SPAM SMS DETECTION

ABSTRACT

Short Message Service (SMS) is the most important communication tool in recent decades. With the increased popularity of mobile devices, the usage rate of SMS will increase more and more in years. SMS is a practical method used to reach individuals directly. But this practical and easy method can cause SMS to be misused. The advertising or promotional SMS of the companies are an examplesofthismisuse.Inthisstudy,aspamSMSdetectiontechniqueisproposedusingData Mining (DM)methods.

INTRODUCTION

Short Message Service (SMS) is one of the most popular communication service where messages are sent electronically. The increase in the use of mobile devices also increased the number ofSMSsentandreceived.WiththeincreaseduseofSMS,thecostofSMSserviceshasalsodecreased. ThelowpriceandthehighbandwidthoftheSMSnetworkhaveattractedalargeamountofSMSspam. ThisrisehasalsoincreasedthemalicioususeofSMS,resultinginaspamSMSproblem.AspamSMSis any unwanted message that is sent to user’s mobile devices. Spam SMS include advertisements, free services, promotions, etc. According to people classify SMS Spam as annoying (32.3%), wasting time (24.8%), and violating personal privacy (21.3%) [1]. SMS is not text-rich. Therefore, spam SMS detection is generally based on text mining. Text mining aims to get structured data through the text, such as classification,clustering, concept or entity extraction, texts production of granular taxonomy, textualsentimentalanalysis,documentsummarizationandentityrelationshipmodeling.Toobtainthe structured data, information retrieval, lexical analysis, pattern recognition, word frequency, tagging, information extraction, data mining and visualization methods areused.[2]

LITERATURE REVİEW

Inthisproject,weworkedonthearticlethatSupportsVectorMachineBasedSpamSMSDetection[1].

AspamSMSdetectiontechniqueisproposedusingDataMining(DM)methods.Intheproposed study, data mining algorithms such as Naive Bayes (NB), K-Nearest Neighborhood (KNN), Support Vector Machine (SVM), Random Forest (RF) and Random Tree (RT) is selected. SMSSpamCollection dataset, which is contain 747 spam SMS and 4827 ham SMS, isused.

First of all, we used Naive Bayes is one of the most effective learning algorithms in machine learning. Bayesian spam SMS filtering is a statistical method of detecting spam SMSs based on Bayes' theorem to calculate the probability that a SMS is actually a spam SMS. Naive Bayes algorithm is one of the machine learning methods that is used in textclassification.

Secondly, we used The KNN is a pattern classifier that allows classification without the need to know the probability distributions of classes.

Thirdly, we used SVM classification method is used to determine if a SMS is a spam SMS or ham SMS. SVM data mining method is used to classify SMS as malicious or not.

WeuseXGBoostthatisanimplementationofgradientboosteddecisiontreesdesignedfor speedand performance that is dominative competitive machinelearning.

MATERIAL and METHOD

Inthisstudy,somedataminingclassificationmethodshavebeenusedtodetermineifaSMSis actually a spam SMS or ham SMS. Naive Bayes (NB), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Gradient Boosting Classifier, XGBootst, and Stochastic Gradient Descent data mining methods are used to classify SMS as malicious or not. These methodsare describedbelow.

Naive Bayes (NB)

It is aclassification techniquebased on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter.Evenifthesefeaturesdependoneachotheroruponthe existenceoftheotherfeatures, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

NBusesadiscriminantfunctiontocomputetheconditionalprobabilitiesofP(Ci|X).Asshown in formula (1) the inputs, P(Ci | X) denotes the probability that, example X belongs to classCi

$$P(Ci|X) = \frac{P(Ci) * P(X|Ci)}{P(X)} \quad (1)$$

$P(X)$ (1) P(Ci) is the probability of observing class i. P(X | Ci) denotes the probability of observing the example, given class Ci. P(X) is the probability of the input, which is independent of the classes.[3]

Order	Study	Dataset	Method	Result(%)	FP	TP
1	Proposed Study	SMS Spam Collection	**SVM**	**98.3315**	**0,087**	**0,983**
			NB	96.7887	0,108	0,968
			RT	95.4611	0,206	0,955
			RF	97.4345	0,166	0,974
			KNN	95.1381	0,310	0,951

Naive Bayes

In [84]:
```
data_tfidf_train = data_tfidf_train.A
data_tfidf_test = data_tfidf_test.A
```

In [85]:
```
spam_detect_model = MultinomialNB().fit(data_tfidf_train, label_train)
pred_test_MNB = spam_detect_model.predict(data_tfidf_test)
acc_MNB = accuracy_score(label_test, pred_test_MNB)
print(acc_MNB) #96.5% accuracy(MultinomialNB)
```

```
0.965311004784689
```

As a result of the article, the result of the naive bayes algorithm applied to the dataset was 96.78% whereas the result of the naive bayes algorithm that we applied to the same dataset was 96.53%. It is quite a value we approach.

K-Nearest Neighborhood (KNN)

K-NN (K-Nearest Neighbor) algorithm is one of the simplest and most widely used classification algorithm. K-NN is a non-parametric, lazy learning algorithm. If we try to understandtheconceptoflazy,unlikeeagerlearning,lazylearningdoesnothaveaneducational stage. It does not learn the training data, but instead ler memorizes ini the training data set. When we want to make a prediction, it looks for the nearest neighbors in the entire data set. In the operation of the algorithm, a K value is determined. This K value means the number of elements to look at. When a value is reached, the nearest K element is taken and the distance betweenthevalueiscalculated.TheEuclideanfunctionisusuallyusedfordistancecalculation. AsanalternativetoEuclideanfunction,Manhattan,MinkowskiandHammingfunctionscanbe used. After the distance is calculated, it is sorted and the incoming value is assigned to the appropriate class.

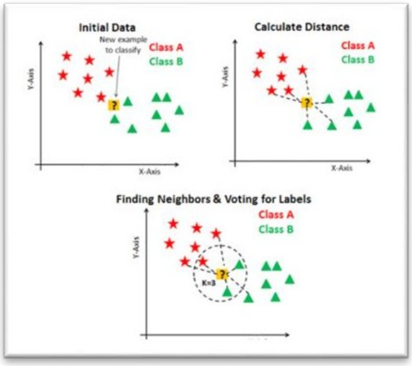


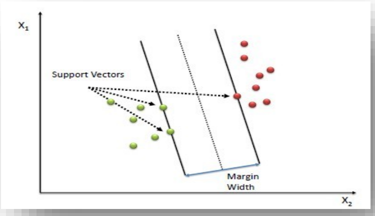
Figure 1 – K –Nearest Neighborhood(KNN)

KNN

```
In [112]: pipe_KNN = Pipeline([ ('bow'  , CountVectorizer(analyzer = remove_punctuation_and_stopwords) ),
                    ('tfidf'   , TfidfTransformer()),
                    ('clf_KNN' , KNeighborsClassifier() )
                     ])

          parameters_KNN = {'clf_KNN__n_neighbors': (8,15,20), }

          grid_KNN = GridSearchCV(pipe_KNN, parameters_KNN, cv=5,
                         n_jobs=-1, verbose=1)

          grid_KNN.fit(X=sms_train, y=label_train)

          Fitting 5 folds for each of 3 candidates, totalling 15 fits

          [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
          [Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:  1.0min finished

Out[112]: GridSearchCV(cv=5, error_score='raise-deprecating',
                 estimator=Pipeline(memory=None,
                           steps=[('bow',
                                   CountVectorizer(analyzer=<function remove_punctuation_and_stopwords at 0x7f4c87fd2050>,
                                                  binary=False,
                                                  decode_error='strict',
                                                  dtype=<class 'numpy.int64'>,
                                                  encoding='utf-8',
                                                  input='content',
                                                  lowercase=True,
                                                  max_df=1.0,
                                                  max_features=None,
                                                  min_df=1,
                                                  ngram_range=(1, 1),
                                                  prepr...
                                                   sublinear_tf=False,
                                                   use_idf=True)),
                                  ('clf_KNN',
                                   KNeighborsClassifier(algorithm='auto',
                                                       leaf_size=30,
                                                       metric='minkowski',
                                                       metric_params=None,
                                                       n_jobs=None,
                                                       n_neighbors=5, p=2,
                                                       weights='uniform'))],
                           verbose=False),
                 iid='warn', n_jobs=-1,
                 param_grid={'clf_KNN__n_neighbors': (8, 15, 20)},
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                 scoring=None, verbose=1)

In [113]: print(grid_KNN.best_params_)
          print(grid_KNN.best_score_) #93% accuracy(K-Nearest Neighbor)

          {'clf_KNN__n_neighbors': 15}
          0.936923076923077
```

As a result of the article, the result of theK –Nearest Neighborhood(KNN)algorithm applied to the dataset was 95.13% whereas the result of theK –Nearest Neighborhood(KNN)algorithm that we applied to the same dataset was 93.69%.

Support Vector Machine



Support Vector Machine; It is a classification algorithm similar to Logistic Regression. Both try to find the best line separating the two classes. The algorithm allows the line to be drawn to be set in two classes to pass the elements farthest away. A classifier that takes no parameters. SVM can also classify linear and nonlinear data, but generally tries to classify the data linearly.

Kernel Trick

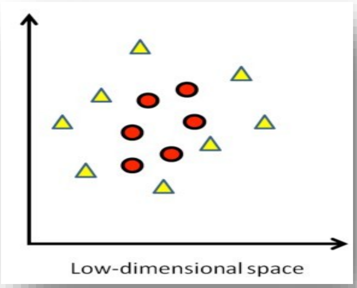


Figure 2

SVM tries to classify the data linearly, but in some cases it is not possible (Figure 2). To get rid of this situation, we use the Kernel Trick. If we can create a new dimension, we may be able to classify it linearly. For example, if we can lift the red dots up a bit (z axis) in the graph of Figure 2 and create a 3rd dimension, we can create a linear line with SVM.

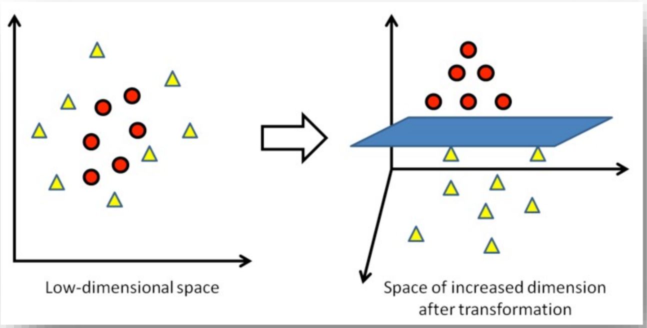


Figure 3

SVC

```
In [67]: pipe_SVC = Pipeline([ ('bow'  , CountVectorizer(analyzer = remove_punctuation_and_stopwords) ),
                              ('tfidf'   , TfidfTransformer()),
                              ('clf_SVC' , SVC(gamma='auto', C=1000)),
                               ])


         parameters_SVC = dict(tfidf=[None, TfidfTransformer()],
                                 clf_SVC__C=[500, 1000,1500]
                                 )
         grid_SVC = GridSearchCV(pipe_SVC, parameters_SVC,
                                   cv=5, n_jobs=-1, verbose=1)

         grid_SVC.fit(X=sms_train, y=label_train)

         Fitting 5 folds for each of 6 candidates, totalling 30 fits

         [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
         [Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.5min finished

Out[67]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=Pipeline(memory=None,
                                         steps=[('bow',
                                                 CountVectorizer(analyzer=<function remove_punctuation_and_stopwords at 0x7f4c87fd2050>,
                                                                 binary=False,
                                                                 decode_error='strict',
                                                                 dtype=<class 'numpy.int64'>,
                                                                 encoding='utf-8',
                                                                 input='content',
                                                                 lowercase=True,
                                                                 max_df=1.0,
                                                                 max_features=None,
                                                                 min_df=1,
                                                                 ngram_range=(1, 1),
                                                                 prepr...
                                                     kernel='rbf', max_iter=-1,
                                                     probability=False,
                                                     random_state=None, shrinking=True,
                                                     tol=0.001, verbose=False))],
                                         verbose=False),
                      iid='warn', n_jobs=-1,
                      param_grid={'clf_SVC__C': [500, 1000, 1500],
                                  'tfidf': [None,
                                            TfidfTransformer(norm='l2', smooth_idf=True,
                                                             sublinear_tf=False,
                                                             use_idf=True)]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring=None, verbose=1)

In [68]: print(grid_SVC.best_params_)
         print(grid_SVC.best_score_)

         {'clf_SVC__C': 1500, 'tfidf': None}
         0.9761538461538461
```

As a result of the article, the result of the Support Vector Machine algorithm applied to the dataset was 97.6 % whereas the result of the Support Vector Machine algorithm that we applied to the same dataset was 97.6 %.

Order	Study	Dataset	Method	Result(%)	FP	TP
1	Proposed Study	SMS Spam Collection	**SVM**	**98.3315**	**0,087**	**0,983**
			NB	96.7887	0,108	0,968
			RT	95.4611	0,206	0,955
			RF	97.4345	0,166	0,974
			KNN	95.1381	0,310	0,951
2	Choudhary, N., et al [1]	SMS Spam Collection	NB	94.1	0.077	0.941
			DT	96	0.133	0.960
			RF	96,5	0.102	0.965
3	Almeida, T.A., el al [16]	SMS Spam Collection	linear SVM	97,6	0.18	0.976

Gradient Boosting Classifier

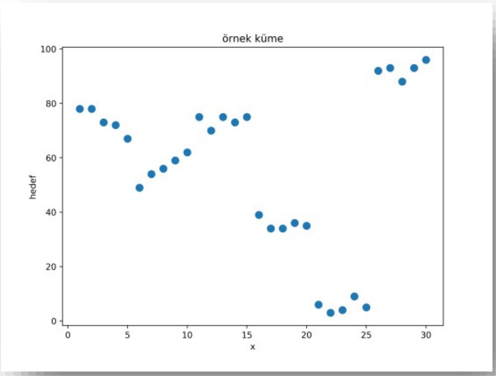


Figure 4

Gradient Boosting creates an “F” function that generates predictions in the first iteration. Calculatesthedifferencebetweentheestimatesandthetargetvalueandcreatesthefunction“h için for these differences. In the second iteration, it combines the functions “F” and “h ve and calculates the difference between re-estimates and targets. In this way, it constantly tries to increase the success of the “F” function by adding on it, thus reducing the difference between the predictions and the targets to zero. In the following image, you can see the model's prediction on the 1st iteration as a red line on the left graph. I also showed the difference betweentheestimatesandthetargetvalueforeachxvalueinthegraphontheright.Thesuccess of the model will increase as the iterationsprogress.

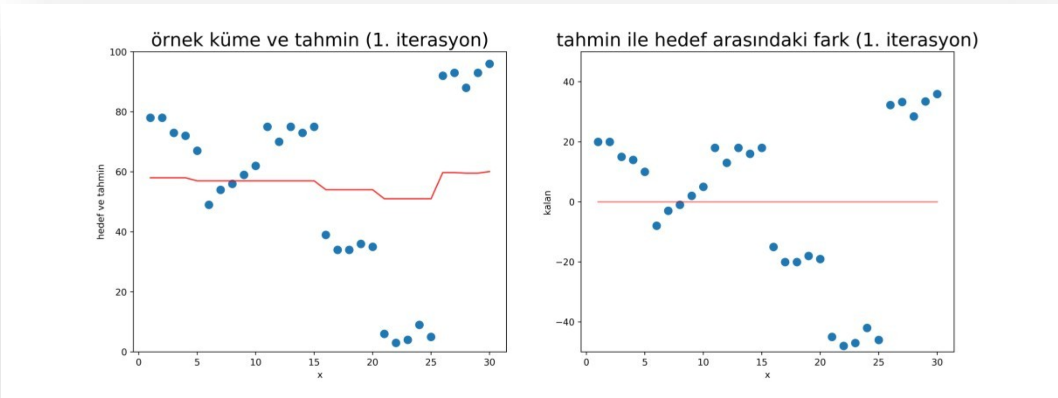


Figure 5

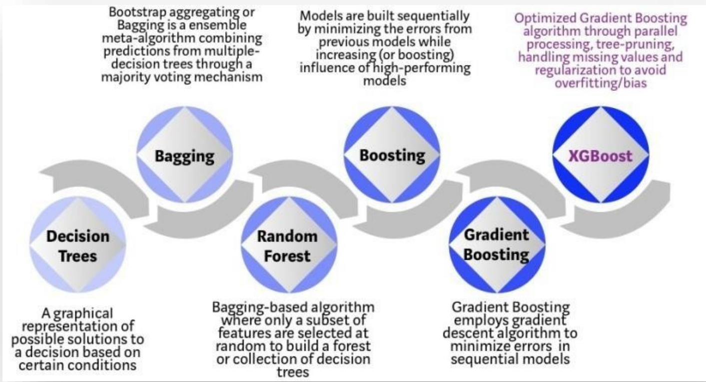
GradientBoostingClassifier

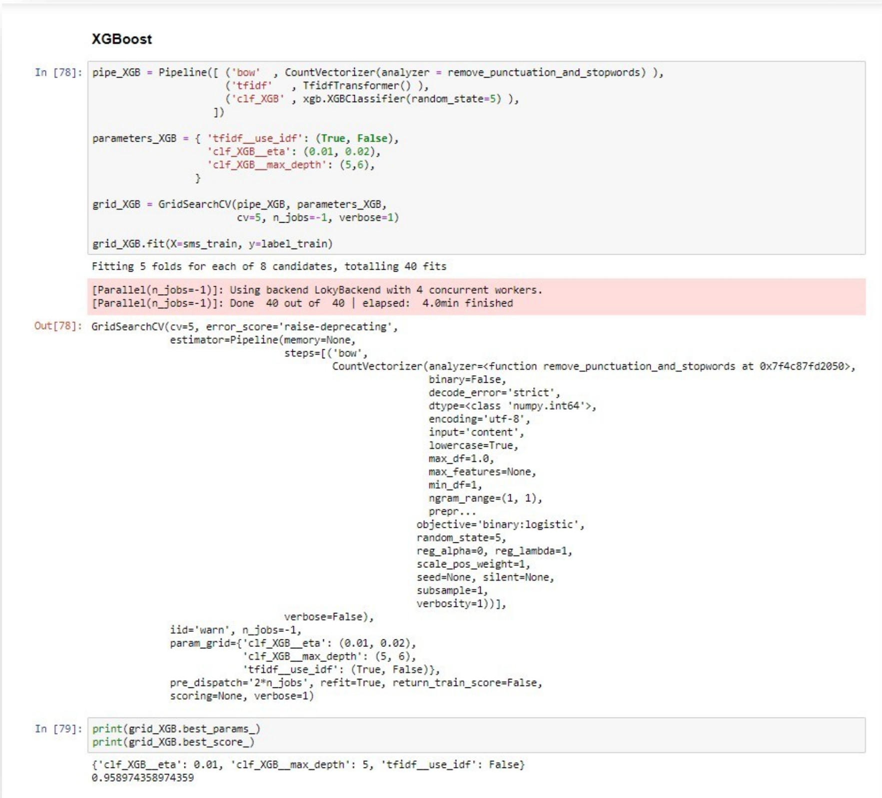
```
In [69]: pipe_GBC = Pipeline([ ('bow'  , CountVectorizer(analyzer = remove_punctuation_and_stopwords) ),
                               ('tfidf'   , TfidfTransformer() ),
                               ('clf_GBC' , GradientBoostingClassifier(random_state=5) ),
                             ])

         parameters_GBC = { 'tfidf__use_idf': (True, False),
                            'clf_GBC__learning_rate': (0.1, 0.2),
                            #'clf_GBC__min_samples_split': (3,5),
                          }

         grid_GBC = GridSearchCV(pipe_GBC, parameters_GBC,
                                 cv=5, n_jobs=-1, verbose=1)

         grid_GBC.fit(X=sms_train, y=label_train)

         Fitting 5 folds for each of 4 candidates, totalling 20 fits

         [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
         [Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:  1.7min finished

Out[69]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=Pipeline(memory=None,
                                         steps=[('bow',
                                                 CountVectorizer(analyzer=<function remove_punctuation_and_stopwords at 0x7f4c87fd2050>,
                                                                 binary=False,
                                                                 decode_error='strict',
                                                                 dtype=<class 'numpy.int64'>,
                                                                 encoding='utf-8',
                                                                 input='content',
                                                                 lowercase=True,
                                                                 max_df=1.0,
                                                                 max_features=None,
                                                                 min_df=1,
                                                                 ngram_range=(1, 1),
                                                                 prepr...
                                                                          min_weight_fraction_leaf=0.0,
                                                                          n_estimators=100,
                                                                          n_iter_no_change=None,
                                                                          presort='auto',
                                                                          random_state=5,
                                                                          subsample=1.0,
                                                                          tol=0.0001,
                                                                          validation_fraction=0.1,
                                                                          verbose=0,
                                                                          warm_start=False))],
                                         verbose=False),
                      iid='warn', n_jobs=-1,
                      param_grid={'clf_GBC__learning_rate': (0.1, 0.2),
                                  'tfidf__use_idf': (True, False)},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring=None, verbose=1)

In [72]: print(grid_GBC.best_params_)
         print(grid_GBC.best_score_)

         {'clf_GBC__learning_rate': 0.2, 'tfidf__use_idf': False}
         0.9641025641025641
```

The Gradient Boosting Classifier algorithm was not applied in the article, but since we wanted to examine the results of the data set for this algorithm, we applied this algorithm in addition and obtained 96.41% result on the data set.

XGBootst

XGBoostisadecison-treebasedMLsystemwithgradientboosting.Ifyourdataisnon-structureddata suchaspicture/text/sound,itwillbetherightchoicefordeeplearningwithartificialneuralnetworks. However, if you don't have a lot of data (probably not, but it does), I suggest you start with decision- basedalgorithms.Decision-basedalgorithmshaveevolvedovertime.Youcanseethisevolutionmore easily in the following flow. As a result of the XGBoost method we applied to our dataset, we got a value of95.89%.





XGBoost

```
In [78]: pipe_XGB = Pipeline([ ('bow'  , CountVectorizer(analyzer = remove_punctuation_and_stopwords) ),
                              ('tfidf'   , TfidfTransformer() ),
                              ('clf_XGB' , xgb.XGBClassifier(random_state=5) ),
                             ])

         parameters_XGB = { 'tfidf__use_idf': (True, False),
                            'clf_XGB__eta': (0.01, 0.02),
                            'clf_XGB__max_depth': (5,6),
                          }

         grid_XGB = GridSearchCV(pipe_XGB, parameters_XGB,
                                 cv=5, n_jobs=-1, verbose=1)

         grid_XGB.fit(X=sms_train, y=label_train)

         Fitting 5 folds for each of 8 candidates, totalling 40 fits

         [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
         [Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  4.0min finished

Out[78]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=Pipeline(memory=None,
                                         steps=[('bow',
                                                 CountVectorizer(analyzer=<function remove_punctuation_and_stopwords at 0x7f4c87fd2050>,
                                                                 binary=False,
                                                                 decode_error='strict',
                                                                 dtype=<class 'numpy.int64'>,
                                                                 encoding='utf-8',
                                                                 input='content',
                                                                 lowercase=True,
                                                                 max_df=1.0,
                                                                 max_features=None,
                                                                 min_df=1,
                                                                 ngram_range=(1, 1),
                                                                 prepr...
                                                               objective='binary:logistic',
                                                               random_state=5,
                                                               reg_alpha=0, reg_lambda=1,
                                                               scale_pos_weight=1,
                                                               seed=None, silent=None,
                                                               subsample=1,
                                                               verbosity=1))],
                                         verbose=False),
                      iid='warn', n_jobs=-1,
                      param_grid={'clf_XGB__eta': (0.01, 0.02),
                                  'clf_XGB__max_depth': (5, 6),
                                  'tfidf__use_idf': (True, False)},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring=None, verbose=1)

In [79]: print(grid_XGB.best_params_)
         print(grid_XGB.best_score_)

         {'clf_XGB__eta': 0.01, 'clf_XGB__max_depth': 5, 'tfidf__use_idf': False}
         0.958974358974359
```

Stochastic Gradient Descent (SGD)

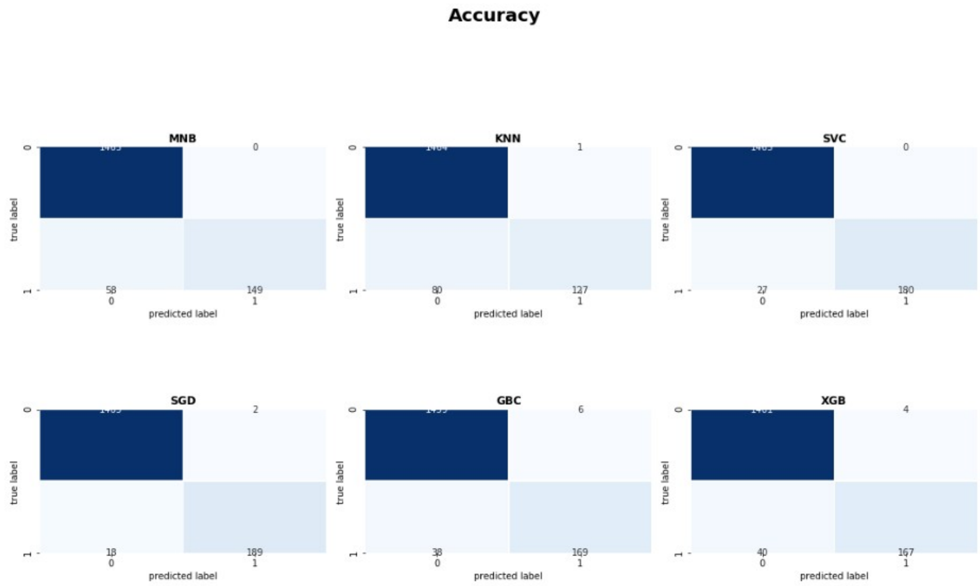
The word ‘stochastic‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get really huge. Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform. This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration. With this algorithm we applied to our dataset, we obtained 97.66% result.

```
In [96]: pipe_SGD = Pipeline([ ('bow'  , CountVectorizer(analyzer = remove_punctuation_and_stopwords) ),
                               ('tfidf'   , TfidfTransformer()),
                               ('clf_SGD' , SGDClassifier(random_state=5)),
                                ])

         parameters_SGD = {
             #'vect__ngram_range': ((1, 1), (1, 2)),  # unigrams or bigrams
             'tfidf__use_idf': (True, False),
             #'tfidf__norm': ('l1', 'l2'),
             #'clf_SGD__max_iter': (5,10),
             'clf_SGD__alpha': (1e-05, 1e-04),
         }

         grid_SGD = GridSearchCV(pipe_SGD, parameters_SGD, cv=5,
                                 n_jobs=-1, verbose=1)

         grid_SGD.fit(X=sms_train, y=label_train)

         Fitting 5 folds for each of 4 candidates, totalling 20 fits

         [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
         [Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:  1.6min finished

Out[96]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=Pipeline(memory=None,
                                         steps=[('bow',
                                                 CountVectorizer(analyzer=<function remove_punctuation_and_stopwords at 0x7f4c87fd2050>,
                                                                 binary=False,
                                                                 decode_error='strict',
                                                                 dtype=<class 'numpy.int64'>,
                                                                 encoding='utf-8',
                                                                 input='content',
                                                                 lowercase=True,
                                                                 max_df=1.0,
                                                                 max_features=None,
                                                                 min_df=1,
                                                                 ngram_range=(1, 1),
                                                                 prepr...
                                                               max_iter=1000,
                                                               n_iter_no_change=5,
                                                               n_jobs=None, penalty='l2',
                                                               power_t=0.5,
                                                               random_state=5,
                                                               shuffle=True, tol=0.001,
                                                               validation_fraction=0.1,
                                                               verbose=0,
                                                               warm_start=False))],
                                         verbose=False),
                      iid='warn', n_jobs=-1,
                      param_grid={'clf_SGD__alpha': (1e-05, 0.0001),
                                  'tfidf__use_idf': (True, False)},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring=None, verbose=1)

In [97]: print(grid_SGD.best_params_)
         print(grid_SGD.best_score_)

         {'clf_SGD__alpha': 0.0001, 'tfidf__use_idf': True}
         0.9766666666666667
```

Comparison of Algorithms

The comparison of our algorithms is as follows:

```
In [149]: plot_all_confusion_matrices(label_test, dict_pred, "Accuracy")
```



Classification Report

```
In [157]: print(classification_report(label_test, pred_test_MNB))

              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1465
           1       1.00      0.72      0.84       207

    accuracy                           0.97      1672
   macro avg       0.98      0.86      0.91      1672
weighted avg       0.97      0.97      0.96      1672
```

```
In [158]: print(classification_report(label_test, pred_test_grid_KNN))

              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1465
           1       0.99      0.61      0.76       207

    accuracy                           0.95      1672
   macro avg       0.97      0.81      0.87      1672
weighted avg       0.95      0.95      0.95      1672
```

```
In [159]: print(classification_report(label_test, pred_test_grid_SVC))

              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1465
           1       1.00      0.87      0.93       207

    accuracy                           0.98      1672
   macro avg       0.99      0.93      0.96      1672
weighted avg       0.98      0.98      0.98      1672
```

```
In [160]: print(classification_report(label_test, pred_test_grid_GBC))

              precision    recall  f1-score   support

           0       0.97      1.00      0.99      1465
           1       0.97      0.82      0.88       207

    accuracy                           0.97      1672
   macro avg       0.97      0.91      0.93      1672
weighted avg       0.97      0.97      0.97      1672
```

```
I'['.?1'prin zlzmifica?ion r«por/label IesI, telp'idXSB))
                  pne‹iiou    recall f1-moresupport

     weighted avg0.97            0.97      0.97      1672
```

```
n [162]: print(classification_report(label_test, pred_test_grid_SGD
                  precision    recall  f1-score   support
```

Precision Score

```
    print(clf, "", p ision_s‹ore(lobeI_test, dit_ped[zIf]))

    print(clf, "", p ision_s‹ore(lobeI_test, dit_ped[zIf]        enage=Mone, lobeIz-[†,1]))
```

```
KNN   0.9921875
```

```
oBc[ x ° 7 v i   exwits)
```

F1-Score

```
    print(clf, "  f1 •cone[labcl testdietp.ed[<lf];

for<lf  list zlf
    print(clf, "", f1_=zor/Izb«l_tz=t dizt_p'ed[lf], a'zrsge-None lzb«ls-[z,1]j)
```

```
GBC   0.8848167539267014
XGB   0.8835978835978836
```

```
GBC   [0.98514517 0.88481675]
XGB   [0.98516521 0.88359788]
```

ROC-AUC Score

In [163]:
```
for clf in list_clf :
    print(clf, " ", roc_auc_score(label_test, dict_pred[clf] ))
```

```
MNB   0.8599033816425121
KNN   0.806421988095827
SVC   0.9347826086956521
SGD   0.9558391452737794
GBC   0.9061647788165076
XGB   0.9020164547987667
```

Precision Score

In [154]:
```
for clf in list_clf :
    print(clf, " ", precision_score(label_test, dict_pred[clf]))
print("---------------------------")
for clf in list_clf :
    print(clf, " ", precision_score(label_test, dict_pred[clf], average=None, labels=[0,1]))

#TP / (TP + FP)
```

```
MNB   1.0
KNN   0.9921875
SVC   1.0
SGD   0.9895287958115183
GBC   0.9657142857142857
XGB   0.9766081871345029
---------------------------
MNB   [0.96191727 1.        ]
KNN   [0.94818653 0.9921875 ]
SVC   [0.98190349 1.        ]
SGD   [0.98784605 0.9895288 ]
GBC   [0.9746159  0.96571429]
XGB   [0.9733511  0.97660819]
```

REFERENCES

[1]Tekerek A., “Support vector machine based spam sms detection”, Politeknik Dergisi, 22(3): 779- 784,(2019).

[2]Choudhary, N., & Jain, A. K., “Towards Filtering of SMS Spam Messages Using Machine Learning BasedTechnique”InAdvancedInformaticsforComputingResearchSpringer,Singapore,18-30(2017).