In [1]:
```
#import warnings
#warnings.filterwarnings('ignore')

#sklearn models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, \
        GradientBoostingClassifier, StackingClassifier, BaggingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier

#sklearn metrics
from sklearn.metrics import roc_curve, auc, average_precision_score, precision_recall_curve
#DOES NOT WORK: , plot_precision_recall_curve
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import balanced_accuracy_score, precision_score, recall_score, f1_score, roc_auc_
score

#sklearn model selection
from sklearn.model_selection import cross_val_score, cross_validate, train_test_split, cross_val_predi
ct

#scikit-plot
from scikitplot.metrics import plot_precision_recall, plot_roc, plot_confusion_matrix
from scikitplot.estimators import plot_learning_curve

#mlxtend
from mlxtend.plotting import plot_learning_curves, plot_decision_regions


#hyperparameter tuning

#miscellaneous
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#some "global" variables defined here
random_state_num = 0
cv_num = 10
classes_num = 2 #there are two classes, non-negative (0) and negative (1)
fig_size_tuple = (15,7)
title_fontsize_num = 15
label_fontsize_num = 12
```

Read Trained Word2Vec Embedding Vectors and Engineered Features

We read in the engineered features dataset as a dataframe.

In [2]:
```
#read the csv of engineered features
#then split the columns into
df_gensim_word2vec_features = pd.read_csv('..\data\gensim_word2vec_trained_with_engineered_features.cs
v')
X = df_gensim_word2vec_features.loc[:, df_gensim_word2vec_features.columns != 'binary_response_variabl
e']
Y= df_gensim_word2vec_features.loc[:, df_gensim_word2vec_features.columns == 'binary_response_variabl
e']
```

Split into Train and Test dataset

The dataset is then split into a train and test dataset.

In [3]:
```
#Split into training set, and test set
# we do not need a validation set because we will be doing k-fold cross validation

#split into 0.7 training, and 0.3 testing set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state = random_state_n
um)

#convert Y_train and Y_test into 1-d array so we don't get stupid warnings about needing a 1d array fo
r Y, in cross_validate function
Y_train_1d = Y_train.values.ravel()
Y_test_1d = Y_test.values.ravel()

#convert Y_train and Y_test into numpy array
Y_train_np_array = Y_train.to_numpy()
Y_test_np_array = Y_test.to_numpy()
```

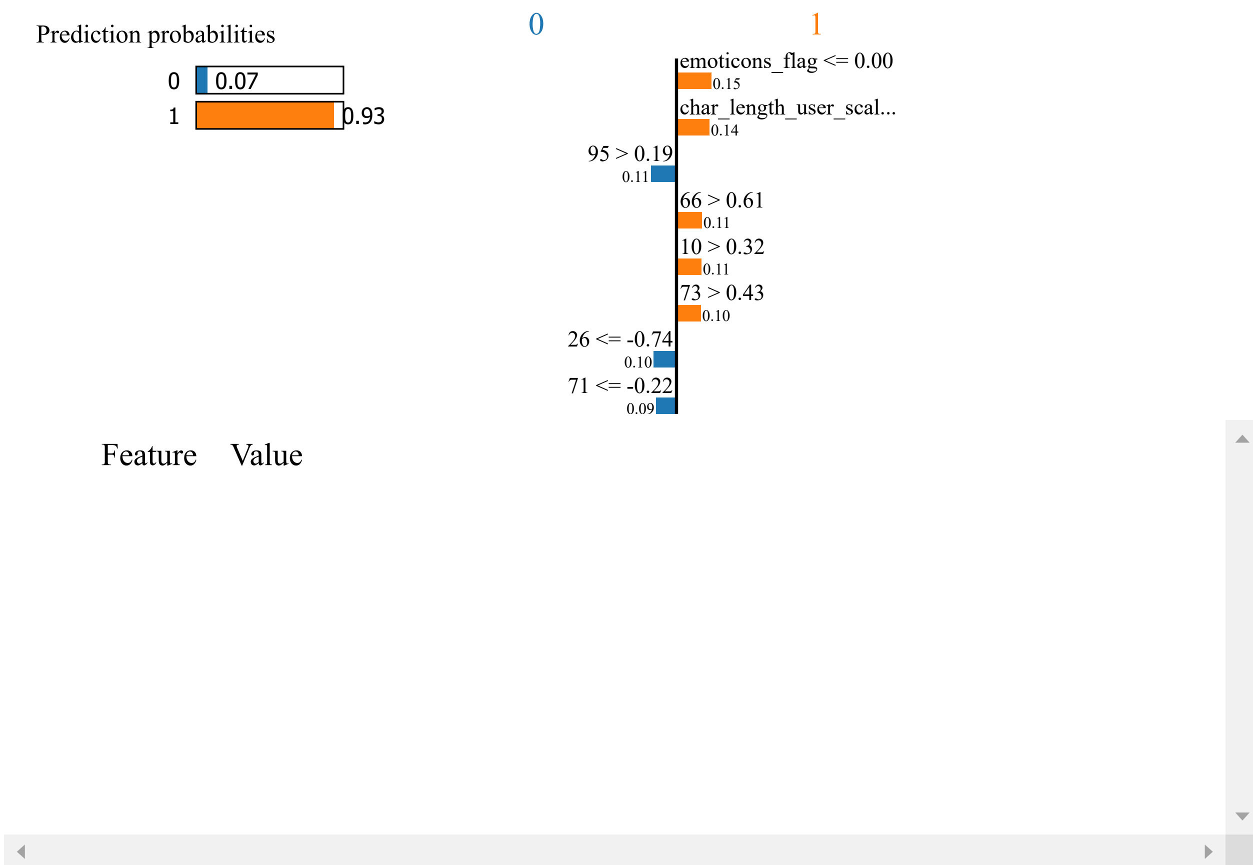
Models

Here we define a set of "basic" models we use as benchmarks to see which model algorithm is better than the others.

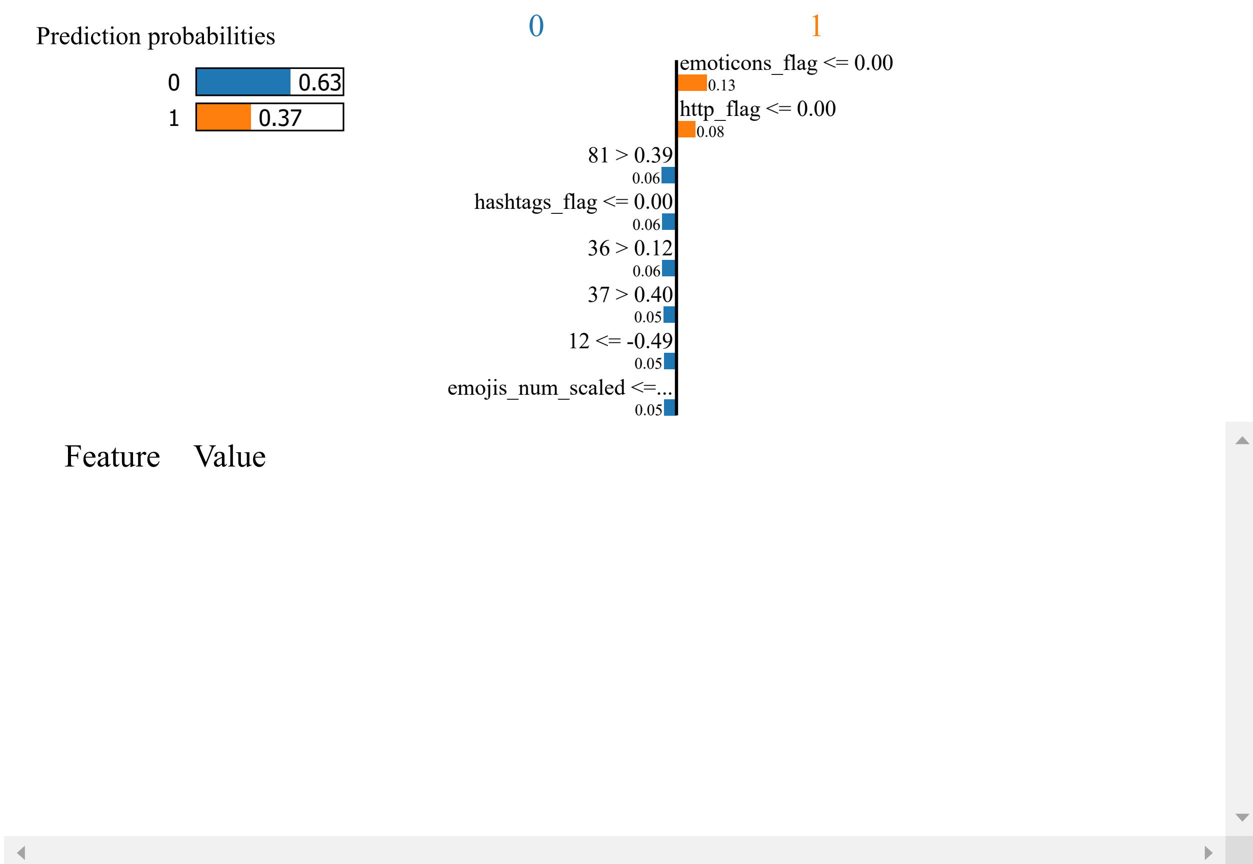
In [4]:
```
stacking_base_learners = [
    ('sbl_1', LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, \
                                          class_weight={1: 0.4, 0: 0.6}, penalty='l1', solver='lib
linear', )),
    ('sbl_2', KNeighborsClassifier(n_neighbors=3, )),
    ('sbl_3', DecisionTreeClassifier()),
    ('sbl_4', GaussianNB())
]

models = [
# Hyperparameter tuned GBM model
#     GradientBoostingClassifier(random_state = random_state_num, min_samples_split=140, min_samples_l
eaf=14, \
#                                subsample=0.9, n_estimators=10, max_features='sqrt', max_depth=8, \
#                                loss='deviance', learning_rate=0.15000000000000002, criterion='friedm
an_mse')

 #   KNeighborsClassifier(n_neighbors=3, ),
  #  DecisionTreeClassifier(),
   # SVC(C = 1000000, gamma = 'auto', kernel = 'rbf', probability=True),
    #GaussianNB(),
    #increased max_iter because it failed to converge at 100
    LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, class_weight={1: 0.4, 0:
0.6}, \
                       penalty='l1', solver='liblinear'),
    #AdaBoostClassifier(random_state = random_state_num),
    #RandomForestClassifier(random_state = random_state_num),
    #GradientBoostingClassifier(random_state = random_state_num),
    #BaggingClassifier(base_estimator=LogisticRegression(random_state = random_state_num, max_iter = 5
00, C=1, \
    #                                                class_weight={1: 0.4, 0: 0.6}, penalty='l1',
 solver='liblinear')),
    #StackingClassifier(estimators=stacking_base_learners, final_estimator=LogisticRegression())
]
```

LIME

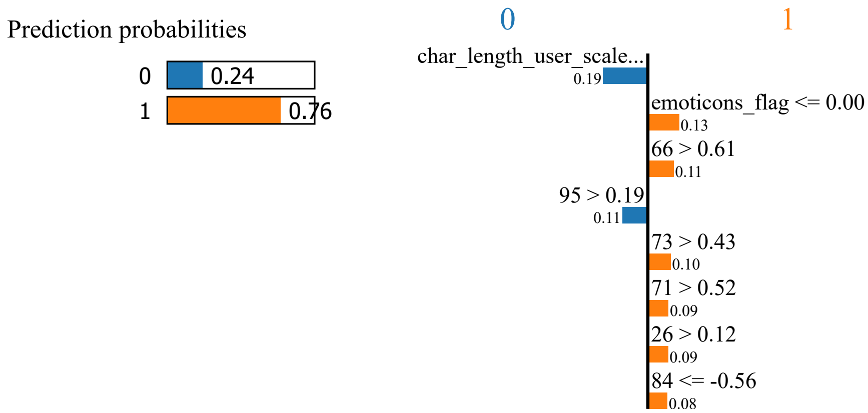
In [6]:
```
logReg_train = Y_train_np_array
logReg_eval = Y_test_np_array

logReg = LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, class_weight={1: 0.4
, 0: 0.6}, \
                       penalty='l1', solver='liblinear')

Y_train_predictions = cross_val_predict(
        logReg,
        X_train,
        Y_train_1d,
        cv = cv_num,
        method = 'predict_proba',
        n_jobs=-1
    )

logReg.fit(X_train, Y_train_1d)

trained_model = logReg.fit(X_train, Y_train_1d)
Y_test_predictions = trained_model.predict_proba(X_test)
Y_test_binary_predictions = trained_model.predict(X_test)

import lime.lime_tabular

explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                 feature_names=X_train.columns.values.tolist(),
                 class_names=['0' , '1'])

predict_fn = lambda x: logReg.predict_proba(x).astype(float)

exp = explainer.explain_instance(X_test.values[17], predict_fn, num_features=8)
exp.show_in_notebook(show_all=False)

exp = explainer.explain_instance(X_test.values[16], predict_fn, num_features=8)
exp.show_in_notebook(show_all=False)

exp = explainer.explain_instance(X_test.values[18], predict_fn, num_features=8)
exp.show_in_notebook(show_all=False)
```



Prediction probabilities
0 0.07
1 0.93
0
1
emoticons_flag <= 0.00
0.15
char_length_user_scal...
0.14
95 > 0.19
0.11
66 > 0.61
0.11
10 > 0.32
0.11
73 > 0.43
0.10
26 <= -0.74
0.10
71 <= -0.22
0.09
Feature Value



Prediction probabilities
0 0.63
1 0.37
0
1
emoticons_flag <= 0.00
0.13
http_flag <= 0.00
0.08
81 > 0.39
0.06
hashtags_flag <= 0.00
0.06
36 > 0.12
0.06
37 > 0.40
0.05
12 <= -0.49
0.05
emojis_num_scaled <=...
0.05
Feature Value



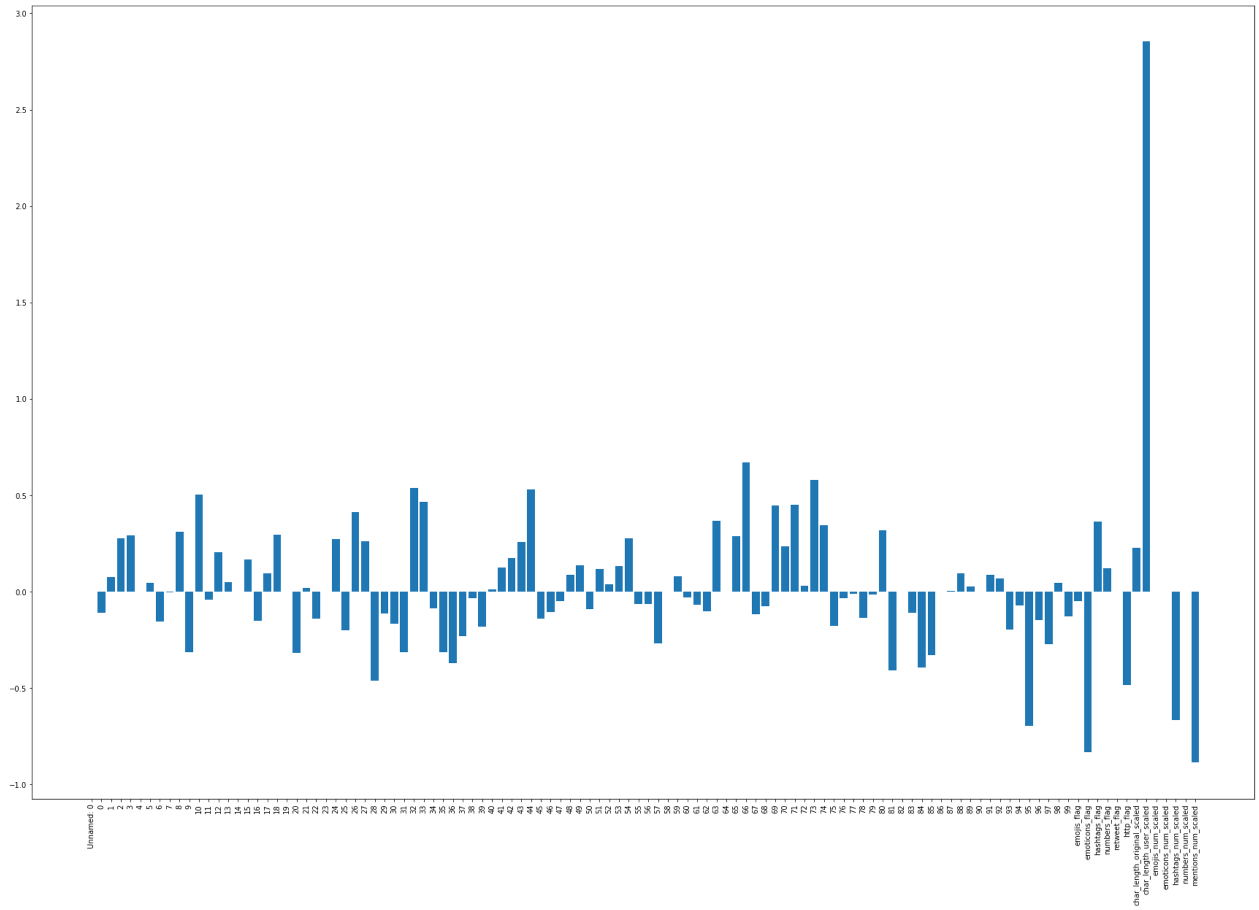
Feature	Value

In [36]:

```
feature_names =X_train.columns

importance = logReg.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
        print('Feature: %0s, Score: %.5f' % (feature_names[i],v))
# plot feature importance
plt.figure(figsize=(20,10))
plt.bar([x for x in range(len(importance))], importance)
plt.xticks(range(X_train.shape[1]), feature_names)
plt.xticks(rotation=90)
plt.savefig('logRegFeatureImportance.png')
plt.show()
```

```
Feature: Unnamed: 0, Score: 0.00002
Feature: 0, Score: -0.10906
Feature: 1, Score: 0.07456
Feature: 2, Score: 0.27539
Feature: 3, Score: 0.29285
Feature: 4, Score: -0.00020
Feature: 5, Score: 0.04547
Feature: 6, Score: -0.15652
Feature: 7, Score: -0.00226
Feature: 8, Score: 0.30894
Feature: 9, Score: -0.31224
Feature: 10, Score: 0.50387
Feature: 11, Score: -0.04224
Feature: 12, Score: 0.20386
Feature: 13, Score: 0.05106
Feature: 14, Score: 0.00000
Feature: 15, Score: 0.16626
Feature: 16, Score: -0.15221
Feature: 17, Score: 0.09515
Feature: 18, Score: 0.29535
Feature: 19, Score: 0.00000
Feature: 20, Score: -0.31937
Feature: 21, Score: 0.02017
Feature: 22, Score: -0.13856
Feature: 23, Score: 0.00000
Feature: 24, Score: 0.27456
Feature: 25, Score: -0.20008
Feature: 26, Score: 0.41356
Feature: 27, Score: 0.26180
Feature: 28, Score: -0.45992
Feature: 29, Score: -0.11188
Feature: 30, Score: -0.16764
Feature: 31, Score: -0.31510
Feature: 32, Score: 0.53725
Feature: 33, Score: 0.46635
Feature: 34, Score: -0.08511
Feature: 35, Score: -0.31413
Feature: 36, Score: -0.36980
Feature: 37, Score: -0.23255
Feature: 38, Score: -0.03237
Feature: 39, Score: -0.17986
Feature: 40, Score: 0.01004
Feature: 41, Score: 0.12500
Feature: 42, Score: 0.17552
Feature: 43, Score: 0.25745
Feature: 44, Score: 0.52913
Feature: 45, Score: -0.14077
Feature: 46, Score: -0.10428
Feature: 47, Score: -0.04785
Feature: 48, Score: 0.08908
Feature: 49, Score: 0.13657
Feature: 50, Score: -0.09150
Feature: 51, Score: 0.11837
Feature: 52, Score: 0.03686
Feature: 53, Score: 0.13400
Feature: 54, Score: 0.27807
Feature: 55, Score: -0.06278
Feature: 56, Score: -0.06548
Feature: 57, Score: -0.26977
Feature: 58, Score: 0.00000
Feature: 59, Score: 0.08153
Feature: 60, Score: -0.03188
Feature: 61, Score: -0.06688
Feature: 62, Score: -0.10128
Feature: 63, Score: 0.36552
Feature: 64, Score: 0.00000
Feature: 65, Score: 0.28738
Feature: 66, Score: 0.67052
Feature: 67, Score: -0.11655
Feature: 68, Score: -0.07475
Feature: 69, Score: 0.44511
```

```
Feature: 70, Score: 0.23567
Feature: 71, Score: 0.44879
Feature: 72, Score: 0.03095
Feature: 73, Score: 0.57961
Feature: 74, Score: 0.34332
Feature: 75, Score: -0.17812
Feature: 76, Score: -0.03402
Feature: 77, Score: -0.01037
Feature: 78, Score: -0.13590
Feature: 79, Score: -0.01533
Feature: 80, Score: 0.31629
Feature: 81, Score: -0.40870
Feature: 82, Score: 0.00000
Feature: 83, Score: -0.11097
Feature: 84, Score: -0.39265
Feature: 85, Score: -0.32938
Feature: 86, Score: 0.00000
Feature: 87, Score: 0.00287
Feature: 88, Score: 0.09670
Feature: 89, Score: 0.02624
Feature: 90, Score: 0.00000
Feature: 91, Score: 0.08645
Feature: 92, Score: 0.06919
Feature: 93, Score: -0.19625
Feature: 94, Score: -0.07249
Feature: 95, Score: -0.69517
Feature: 96, Score: -0.14701
Feature: 97, Score: -0.27087
Feature: 98, Score: 0.04453
Feature: 99, Score: -0.12889
Feature: emojis_flag, Score: -0.05040
Feature: emoticons_flag, Score: -0.83323
Feature: hashtags_flag, Score: 0.36318
Feature: numbers_flag, Score: 0.12090
Feature: retweet_flag, Score: 0.00000
Feature: http_flag, Score: -0.48279
Feature: char_length_original_scaled, Score: 0.22784
Feature: char_length_user_scaled, Score: 2.85418
Feature: emojis_num_scaled, Score: 0.00000
Feature: emoticons_num_scaled, Score: 0.00000
Feature: hashtags_num_scaled, Score: -0.66614
Feature: numbers_num_scaled, Score: 0.00000
Feature: mentions_num_scaled, Score: -0.88700
```



References

Code for this project is either directly from (with some modification), or inspired by, but not limited to the following sources:

- Interpreting Machine Learning Models: https://towardsdatascience.com/interpretability-in-machine-learning-70c30694a05f (https://towardsdatascience.com/interpretability-in-machine-learning-70c30694a05f)
- Interpreting Machine Learning Models: https://medium.com/ansaro-blog/interpreting-machine-learning-models-1234d735d6c9 (https://medium.com/ansaro-blog/interpreting-machine-learning-models-1234d735d6c9)
- SciKit-Learn API Reference: https://scikit-learn.org/stable/modules/classes.html (https://scikit-learn.org/stable/modules/classes.html)
- Respective API Reference for each package used
- Feature Importance and Model Interpretability (including LIME):
  - https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5 (https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5)
  - https://www.analyticsvidhya.com/blog/2017/06/building-trust-in-machine-learning-models/ (https://www.analyticsvidhya.com/blog/2017/06/building-trust-in-machine-learning-models/)
  - https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5 (https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5)
  - https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5 (https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5)
  - https://lime-ml.readthedocs.io/en/latest/lime.html#module-lime.lime_text (https://lime-ml.readthedocs.io/en/latest/lime.html#module-lime.lime_text)
  - https://marcotcr.github.io/lime/tutorials/Lime%20-%20basic%20usage%2C%20two%20class%20case.html (https://marcotcr.github.io/lime/tutorials/Lime%20-%20basic%20usage%2C%20two%20class%20case.html)
  - https://www.kdnuggets.com/2019/09/python-libraries-interpretable-machine-learning.html (https://www.kdnuggets.com/2019/09/python-libraries-interpretable-machine-learning.html)