Email Classification using Common Machine learning Model with Bootstrap Confidence Intervals and Other Evaluation Metric

Introduction: The goal of this project was to develop a machine learning model for email classification into spam and ham categories. The dataset used for training and evaluation contained email messages labeled with their respective categories. The approach involved using logistic regression as the classification algorithm and employing

Method: Bootstrap resampling to estimate confidence intervals for the Spam Email s performance metrics.

1. Data Preparation:
    a. The dataset was read into a Pandas Data Frame, which consisted of two columns: Category and Message.
    b. Preprocessing steps, such as removing non-text emails, were applied to clean the dataset.
    c. The Category column was encoded, assigning the value 1 for ham emails and 0 for spam emails.
2. Training and Testing Data Split:
    a. The dataset was divided into training and test sets using the train_test_split function from scikit-learn.
    b. The training set comprised 80% of the data, while the remaining 20% was used for testing.
3. Feature Extraction:
    a. Text data was transformed into numerical feature vectors using the TF-IDF vectorization technique.
    b. The TfidfVectorizer class from scikit-learn was utilized, considering parameters such as minimum document frequency, stop words, and lowercase conversion.
    c. The training set was transformed into feature vectors using the fit_transform method, and the test set was transformed using transform.

Model Application:

1. Logistic Regression:
    - Training Accuracy: 0.9670
    - Training Precision: 0.9643
    - Training Recall: 0.9990
    - Test Accuracy: 0.9094
    - Test Precision: 0.9048
    - Test Recall: 1.0000
    - Accuracy Confidence Interval: [0.02421525 0.04484305]

2. K-Nearest Neighbors (KNN):

- Training Accuracy: 0.9201
- Training Precision: 0.9159
- Training Recall: 0.9997
- Test Accuracy: 0.9094
- Test Precision: 0.9048
- Test Recall: 1.0000
- Accuracy Confidence Interval: [0.89237668 0.92556054]

3. Lasso Regression Training Accuracy:
    - Accuracy: 0.8609865470852018
    - Precision: 0.8609865470852018
    - Recall: 1.0
    - F1-Score: 0.9253012048192771
    - Accuracy Confidence Interval: [0.83946188 0.88161435]

Conclusion:

In this project, we developed machine learning models for email classification into spam and ham categories. Logistic regression, K-nearest neighbors (KNN), and Lasso regression were applied as the classification algorithms. The models achieved decent performance with respect to accuracy, precision, and recall on both the training and test data. Furthermore, bootstrap resampling was employed to estimate confidence intervals for the accuracy metric, providing a measure of the model's stability. The confidence intervals obtained through resampling provide valuable insights into the variability of the model's performance. Based on the results, it can be concluded that both logistic regression and KNN models performed similarly well in terms of accuracy, precision, and recall. However, the Lasso regression model showed slightly lower performance in terms of accuracy but achieved a higher F1-score. Overall, this project demonstrates the effectiveness of machine learning models in email classification tasks and highlights the importance of evaluating model performance using appropriate evaluation metrics and estimating confidence intervals to assess the model's stability.

Source code:

Outline:

1. clean the data
2. build feature vectors
3. use OLS, logistic regression,knn and losso logistic to training the data and make predictions
4. compare these models and discuss

In [76]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
```

Read in the dataset

In [40]:

```
raw_mail_data = pd.read_csv('mail_data.csv')
print(raw_mail_data)
     Category                                            Message
0         ham  Go until jurong point, crazy.. Available only ...
1         ham                      Ok lar... Joking wif u oni...
2        spam  Free entry in 2 a wkly comp to win FA Cup fina...
3         ham  U dun say so early hor... U c already then say...
4         ham  Nah I don't think he goes to usf, he lives aro...
...       ...                                                ...
5567     spam  This is the 2nd time we have tried 2 contact u...
5568      ham               Will ü b going to esplanade fr home?
5569      ham  Pity, * was in mood for that. So...any other s...
5570      ham  The guy did some bitching but I acted like i'd...
5571      ham                         Rofl. Its true to its name

[5572 rows x 2 columns]
```

Preprocesse all the datasets

In [41]:

```
# I am removing all the non-text email
mail_data = raw_mail_data.where((pd.notnull(raw_mail_data)),'')
```

Encoding the label

In [45]:

```
# label spam mail as 0;  ham mail as 1;
mail_data.loc[mail_data['Category'] == 'spam', 'Category',] = 0
mail_data.loc[mail_data['Category'] == 'ham', 'Category',] = 1
```

In [46]:

```
# separating the data as texts and label
X = mail_data['Message']
Y = mail_data['Category']
```

In [47]:

```
print (X)
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
```

```
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                              ...
5567    This is the 2nd time we have tried 2 contact u...
5568                 Will ü b going to esplanade fr home?
5569    Pity, * was in mood for that. So...any other s...
5570    The guy did some bitching but I acted like i'd...
5571                               Rofl. Its true to its name
Name: Message, Length: 5572, dtype: object
```

In [48]:

```
print (Y)
0       0
1       0
2       1
3       0
4       0
       ..
5567    1
5568    0
5569    0
5570    0
5571    0
Name: Category, Length: 5572, dtype: object
```

In [49]:

In [50]:

```
# Spliting the dataset into training/test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=3)
print(X.shape)
print(X_train.shape)
print(X_test.shape)
(5572,)
(4457,)
(1115,)
```

In [51]:

```
# transform the text data to feature vectors that can be used as input to the
Logistic regression
feature_extraction = TfidfVectorizer(min_df = 1, stop_words='english',
lowercase='True')
X_train_features = feature_extraction.fit_transform(X_train)
X_test_features = feature_extraction.transform(X_test)

# convert Y_train and Y_test values as integers
Y_train = Y_train.astype('int')
Y_test = Y_test.astype('int')
```

In [53]:

```
print(X_train)
3075                 Don know. I did't msg him recently.
1787    Do you know why god created gap between your f...
1614                        Thnx dude. u guys out 2nite?
4304                                     Yup i'm free...
```

```
3266    44 7732584351, Do you want a New Nokia 3510i c...
                                ...
789     5 Free Top Polyphonic Tones call 087018728737,...
968     What do u want when i come back?.a beautiful n...
1667    Guess who spent all last night phasing in and ...
3321    Eh sorry leh... I din c ur msg. Not sad alread...
1688    Free Top ringtone -sub to weekly ringtone-get ...
Name: Message, Length: 4457, dtype: object
```

In [54]:

```
print(X_train_features)
  (0, 5413)	0.6198254967574347
  (0, 4456)	0.4168658090846482
  (0, 2224)	0.413103377943378
  (0, 3811)	0.34780165336891333
  (0, 2329)	0.38783870336935383
  (1, 4080)	0.18880584110891163
  (1, 3185)	0.29694482957694585
  (1, 3325)	0.31610586766078863
  (1, 2957)	0.3398297002864083
  (1, 2746)	0.3398297002864083
  (1, 918)	0.22871581159877646
  (1, 1839)	0.2784903590561455
  (1, 2758)	0.3226407885943799
  (1, 2956)	0.33036995955537024
  (1, 1991)	0.33036995955537024
  (1, 3046)	0.2503712792613518
  (1, 3811)	0.17419952275504033
  (2, 407)	0.509272536051008
  (2, 3156)	0.4107239318312698
  (2, 2404)	0.45287711070606745
  (2, 6601)	0.6056811524587518
  (3, 2870)	0.5864269879324768
  (3, 7414)	0.8100020912469564
  (4, 50)	0.23633754072626942
  (4, 5497)	0.15743785051118356
  :	:
  (4454, 4602)	0.2669765732445391
  (4454, 3142)	0.32014451677763156
  (4455, 2247)	0.37052851863170466
  (4455, 2469)	0.35441545511837946
  (4455, 5646)	0.33545678464631296
  (4455, 6810)	0.29731757715898277
  (4455, 6091)	0.23103841516927642
  (4455, 7113)	0.30536590342067704
  (4455, 3872)	0.3108911491788658
  (4455, 4715)	0.30714144758811196
  (4455, 6916)	0.19636985317119715
  (4455, 3922)	0.31287563163368587
  (4455, 4456)	0.24920025316220423
  (4456, 141)	0.292943737785358
  (4456, 647)	0.30133182431707617
  (4456, 6311)	0.30133182431707617
```

```
  (4456, 5569)	0.4619395404299172
  (4456, 6028)	0.21034888000987115
  (4456, 7154)	0.24083218452280053
  (4456, 7150)	0.3677554681447669
  (4456, 6249)	0.17573831794959716
  (4456, 6307)	0.2752760476857975
  (4456, 334)	0.2220077711654938
  (4456, 5778)	0.16243064490100795
  (4456, 2870)	0.31523196273113385
```

Logistic Regression

In [56]:

```
from sklearn.linear_model import LogisticRegression
```

Implement logistic regression

In [66]:

```
# training the logistic regression witn model on the training dataset
model = LogisticRegression()
X_train_features_array = X_train_features.toarray()
model.fit(X_train_features_array, Y_train)
```

Out[66]:

```
LogisticRegression()
```

In [87]:

```
from sklearn.utils import resample

# Number of bootstrap iterations
n_iterations = 1000

# Initialize lists to store metric scores from each iteration
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

# Perform bootstrap resampling and evaluate metrics
for _ in range(n_iterations):
    X_boot, y_boot = resample(X_test_features, y_test)
    y_pred = model.predict(X_boot)
    accuracy = accuracy_score(y_boot, y_pred)
    accuracy_scores.append(accuracy)


confidence_interval = 0.95
alpha = (1 - confidence_interval) / 2

accuracy_ci = np.percentile(accuracy_scores, [alpha * 100, (1 - alpha) * 
100])
print("Accuracy Confidence Interval:", accuracy_ci)
Accuracy Confidence Interval: [0.02421525 0.04484305]
```

In [82]:

```
# Make predictions on training data
train_predictions = logreg.predict(X_train_features)
train_accuracy = accuracy_score(y_train, train_predictions)
train_precision = precision_score(y_train, train_predictions)
train_recall = recall_score(y_train, train_predictions)

# Print the evaluation metrics
print('Training Accuracy: {:.4f}'.format(train_accuracy))
print('Training Precision: {:.4f}'.format(train_precision))
print('Training Recall: {:.4f}'.format(train_recall))

# prediction on test data
prediction_on_test_data = model.predict(X_test_features)
accuracy_on_test_data = accuracy_score(Y_test, prediction_on_test_data)
precision_on_test_data = precision_score(y_test, prediction_on_test_data)
recall_on_test_data = recall_score(y_test, prediction_on_test_data)

print('Test Accuracy: {:.4f}'.format(test_accuracy))
print('Test Precision: {:.4f}'.format(test_precision))
print('Test Recall: {:.4f}'.format(test_recall))
```

```
Training Accuracy: 0.9670
Training Precision: 0.9643
Training Recall: 0.9990
Test Accuracy: 0.9094
Test Precision: 0.9048
Test Recall: 1.0000
```

detection system

In [60]:

```
input_mail = ["WINNER!! As a valued network customer you have been selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only."]

# convert text to feature vectors
input_data_features = feature_extraction.transform(input_mail)


# making prediction
prediction = model.predict(input_data_features)
print(prediction)


if (prediction[0]==1):
  print('Ham mail')

else:
  print('Spam mail')
```

```
[1]
Ham mail
```

Knn

In [74]:

```
# Train the KNN classifier
k = 5  # Number of neighbors to consider
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_features, y_train)

# Make predictions on training data
train_predictions = knn.predict(X_train_features)
train_accuracy = accuracy_score(y_train, train_predictions)
train_precision = precision_score(y_train, train_predictions)
train_recall = recall_score(y_train, train_predictions)

# Make predictions on test data
test_predictions = knn.predict(X_test_features)
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions)
test_recall = recall_score(y_test, test_predictions)

# Print the evaluation metrics
print('Training Accuracy: {:.4f}'.format(train_accuracy))
print('Training Precision: {:.4f}'.format(train_precision))
print('Training Recall: {:.4f}'.format(train_recall))
print('Test Accuracy: {:.4f}'.format(test_accuracy))
print('Test Precision: {:.4f}'.format(test_precision))
print('Test Recall: {:.4f}'.format(test_recall))
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
Training Accuracy: 0.9201
Training Precision: 0.9159
Training Recall: 0.9997
Test Accuracy: 0.9094
Test Precision: 0.9048
Test Recall: 1.0000
```

In [88]:

```
# Number of bootstrap iterations
n_iterations = 1000

# Initialize lists to store metric scores from each iteration
```

```
accuracy_scores = []

# Perform bootstrap resampling and evaluate metrics
for _ in range(n_iterations):
    X_boot, y_boot = resample(X_test_features, y_test)
    y_pred = knn.predict(X_boot)
    accuracy = accuracy_score(y_boot, y_pred)
    accuracy_scores.append(accuracy)


confidence_interval = 0.95
alpha = (1 - confidence_interval) / 2

accuracy_ci = np.percentile(accuracy_scores, [alpha * 100, (1 - alpha) *
100])
print("Accuracy Confidence Interval:", accuracy_ci)
Accuracy Confidence Interval: [0.89237668 0.92556054]
```

Lasso Regression

In [90]:

```
# Train the Lasso regression model
lasso = Lasso(alpha=0.1)  # Adjust alpha as per your requirement
lasso.fit(X_train_features, y_train)

# Make predictions on the test set
y_pred = lasso.predict(X_test_features)
y_pred_binary = (y_pred >= 0.5).astype(int)  # Convert probabilities to
binary predictions

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)


# Calculate the confidence intervals

n_iterations = 1000
accuracy_scores = []
for _ in range(n_iterations):
    X_boot, y_boot = resample(X_test_features, y_test)
    y_pred = lasso.predict(X_boot)
    y_pred_binary = (y_pred >= 0.5).astype(int)
```

```
    accuracy = accuracy_score(y_boot, y_pred_binary)
    accuracy_scores.append(accuracy)


confidence_interval = 0.95
alpha = (1 - confidence_interval) / 2

accuracy_ci = np.percentile(accuracy_scores, [alpha * 100, (1 - alpha) *
100])
print("Accuracy Confidence Interval:", accuracy_ci)
```

```
Accuracy: 0.8609865470852018
Precision: 0.8609865470852018
Recall: 1.0
F1-Score: 0.9253012048192771
Accuracy Confidence Interval: [0.83946188 0.88161435]
```

In [ ]: