```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Mounted at /gdrive
/gdrive
```

```
ls
```

```
MyDrive/  Shareddrives/
```

```
cd /gdrive/My Drive/Mood Analysis
```

```
/gdrive/My Drive/Mood Analysis
```

```
ls
```

```
test.txt  train.txt  val.txt
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from pandas.plotting import scatter_matrix
warnings.filterwarnings('ignore')
%matplotlib inline
import os
```

```
train_data = pd.read_csv("/gdrive/My Drive/Mood Analysis/train.txt", sep=';',names=['text','e
test_data = pd.read_csv("/gdrive/My Drive/Mood Analysis/test.txt", sep=';',names=['text','emo
validation_data = pd.read_csv("/gdrive/My Drive/Mood Analysis/val.txt", sep=';',names=['text'
```

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16000 entries, 0 to 15999
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   text     16000 non-null  object
 1   emotion  16000 non-null  object
dtypes: object(2)
memory usage: 250.1+ KB
```

```
train_data.describe()
```

	text	emotion
count	16000	16000
unique	15969	6
top	i have chose for myself that makes me feel ama...	joy
freq	2	5362

```
list(train_data.emotion.unique())
```

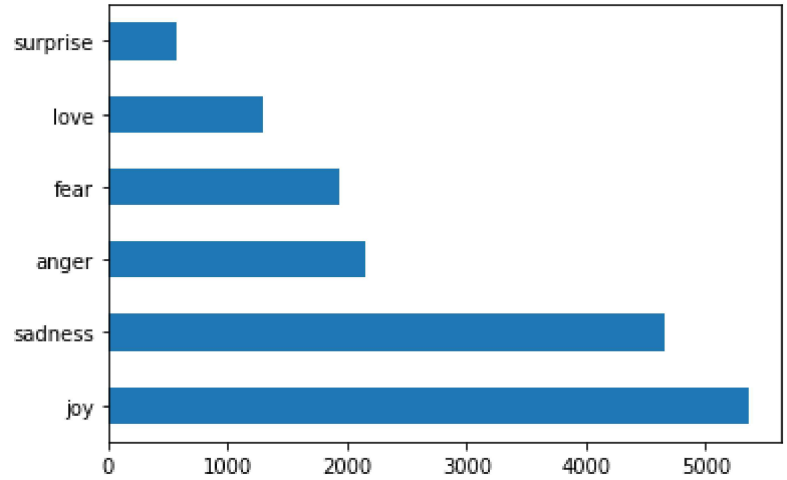
```
['sadness', 'anger', 'love', 'surprise', 'fear', 'joy']
```

```
train_data.head(10)
```

	text	emotion
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
6	ive been taking or milligrams or times recomme...	surprise
7	i feel as confused about life as a teenager or...	fear
8	i have been with petronas for years i feel tha...	joy
9	i feel romantic too	love

```
train_data['emotion'].value_counts().plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe490534a58>
```



```
train_data['emotion'].value_counts().plot(kind='pie')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe485dfa5c0>
```


```
from nltk.corpus import wordnet

def get_wordnet_pos(pos_tag):
    if pos_tag.startswith('J'):
        return wordnet.ADJ
    elif pos_tag.startswith('V'):
        return wordnet.VERB
    elif pos_tag.startswith('N'):
        return wordnet.NOUN
    elif pos_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

import string
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.tokenize import WhitespaceTokenizer
from nltk.stem import WordNetLemmatizer

def clean_text(text):
    # lower text
    text = text.lower()
    # tokenize text and remove puncutation
    text = [word.strip(string.punctuation) for word in text.split(" ")]
    # remove words that contain numbers
    text = [word for word in text if not any(c.isdigit() for c in word)]
    # remove stop words
    stop = stopwords.words('english')
    text = [x for x in text if x not in stop]
    # remove empty tokens
    text = [t for t in text if len(t) > 0]
    # pos tag text
    pos_tags = pos_tag(text)
```

```
    # lemmatize text
    text = [WordNetLemmatizer().lemmatize(t[0], get_wordnet_pos(t[1])) for t in pos_tags]
    # remove words with only one letter
    text = [t for t in text if len(t) > 1]
    # join all
    text = " ".join(text)
    return(text)


import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
train_data["text"] = train_data["text"].apply(lambda x: clean_text(x))
test_data["text"] = test_data["text"].apply(lambda x: clean_text(x))
train_data.head()
```

```
    [nltk_data] Downloading package averaged_perceptron_tagger to
    [nltk_data]     /root/nltk_data...
    [nltk_data]   Package averaged_perceptron_tagger is already up-to-
    [nltk_data]       date!
    [nltk_data] Downloading package wordnet to /root/nltk_data...
    [nltk_data]   Unzipping corpora/wordnet.zip.
```

	text	emotion
0	didnt feel humiliate	sadness
1	go feeling hopeless damn hopeful around someon...	sadness
2	im grab minute post feel greedy wrong	anger
3	ever feel nostalgic fireplace know still property	love
4	feel grouchy	anger

```
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
train_data["sentiments"] = train_data["text"].apply(lambda x: sid.polarity_scores(x))
train_data = pd.concat([train_data.drop(['sentiments'], axis=1), train_data['sentiments'].app
train_data.head()
sid = SentimentIntensityAnalyzer()
test_data["sentiments"] = test_data["text"].apply(lambda x: sid.polarity_scores(x))
test_data = pd.concat([test_data.drop(['sentiments'], axis=1), test_data['sentiments'].apply(
```

```
    [nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

```
test_data.shape
```

```
    (2000, 6)
```

```
train_data.head()
```

	text	emotion	neg	neu	pos	compound
0	didnt feel humiliate	sadness	0.000	0.412	0.588	0.4310
1	go feeling hopeless damn hopeful around someon...	sadness	0.322	0.226	0.452	0.3182
2	im grab minute post feel greedy wrong	anger	0.519	0.481	0.000	-0.6597
3	ever feel nostalgic fireplace know still property	love	0.000	1.000	0.000	0.0000
4	feel grouchy	anger	0.744	0.256	0.000	-0.4404

```
joy_text = train_data[train_data["emotion"] == 'joy']["text"].values
for i in range(0,5):
    print(joy_text[i], "\n")
```

```
 i have been with petronas for years i feel that petronas has performed well and made a h

 i do feel that running is a divine experience and that i can expect to have some type of

 i have immense sympathy with the general point but as a possible proto writer trying to

 i do not feel reassured anxiety is on each side

 i have the feeling she was amused and delighted
```

```
sadness_text = train_data[train_data["emotion"] == 'sadness']["text"].values
for i in range(0,5):
    print(sadness_text[i], "\n")
```

```
 i didnt feel humiliated

 i can go from feeling so hopeless to so damned hopeful just from being around someone wh

 ive been feeling a little burdened lately wasnt sure why that was

 i feel like i have to make the suffering i m seeing mean something

 i feel low energy i m just thirsty
```

```
anger_text = train_data[train_data["emotion"] == 'anger']["text"].values
for i in range(0,5):
    print(anger_text[i], "\n")
```

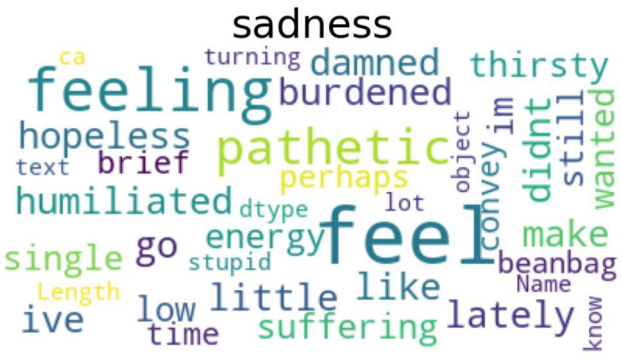
```
 im grabbing a minute to post i feel greedy wrong

 i am feeling grouchy

 i think it s the easiest time of year to feel dissatisfied
```

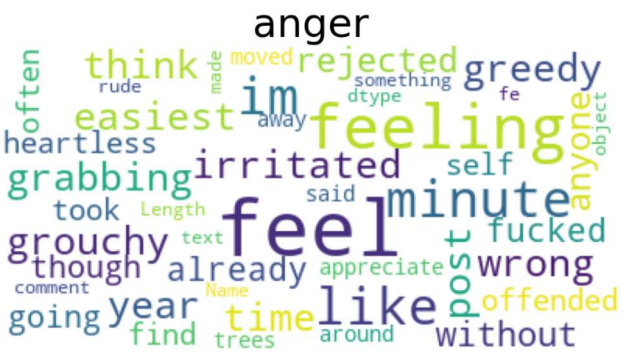
```
i feel irritated and rejected without anyone doing anything or saying anything

i already feel like i fucked up though because i dont usually eat at all in the morning
```

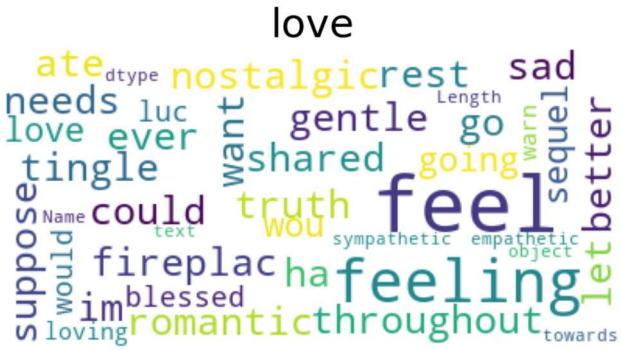
```
import nltk
from wordcloud import WordCloud
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

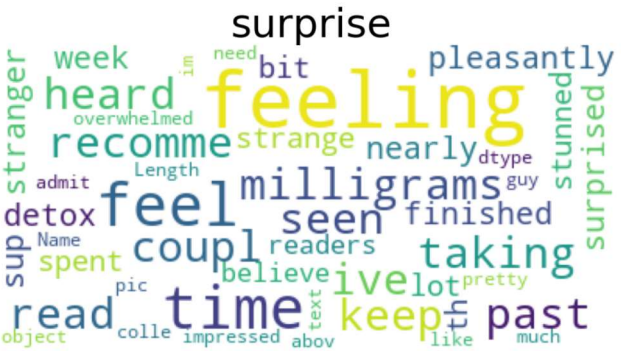
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
def get_wordcloud(text):
    wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white", stopword

    return wordcloud;


emotions = train_data["emotion"].unique()

figure, axes = plt.subplots(ncols=2, nrows=3,figsize=(30,25))
plt.axis('off')

# for each emotion
for emotion, ax in zip(emotions, axes.flat):
    wordcloud = get_wordcloud(train_data[train_data["emotion"]==emotion]['text'])
    ax.imshow(wordcloud)
    ax.title.set_text(emotion)
    ax.title.set_size(50)

    ax.axis('off')

plt.subplots_adjust(wspace=0.15, hspace=0.05)
```










```
from sklearn.metrics import make_scorer
from sklearn.metrics import fbeta_score
import spacy
from sklearn import preprocessing

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

```
import pickle


labelEncoder = preprocessing.LabelEncoder()
labelEncoder.fit(train_data.emotion)

# Train data
train_data.emotion = labelEncoder.transform(train_data.emotion)


# Validation data
# validation_data.emotion = labelEncoder.transform(validation_data.emotion)

# Test data
test_data.emotion = labelEncoder.transform(test_data.emotion)


list(train_data.emotion.unique())
```

```
[4, 0, 3, 5, 1, 2]
```

```
test_data.shape
```

```
(2000, 6)
```

```
train_data.head()
```

	text	emotion	neg	neu	pos	compound
0	didnt feel humiliate	4	0.000	0.412	0.588	0.4310
1	go feeling hopeless damn hopeful around someon...	4	0.322	0.226	0.452	0.3182
2	im grab minute post feel greedy wrong	0	0.519	0.481	0.000	-0.6597
3	ever feel nostalgic fireplace know still property	3	0.000	1.000	0.000	0.0000
4	feel grouchy	0	0.744	0.256	0.000	-0.4404

```
import spacy


import spacy.cli
spacy.cli.download("en_core_web_lg")
```

```
✓ Download and installation successful
You can now load the model via spacy.load('en_core_web_lg')
```

```
nlp = spacy.load('en_core_web_lg')


def vectorize(dataset):
```

```
    """
    Get vectors for data in parameter
    ================================

    Parameters:
        data:
            Dataset
    """
    with nlp.disable_pipes():
        vectors = np.array([nlp(data.text).vector for index, data in dataset.iterrows()])
        return vectors


from gensim.test.utils import common_texts
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(train_data["text"].apply(lambda

# train a Doc2Vec model with train data
model = Doc2Vec(documents, vector_size=5, window=2, min_count=1, workers=4)

# transform each document into a vector data
# Train data
doc2vec_df = train_data["text"].apply(lambda x: model.infer_vector(x.split(" "))).apply(pd.Se
doc2vec_df.columns = ["doc2vec_vector_" + str(x) for x in doc2vec_df.columns]
train_data = pd.concat([train_data, doc2vec_df], axis=1)

train_data.head()

# Test data
documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(test_data["text"].apply(lambda
model = Doc2Vec(documents, vector_size=5, window=2, min_count=1, workers=4)

doc2vec_df = test_data["text"].apply(lambda x: model.infer_vector(x.split(" "))).apply(pd.Ser
doc2vec_df.columns = ["doc2vec_vector_" + str(x) for x in doc2vec_df.columns]
test_data = pd.concat([test_data, doc2vec_df], axis=1)


text_column_data = train_data.drop(train_data.columns.difference(['text']), 1, inplace=False)
train_data_text_vectors = vectorize(text_column_data)

text_column_data = test_data.drop(test_data.columns.difference(['text']), 1, inplace=False)
test_data_text_vectors = vectorize(text_column_data)


text_column_data.head()
```

	text
0	im feel rather rotten im ambitious right
1	im update blog feel shitty
4	feel little vain one

```
test_data.shape
```

```
(2000, 11)
```

```
test_data_text_vectors.shape
```

```
(2000, 300)
```

```
train_data_text_vectors.shape
```

```
(16000, 300)
```

```
type(train_data_text_vectors)
```

```
numpy.ndarray
```

```
train_data.head()
```

	text	emotion	neg	neu	pos	compound	doc2vec_vector_0	doc2vec_vector_1
0	didnt feel humiliate	4	0.000	0.412	0.588	0.4310	-0.083021	-0.041404
1	go feeling hopeless damn hopeful around someon...	4	0.322	0.226	0.452	0.3182	-0.035258	0.043801

```
X_train = pd.concat([pd.DataFrame(train_data_text_vectors), train_data.drop(["emotion", "text
y_train = train_data.emotion

print("X_train shape ", X_train.shape)
print("y_train shape ", y_train.shape)

# Test set
X_test = pd.concat([pd.DataFrame(test_data_text_vectors), test_data.drop(["emotion", "text"],
y_test = test_data.emotion
```

```
X_train shape  (16000, 309)
y_train shape  (16000,)
```

```
X_test.shape
```

```
(2000, 309)
```

```
X_test.head()
```

	0	1	2	3	4	5	6	7	
0	0.063198	0.211076	-0.358384	-0.126282	0.068429	0.166817	0.184461	-0.227866	0.0
1	-0.109183	0.291150	-0.392512	-0.022207	-0.092217	0.207853	0.230753	-0.503302	0.0
2	-0.042222	0.089083	-0.140859	-0.095002	0.042288	0.032222	0.149373	-0.163829	-0.1
3	-0.137298	0.171552	-0.263142	-0.035255	0.158753	-0.128415	-0.187472	0.092865	-0.0
4	-0.123115	0.070863	-0.133397	-0.015207	0.067200	-0.147063	0.074698	-0.088150	-0.1

5 rows × 309 columns

```
X_train.head()
```

	0	1	2	3	4	5	6	7	
0	-0.307954	0.122157	-0.249985	-0.243244	-0.125813	-0.084080	-0.015906	-0.208288	-0.1
1	0.013919	0.161146	-0.241964	0.021802	0.009294	-0.157627	0.022174	-0.063920	0.1
2	-0.139047	0.162944	-0.207012	0.082678	-0.015077	0.153544	0.164179	-0.218857	0.0
3	0.230904	0.141165	-0.151757	-0.165659	0.039434	0.019943	0.304757	-0.287210	-0.1
4	-0.043325	0.257403	-0.399145	-0.110763	-0.151330	0.484480	0.305390	-0.260290	-0.0

5 rows × 309 columns

```
def checkPerformances(classifier, best_clf):
    """
    Check model performance with unoptimized
    and optimized same model for comparison
    =======================================

    Parameters:
        classifier:
            Basic unoptimized classifier model
        best_clf:
            Optimized classifier model
    """
    # Make predictions using the unoptimized and optimized and model
    predictions = (classifier.fit(X_train, y_train)).predict(X_test)
    best_predictions = best_clf.predict(X_test)
```

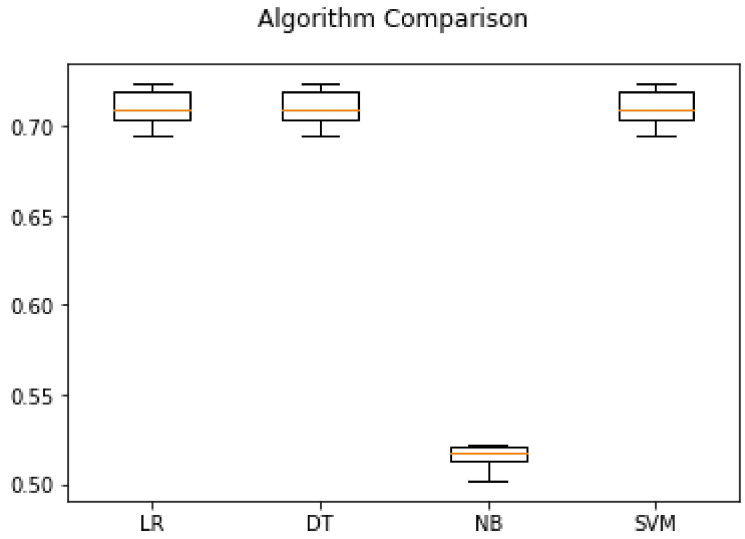
```
    # Report the before-and-afterscores
    print("Unoptimized model\n------")
    print(classifier)
    print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)
    print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.
    print("\nOptimized Model\n------")
    print(best_clf)
    print("\nFinal accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test,
    print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predict


scorer = make_scorer(fbeta_score, beta=0.5, average="micro")


import os.path
from os import path


best_LR_clf_filepath = "/gdrive/My Drive/Mood Analysis.pkl"

LR_clf = LogisticRegression(random_state = 0)
if not path.exists(best_LR_clf_filepath):
    print("Performing GridSearch for LR")
    LR_parameters = {
        'penalty': ['l1', 'l2'],
        'C': np.logspace(0, 4, 10)
        }

    # Run the grid search
    grid_obj = GridSearchCV(LR_clf, LR_parameters, scoring=scorer)
    grid_obj = grid_obj.fit(X_train, y_train)

    # Set the regressor to the best combination of parameters
    best_LR_clf = grid_obj.best_estimator_

    best_LR_clf

    # save the model
    pickle.dump(best_LR_clf, open("/gdrive/My Drive/Mood Analysis.pkl", 'wb'))
else:
    print("Best LR file loading")
    best_LR_clf = pickle.load(open(best_LR_clf_filepath, 'rb'))

# check performances
checkPerformances(LR_clf, best_LR_clf)
```

```
 Performing GridSearch for LR
 Unoptimized model
 ------
 LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
```

```
                        warm_start=False)
      Accuracy score on testing data: 0.7145
      F-score on testing data: 0.7145

      Optimized Model
      ------
      LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                         intercept_scaling=1, l1_ratio=None, max_iter=100,
                         multi_class='auto', n_jobs=None, penalty='l2',
                         random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                         warm_start=False)

      Final accuracy score on the testing data: 0.7145
      Final F-score on the testing data: 0.7145


best_SVM_clf_filepath = "/gdrive/My Drive/Mood Analysis.pkl"
SVM_clf = SVC()
if not path.exists(best_SVM_clf_filepath):
    print("Performing GridSearch for SVM")
    SVM_parameters = {
        'kernel': ['linear', 'poly', 'rbf'],
        'degree': [1, 2, 3, 4],
        'shrinking' : [True, False]
        }

    # Run the grid search
    grid_obj = GridSearchCV(SVM_clf, SVM_parameters, scoring='accuracy')
    grid_obj = grid_obj.fit(X_train, y_train)

    # Set the regressor to the best combination of parameters
    best_SVM_clf = grid_obj.best_estimator_

    # save the model
    pickle.dump(best_SVM_clf, open("/gdrive/My Drive/Mood Analysis.pkl", 'wb'))
else:
    print("Best SVM file loading")
    best_SVM_clf = pickle.load(open(best_SVM_clf_filepath, 'rb'))

# check performances
checkPerformances(SVM_clf, best_SVM_clf)

      Best SVM file loading
      Unoptimized model
      ------
      SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
      Accuracy score on testing data: 0.7125
      F-score on testing data: 0.7125

      Optimized Model
      ------
```

```
      LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                         intercept_scaling=1, l1_ratio=None, max_iter=100,
                         multi_class='auto', n_jobs=None, penalty='l2',
                         random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                         warm_start=False)

      Final accuracy score on the testing data: 0.7145
      Final F-score on the testing data: 0.7145


best_DT_clf_filepath = "/gdrive/My Drive/Mood Analysis.pkl"

DT_clf = DecisionTreeClassifier()

if not path.exists(best_DT_clf_filepath):
    print("Performing GridSearch for DT")
    DT_parameters = {
        'criterion': ['gini', 'entropy'],
        'min_samples_leaf': [1, 2, 3, 4]
        }

    # Run the grid search
    grid_obj = GridSearchCV(DT_clf, DT_parameters, scoring=scorer)
    grid_obj = grid_obj.fit(X_train, y_train)

    # Set the regressor to the best combination of parameters
    best_DT_clf = grid_obj.best_estimator_

    # save the model
    pickle.dump(best_DT_clf, open("/gdrive/My Drive/Mood Analysis.pkl", 'wb'))
else:
    print("Best DT file loading")
    best_DT_clf = pickle.load(open(best_DT_clf_filepath, 'rb'))

checkPerformances(DT_clf, best_DT_clf)

     Best DT file loading
     Unoptimized model
     ------
     DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                            max_depth=None, max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort='deprecated',
                            random_state=None, splitter='best')
     Accuracy score on testing data: 0.4390
     F-score on testing data: 0.4390

     Optimized Model
     ------
     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
```

```
                           warm_start=False)

     Final accuracy score on the testing data: 0.7145
     Final F-score on the testing data: 0.7145


NB_clf = GaussianNB()
NB_clf.fit(X_train, y_train)
predictions = (NB_clf.fit(X_train, y_train)).predict(X_test)

print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5, a

     Accuracy score on testing data: 0.5180
     F-score on testing data: 0.5180


from sklearn import model_selection
# 10-fold cross validation
# Test options and evaluation metric
seed = 7
# Using metric accuracy to measure performance
scoring = 'accuracy'


models = []
models.append(('LR', best_LR_clf))
models.append(('DT', best_DT_clf))
models.append(('NB', NB_clf))
models.append(('SVM', best_SVM_clf))

# evaluate each model in turn
results = []
names = []
for name, model in models:
  kfold = model_selection.KFold(n_splits=10, random_state=seed)
  cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=sco
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)

     LR: 0.710562 (0.009331)
     DT: 0.710562 (0.009331)
     NB: 0.515812 (0.006219)
     SVM: 0.710562 (0.009331)


fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```




```
from sklearn.metrics import confusion_matrix


def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
                  see http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion matrix created by
                                                              # sklearn.metrics.confusion_mat
                          normalize    = True,                # show proportions
                          target_names = y_labels_vals,       # list of names of the classes
                          title        = best_estimator_name) # title of graph
```

```
    Citiation
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclas
    plt.show()


y_pred = (best_SVM_clf.fit(X_train, y_train)).predict(X_test)
cm = confusion_matrix(y_test, y_pred)
targets = labelEncoder.inverse_transform([0,1,2,3,4,5])

plot_confusion_matrix(cm, target_names=targets, title="Confusion Matrix for SVM")
```

```
plot_confusion_matrix(cm, target_names=targets, title='Confusion Matrix for SVM')
```