TiisetsoPROJECT4

October 16, 2023

1 Tiisetso Khasebe

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
tweet = pd.read_csv('/kaggle/input/twitter-hate-speech/TwitterHate.
 ↪csv',delimiter=',',engine='python',encoding='utf-8-sig')
tweet.head()
```

[2]:

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s…
1	2	0	@user @user thanks for #lyft credit i can't us…
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in …
4	5	0	factsguide: society now #motivation

[3]:
```
tweet.drop('id',axis=1,inplace=True)
```

[4]:
```
random = np.random.randint(0,len(tweet))
print(random)
tweet.iloc[random]['tweet']
```

27706

[4]:
```
"it won't david. the business world suppos narcissists and abusers are
attractive to many on a cry wick level  "
```

[5]:
```
data = tweet.copy()
```

Text Cleaning

[6]:
```
def simplify(text):
    '''Function to handle the diacritics in the text'''
    import unicodedata
    try:
        text = unicode(text, 'utf-8')
    except NameError:
```

```
        pass
    text = unicodedata.normalize('NFD', text).encode('ascii', 'ignore').
↪decode("utf-8")
    return str(text)
```

[7]:
```
data['tweet'] = data['tweet'].apply(simplify)
```

Remove user handles

[8]:
```
sample = "and @user1 i would like you to discuss with @user2 and then with␣
↪@username3"
pattern = re.compile(r'@\w+')
re.findall(pattern,sample)
```

[8]:
```
['@user1', '@user2', '@username3']
```

[9]:
```
data['tweet'].replace(r'@\w+','',regex=True,inplace=True)
```

[10]:
```
sample = "https://www.machinelearing.com prakhar and https://www.simple.com"
pattern = re.compile(r'http\S+')
re.findall(pattern,sample)
```

[10]:
```
['https://www.machinelearing.com', 'https://www.simple.com']
```

[11]:
```
data['tweet'].replace(r'http\S+','',regex=True,inplace=True)
```

Tokenize using tweet tokenizer

[12]:
```
sample = 'wonderfl :-)  when are you coming for #party'
tweet_tokenize = TweetTokenizer(preserve_case=True)
tweet_tokenize.tokenize(sample)
```

[12]:
```
['wonderfl', ':-)', 'when', 'are', 'you', 'coming', 'for', '#party']
```

[13]:
```
tokenizer = TweetTokenizer(preserve_case=True)
data['tweet'] = df['tweet'].apply(tokenizer.tokenize)
```

[14]:
```
data.head(3)
```

[14]:

	label	tweet
0	0	[when, a, father, is, dysfunctional, and, is, …
1	0	[thanks, for, #lyft, credit, i, can't, use, ca…
2	0	[bihday, your, majesty]

[15]:
```
stop_words = stopwords.words('english')

additional_list = ['amp','rt','u',"can't",'ur']
```

```
for words in additional_list:
    stop_words.append(words)
```

[16]:
```
stop_words[-10:]
```

[16]:
```
["weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't",
 'amp',
 'rt',
 'u',
 "can't",
 'ur']
```

[17]:
```
def remove_stopwords(text):
    '''Function to remove the stop words from the text corpus'''
    clean_text = [word for word in text if not word in stop_words]
    return clean_text
```

[18]:
```
data['tweet'] = data['tweet'].apply(remove_stopwords)
```

[19]:
```
data['tweet'].head()
```

[19]:
```
0    [father, dysfunctional, selfish, drags, kids, …
1    [thanks, #lyft, credit, use, cause, offer, whe…
2                                    [bihday, majesty]
3                  [#model, love, take, time, !, !, !]
4                [factsguide, :, society, #motivation]
Name: tweet, dtype: object
```

Spelling corrections

[20]:
```
from textblob import TextBlob
sample = 'amazng man you did it finallyy'
txtblob = TextBlob(sample)
corrected_text = txtblob.correct()
print(corrected_text)
```

```
amazing man you did it finally
```

[21]:
```
from textblob import TextBlob

def spell_check(text):
    '''Function to do spelling correction using '''
```

```
    txtblob = TextBlob(text)
    corrected_text = txtblob.correct()
    return corrected_text
```

Remove # symbols

[22]:
```
sample = '#winner #machine i am learning'
pattern = re.compile(r'#')
re.sub(pattern,'',sample)
```

[22]:
```
'winner machine i am learning'
```

[23]:
```
def remove_hashsymbols(text):
    '''Function to remove the hashtag symbol from the text'''
    pattern = re.compile(r'#')
    text = ' '.join(text)
    clean_text = re.sub(pattern,'',text)
    return tokenizer.tokenize(clean_text)
```

[24]:
```
data['tweet'] = data['tweet'].apply(remove_hashsymbols)
```

[25]:
```
data.head(3)
```

[25]:

	label	tweet
0	0	[father, dysfunctional, selfish, drags, kids, …
1	0	[thanks, lyft, credit, use, cause, offer, whee…
2	0	[bihday, majesty]

Remove single and double length characters

[26]:
```
def rem_shortwords(text):
    '''Function to remove the short words of length 1 and 2 characters'''
    '''Arguments:
       text: string
       returns: string without containing words of length 1 and 2'''
    lengths = [1,2]
    new_text = ' '.join(text)
    for word in text:
        text = [word for word in tokenizer.tokenize(new_text) if not len(word)␣
 ↪in lengths]

    return new_text
```

[27]:
```
data['tweet'] = data['tweet'].apply(rem_shortwords)
```

[28]:
```
data.head(2)
```

[28]:
```
   label                                              tweet
0      0  father dysfunctional selfish drags kids dysfun…
1      0  thanks lyft credit use cause offer wheelchair …
```

[29]:
```
data['tweet'] = data['tweet'].apply(tokenizer.tokenize)
```

[30]:
```
data.head(3)
```

[30]:
```
   label                                              tweet
0      0  [father, dysfunctional, selfish, drags, kids, …
1      0  [thanks, lyft, credit, use, cause, offer, whee…
2      0                                  [bihday, majesty]
```

Remove digits

[31]:
```
def rem_digits(text):
    '''Function to remove the digits from the list of strings'''
    no_digits = []
    for word in text:
        no_digits.append(re.sub(r'\d','',word))
    return ' '.join(no_digits)
```

[32]:
```
data['tweet'] = data['tweet'].apply(rem_digits)
```

[33]:
```
data['tweet'] = data['tweet'].apply(tokenizer.tokenize)
```

[34]:
```
data.head()
```

[34]:
```
   label                                              tweet
0      0  [father, dysfunctional, selfish, drags, kids, …
1      0  [thanks, lyft, credit, use, cause, offer, whee…
2      0                                  [bihday, majesty]
3      0                [model, love, take, time, !, !, !]
4      0              [factsguide, :, society, motivation]
```

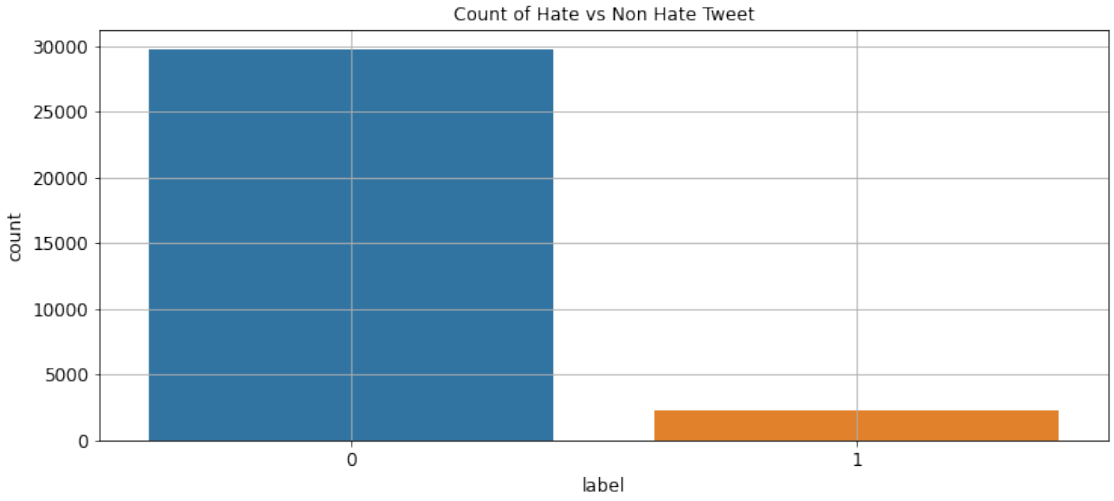
Remove special characters

[35]:
```
def rem_nonalpha(text):
    '''Function to remove the non-alphanumeric characters from the text'''
    text = [word for word in text if word.isalpha()]
    return text
```

[36]:
```
data['tweet'] = data['tweet'].apply(rem_nonalpha)
```

Check for data balance

[37]:
```
sns.countplot(data['label'])
plt.title('Count of Hate vs Non Hate Tweet')
plt.grid()
plt.show()
```

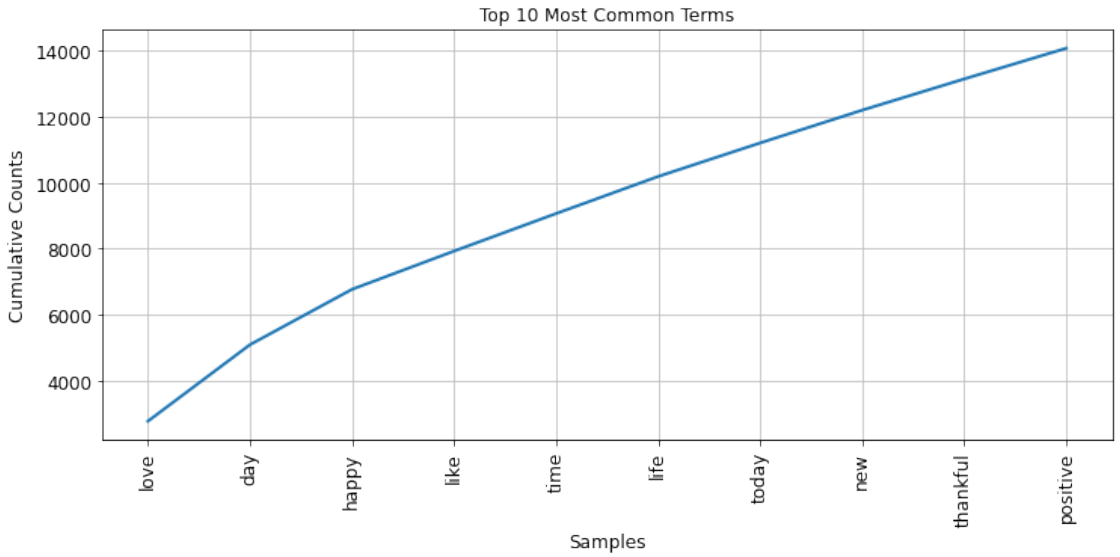


Check the top terms in the tweets

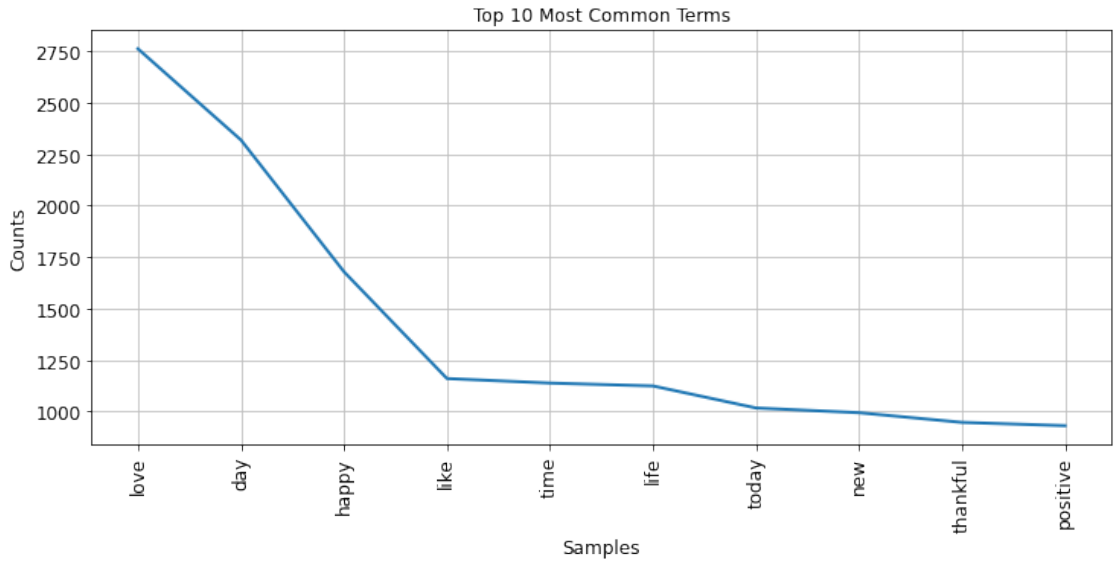
[38]:
```
from collections import Counter
results = Counter()
data['tweet'].apply(results.update)
print(results.most_common(10))
```

```
[('love', 2762), ('day', 2319), ('happy', 1679), ('like', 1160), ('time', 1138),
('life', 1124), ('today', 1017), ('new', 994), ('thankful', 947), ('positive',
931)]
```

[39]:
```
frequency = nltk.FreqDist(results)
plt.title('Top 10 Most Common Terms')
frequency.plot(10,cumulative=True)
plt.show()
```



[40]:
```
frequency = nltk.FreqDist(results)
plt.title('Top 10 Most Common Terms')
frequency.plot(10,cumulative=False)
plt.show()
```



1.0.1 *Data Formatting for Predictive Modeling*

[41]:
```
data.head()
```

[41]:
```
   label                                              tweet
0      0  [father, dysfunctional, selfish, drags, kids, …
1      0  [thanks, lyft, credit, use, cause, offer, whee…
2      0                                  [bihday, majesty]
3      0                           [model, love, take, time]
4      0                 [factsguide, society, motivation]
```

[42]:
```
data.isnull().sum()
```

[42]:
```
label    0
tweet    0
dtype: int64
```

[43]:
```
data['tweet'] = data['tweet'].apply(lambda x: ' '.join(x))
```

[44]:
```
data.head(3)
```

[44]:
```
   label                                              tweet
0      0  father dysfunctional selfish drags kids dysfun…
1      0  thanks lyft credit use cause offer wheelchair …
2      0                                    bihday majesty
```

[45]:
```
X = data['tweet']
y = data['label']
```

[46]:
```
from sklearn.model_selection import train_test_split
seed = 51
test_size = 0.2
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=seed,stratify=data['label'])
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(25569,) (6393,) (25569,) (6393,)
```

[47]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

[48]:
```
vectorizer = TfidfVectorizer(max_features=5000)
```

[49]:
```
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

[50]:
```
X_train.shape, X_test.shape
```

[50]:
```
((25569, 5000), (6393, 5000))
```

Model building

[51]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
```

[52]:
```
clf = LogisticRegression()
clf.fit(X_train,y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
```

Model evaluation

[53]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

[54]:
```
print('Accuracy Score on training set %.5f'␣
  ↪%accuracy_score(y_train,train_predictions))
print('Accuracy Score on test set %.5f'␣
  ↪%accuracy_score(y_test,test_predictions))
```

```
Accuracy Score on training set 0.95569
Accuracy Score on test set 0.94791
```

[55]:
```
print('Classification Report Training set')
print('\n')
print(classification_report(y_train,train_predictions))
```

```
Classification Report Training set


              precision    recall  f1-score   support

           0       0.96      1.00      0.98     23775
           1       0.95      0.39      0.55      1794

    accuracy                           0.96     25569
   macro avg       0.95      0.69      0.76     25569
weighted avg       0.96      0.96      0.95     25569
```

[56]:
```
print('Classification Report Testing set')
print('\n')
print(classification_report(y_test,test_predictions))
```

```
Classification Report Testing set


              precision    recall  f1-score   support
```

```
              0       0.95      1.00      0.97      5945
              1       0.90      0.29      0.44       448

       accuracy                           0.95      6393
      macro avg       0.93      0.64      0.70      6393
   weighted avg       0.95      0.95      0.94      6393
```

Weighted Logistic Regression Or Cost Sensitive Logistic Regression

[57]:
```
data['label'].value_counts()
```

[57]:
```
0    29720
1     2242
Name: label, dtype: int64
```

[58]:
```
weights = {0:1.0,1:13
data.clf = LogisticRegression(solver='lbfgs',class_weight=weights)
clf.fit(X_train,y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
print('Classification Report Training set')
print('-------------------------------------')
print('\n')
print(classification_report(y_train,train_predictions))
print('\n')
print('Classification Report Testing set')
print('-------------------------------------')
print('\n')
print(classification_report(y_test,test_predictions))
```

```
Classification Report Training set
-------------------------------------


              precision    recall  f1-score   support

           0       1.00      0.95      0.97     23775
           1       0.60      0.98      0.74      1794

    accuracy                           0.95     25569
   macro avg       0.80      0.96      0.86     25569
weighted avg       0.97      0.95      0.96     25569



Classification Report Testing set
```

```
-----------------------------------
```

	precision	recall	f1-score	support
0	0.98	0.94	0.96	5945
1	0.48	0.75	0.58	448
accuracy			0.92	6393
macro avg	0.73	0.84	0.77	6393
weighted avg	0.94	0.92	0.93	6393

Regularization and Hyperparameter tuning

[59]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

[60]:
```
from scipy.stats import loguniform
space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 100)
```

[61]:
```
print(space)
```

```
{'solver': ['newton-cg', 'lbfgs', 'liblinear'], 'penalty': ['l1', 'l2',
'elasticnet'], 'C': <scipy.stats._distn_infrastructure.rv_frozen object at
0x7f17226ea090>}
```

Fine tuned Model with Balanced Class Weights

[62]:
```
weights = {0:1.0,1:1.0}
clf = LogisticRegression(class_weight=weights)
folds = StratifiedKFold(n_splits=4,random_state=seed)
grid_search = RandomizedSearchCV(estimator=clf,param_distributions=space,␣
 ↪n_iter=100, scoring='recall',
                            n_jobs=-1, cv=folds, random_state=seed)
grid_result = grid_search.fit(X_train,y_train)
```

[63]:
```
grid_result.best_estimator_
```

[63]:
```
LogisticRegression(C=23.871926754399514, class_weight={0: 1.0, 1: 1.0},
                   penalty='l1', solver='liblinear')
```

[64]:
```
clf = LogisticRegression(C=23.
 ↪871926754399514,penalty='l1',solver='liblinear',class_weight=weights)
```

[65]:
```
clf.fit(X_train,y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
print('Classification Report Training set')
print('-----------------------------------')
print('\n')
print(classification_report(y_train,train_predictions))
print('\n')
print('Classification Report Testing set')
print('-----------------------------------')
print('\n')
print(classification_report(y_test,test_predictions))
```

```
Classification Report Training set
-----------------------------------


              precision    recall  f1-score   support

           0       0.99      1.00      1.00     23775
           1       0.98      0.93      0.95      1794

    accuracy                           0.99     25569
   macro avg       0.99      0.96      0.97     25569
weighted avg       0.99      0.99      0.99     25569



Classification Report Testing set
-----------------------------------


              precision    recall  f1-score   support

           0       0.97      0.97      0.97      5945
           1       0.62      0.56      0.59       448

    accuracy                           0.94      6393
   macro avg       0.79      0.77      0.78      6393
weighted avg       0.94      0.94      0.94      6393
```

Fine tuned model with class weights proportional to the class imbalance

[66]:
```
weights = {0:1.0,1:13}

clf = LogisticRegression(class_weight=weights)
folds = StratifiedKFold(n_splits=4,random_state=seed)
```

```
grid_search = RandomizedSearchCV(estimator=clf,param_distributions=space,␣
 ↪n_iter=100, scoring='recall',
                            n_jobs=-1, cv=folds, random_state=seed)
grid_result = grid_search.fit(X_train,y_train)

grid_result.best_estimator_
```

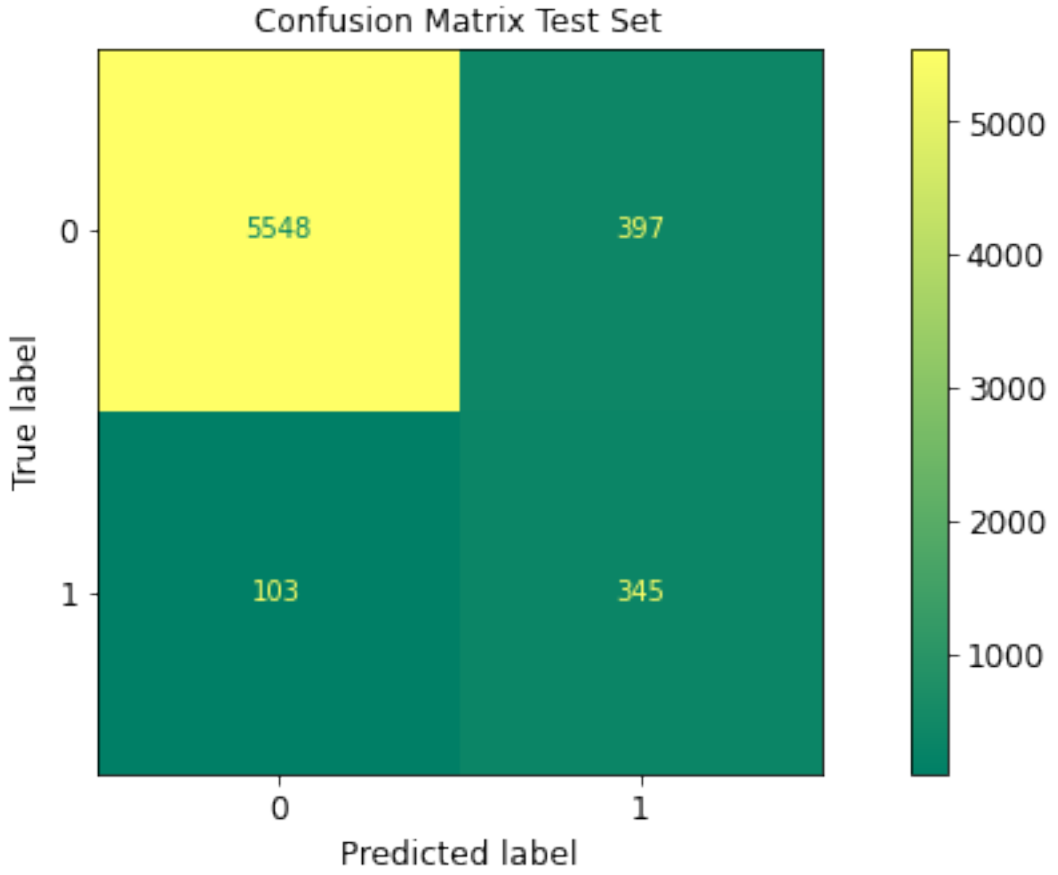
[66]:
```
LogisticRegression(C=0.16731783677034165, class_weight={0: 1.0, 1: 13},
                   solver='liblinear')
```

[67]:
```
clf = LogisticRegression(C=0.
 ↪16731783677034165,penalty='l2',solver='liblinear',class_weight=weights)
clf.fit(X_train,y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
print('Classification Report Training set')
print('-------------------------------------')
print('\n')
print(classification_report(y_train,train_predictions))
print('\n')
print('Classification Report Testing set')
print('-------------------------------------')
print('\n')
print(classification_report(y_test,test_predictions))
```

```
Classification Report Training set
-------------------------------------


              precision    recall  f1-score   support

           0       0.99      0.94      0.97     23775
           1       0.53      0.93      0.68      1794

    accuracy                           0.94     25569
   macro avg       0.76      0.93      0.82     25569
weighted avg       0.96      0.94      0.95     25569



Classification Report Testing set
-------------------------------------


              precision    recall  f1-score   support

           0       0.98      0.93      0.96      5945
```

```
              1       0.46      0.77      0.58       448

    accuracy                           0.92      6393
   macro avg       0.72      0.85      0.77      6393
weighted avg       0.95      0.92      0.93      6393
```

[68]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf,X_test,y_test,cmap='summer')
plt.title('Confusion Matrix Test Set')
plt.show()
```



[ ]: