applied Logistic regression on train data with internal cross validation = 2 using gridsearchCV that is present in Sklearn library. And we have done hyperparameter tuning on the regularization parameter "C" which helps in controlling overfitting & underfitting. We also plot the train vs CV accuracy plot for different values of hyperparameter and pick the optimal one.

Output :-



```
training & validation time: 5.05s
```

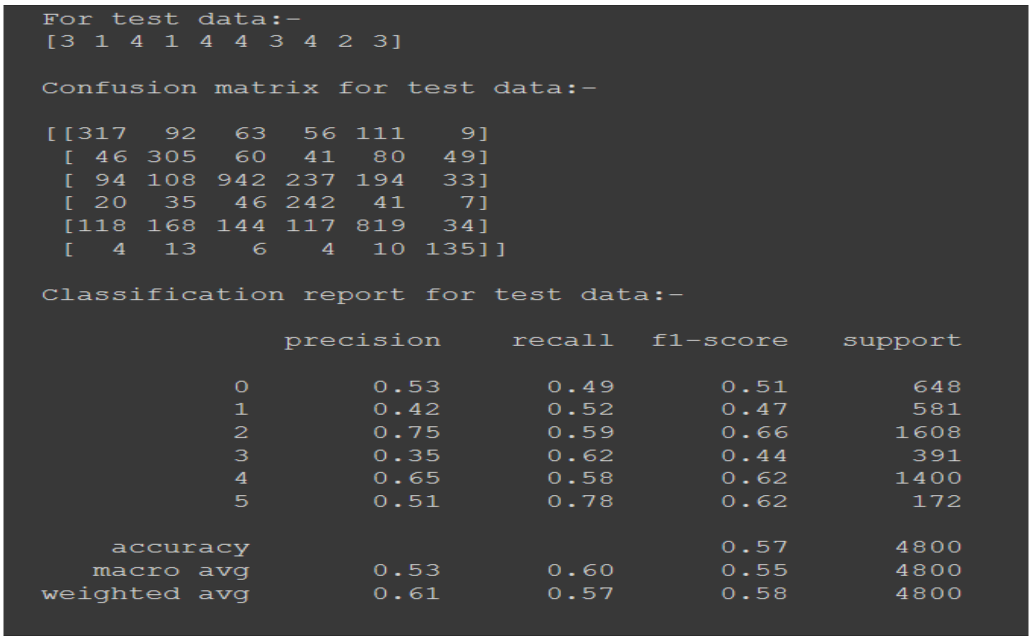
Observation:-

The key inference is that the optimal hyperparameter is C = 1 - - > where the

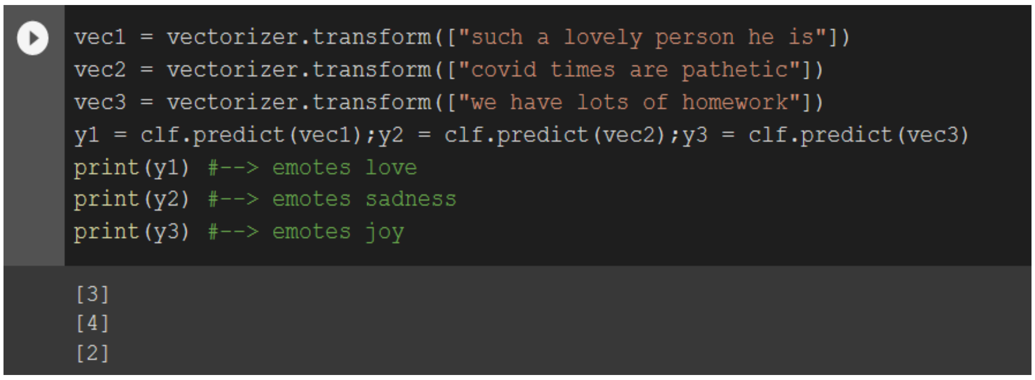
[train accuracy – cv accracy] = 10%

Hence, this hyperparameter ensures that model didn't overfit or underfit.

Results on test data:



```
For test data:-
[3 1 4 1 4 4 3 4 2 3]

Confusion matrix for test data:-

[[317  92  63  56 111   9]
 [ 46 305  60  41  80  49]
 [ 94 108 942 237 194  33]
 [ 20  35  46 242  41   7]
 [118 168 144 117 819  34]
 [  4  13   6   4  10 135]]

Classification report for test data:-

              precision    recall  f1-score   support

           0       0.53      0.49      0.51       648
           1       0.42      0.52      0.47       581
           2       0.75      0.59      0.66      1608
           3       0.35      0.62      0.44       391
           4       0.65      0.58      0.62      1400
           5       0.51      0.78      0.62       172

    accuracy                           0.57      4800
   macro avg       0.53      0.60      0.55      4800
weighted avg       0.61      0.57      0.58      4800
```



```
vec1 = vectorizer.transform(["such a lovely person he is"])
vec2 = vectorizer.transform(["covid times are pathetic"])
vec3 = vectorizer.transform(["we have lots of homework"])
y1 = clf.predict(vec1);y2 = clf.predict(vec2);y3 = clf.predict(vec3)
print(y1) #--> emotes love
print(y2) #--> emotes sadness
print(y3) #--> emotes joy

[3]
[4]
[2]
```

Observations:-

Here, we use the test data to check the model performance where we infer that accuracy on test data is - - > 57% which is not very bad and also model is less sensible than SVM.

REAL WORLD CASES :-

- Automatically helps in feature engineering by leveraging kernel trick.
- Svm can work well if a similarity function is given.
- Obtaining feature importance is again a drawback of Svm.
- Outliers impact is very minimal because only the support vectors that matter in Svm.
- Training time complexity is large - - > O(n^2) where 'n'(i.e the number of data points)
- Given a dataset , Choosing the right kernel is not trivial because we have so many domain specific kernels in real time.

CONCLUSION:-

Thus,

we can conclude that SVM can also work quite well with NLP tasks which has been analyzed in this report. We also observed that SVM is not very suitable for low latency systems where it consumed lot of time for training as well as testing the data model.

REFERENCE :-

https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a

https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34

https://gdcoder.com/support-vector-machine-vs-logistic-regression/