Spam_email_detect

October 11, 2023

[48]:
```
import numpy as np
import pandas as pd
import string
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from collections import Counter
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, log_loss,␣
↪precision_score, classification_report, f1_score
```

[49]:
```
# Read the dataset full of spams and non-spams.
dataframe = pd.read_csv('spam.csv', encoding = 'latin1')
print(dataframe)
```

```
        v1                                                 v2 Unnamed: 2  \
0      ham  Go until jurong point, crazy.. Available only …        NaN
1      ham                      Ok lar… Joking wif u oni…        NaN
2     spam  Free entry in 2 a wkly comp to win FA Cup fina…        NaN
3      ham  U dun say so early hor… U c already then say…        NaN
4      ham  Nah I don't think he goes to usf, he lives aro…        NaN
…     …                                                …        …
5567  spam  This is the 2nd time we have tried 2 contact u…        NaN
5568   ham              Will Ì_ b going to esplanade fr home?        NaN
5569   ham  Pity, * was in mood for that. So…any other s…        NaN
5570   ham  The guy did some bitching but I acted like i'd…        NaN
5571   ham                         Rofl. Its true to its name        NaN

     Unnamed: 3 Unnamed: 4
0           NaN        NaN
1           NaN        NaN
2           NaN        NaN
3           NaN        NaN
4           NaN        NaN
```

```
…          …         …
5567        NaN        NaN
5568        NaN        NaN
5569        NaN        NaN
5570        NaN        NaN
5571        NaN        NaN

[5572 rows x 5 columns]
```

[50]:
```python
# Let's drop the unnececcessary columns and extract the data we want, which is␣
↪emails with some kind of content i.e not empty
dataframe = dataframe.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],axis=1)
dataframe.head()
```

[50]:
```
     v1                                                 v2
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…
```

[51]:
```python
# Let's also change 'v1' and 'v2' names for clearer set
dataframe.columns = ['Label', 'E-mail']
```

[52]:
```python
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Label   5572 non-null   object
 1   E-mail  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

[53]:
```python
# Let's check and get rid of duplicates if there are any
dataframe.duplicated().sum()
```

[53]: 403

[54]:
```python
dataframe = dataframe.drop_duplicates()
dataframe.shape
```

[54]: (5169, 2)

[55]:
```python
# Let's see how many spam- and non-spam messages we have (datapoints)
count = dataframe['Label'].value_counts()
```

[56]:
```python
count_of_non_spams = count['ham']
count_of_spams = count['spam']

print("Count of non-spams:", count_of_non_spams)
print("Count of spams:", count_of_spams)
```

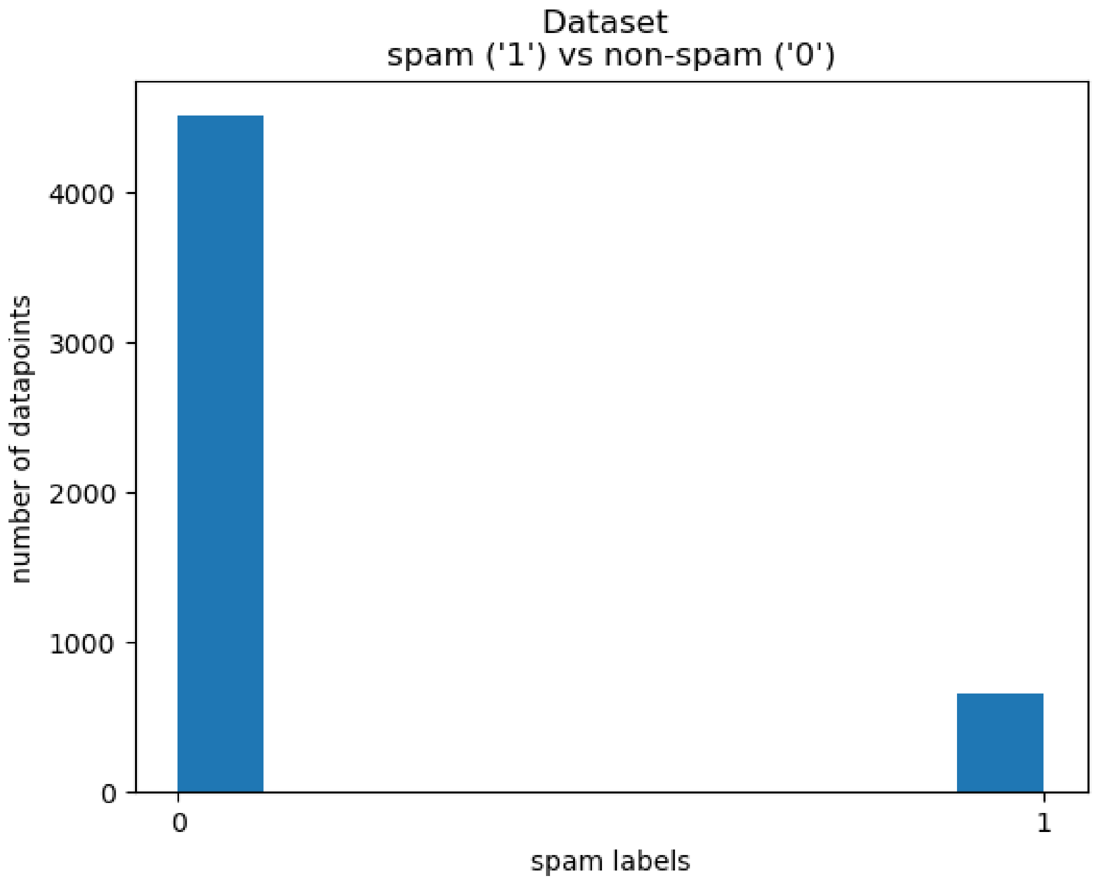
```
Count of non-spams: 4516
Count of spams: 653
```

[57]:
```python
# Let's change 'ham' and 'spam' into binary values, where 1 = spam and 0 =␣
↪non-spam
dataframe.loc[dataframe['Label'] == 'spam', 'Label',] = 1
dataframe.loc[dataframe['Label'] == 'ham', 'Label',] = 0
```

[58]:
```python
dataframe.head()
```

[58]:
```
  Label                                             E-mail
0     0  Go until jurong point, crazy.. Available only …
1     0                      Ok lar… Joking wif u oni…
2     1  Free entry in 2 a wkly comp to win FA Cup fina…
3     0  U dun say so early hor… U c already then say…
4     0  Nah I don't think he goes to usf, he lives aro…
```

[59]:
```python
fig, ax = plt.subplots()
ax.hist(dataframe['Label'])
ax.set_title('Dataset \nspam (\'1\') vs non-spam (\'0\')')
ax.set_xlabel("spam labels")
ax.set_ylabel('number of datapoints')
ax.set_xticks([0,1])
plt.show()
```



[60]:
```
def remove_special_characters(text):
    # Remove punctuation using string.punctuation and str.translate
    translator = str.maketrans('', '', string.punctuation)
    text = text.translate(translator)
    return text.lower()
```

[61]:
```
dataframe['E-mail'] = dataframe['E-mail'].apply(remove_special_characters)
```

[62]:
```
dataframe.head()
```

[62]:
```
   Label                                             E-mail
0      0  go until jurong point crazy available only in …
1      0                            ok lar joking wif u oni
2      1  free entry in 2 a wkly comp to win fa cup fina…
3      0        u dun say so early hor u c already then say
4      0  nah i dont think he goes to usf he lives aroun…
```

[63]:
```
# Let's seperate the columns of the data into their own parameters:
X = dataframe['E-mail']
```

```
y = dataframe['Label']
```

[64]:
```
print(X, y)
```

```
0       go until jurong point crazy available only in …
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 dont think he goes to usf he lives aroun…
                                  …
5567    this is the 2nd time we have tried 2 contact u…
5568                  will ì b going to esplanade fr home
5569    pity  was in mood for that soany other suggest…
5570    the guy did some bitching but i acted like id …
5571                          rofl its true to its name
Name: E-mail, Length: 5169, dtype: object 0       0
1       0
2       1
3       0
4       0
       ..
5567    1
5568    0
5569    0
5570    0
5571    0
Name: Label, Length: 5169, dtype: object
```

[65]:
```
# Let's make our sets. Training, valid and test with 7:1,5:1,5
# Split the data into "train" (70%) and "temporary" (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=10)

# Split the "temp" set into "validation" (50%) and "test" (50%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=10)
```

[66]:
```
# Let's check the shapes of the original-, test-, val- and train data's X and y␣
 ↪to see that we have succeeded in the grouping
print(X.shape)
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(5169,)
(3618,)
(775,)
```

```
(776,)
```

[67]:
```
print(y.shape)
print(y_train.shape)
print(y_val.shape)
print(y_test.shape)
```

```
(5169,)
(3618,)
(775,)
(776,)
```

[68]:
```
# Since the lengths match, the grouping and distribution has succeeded
```

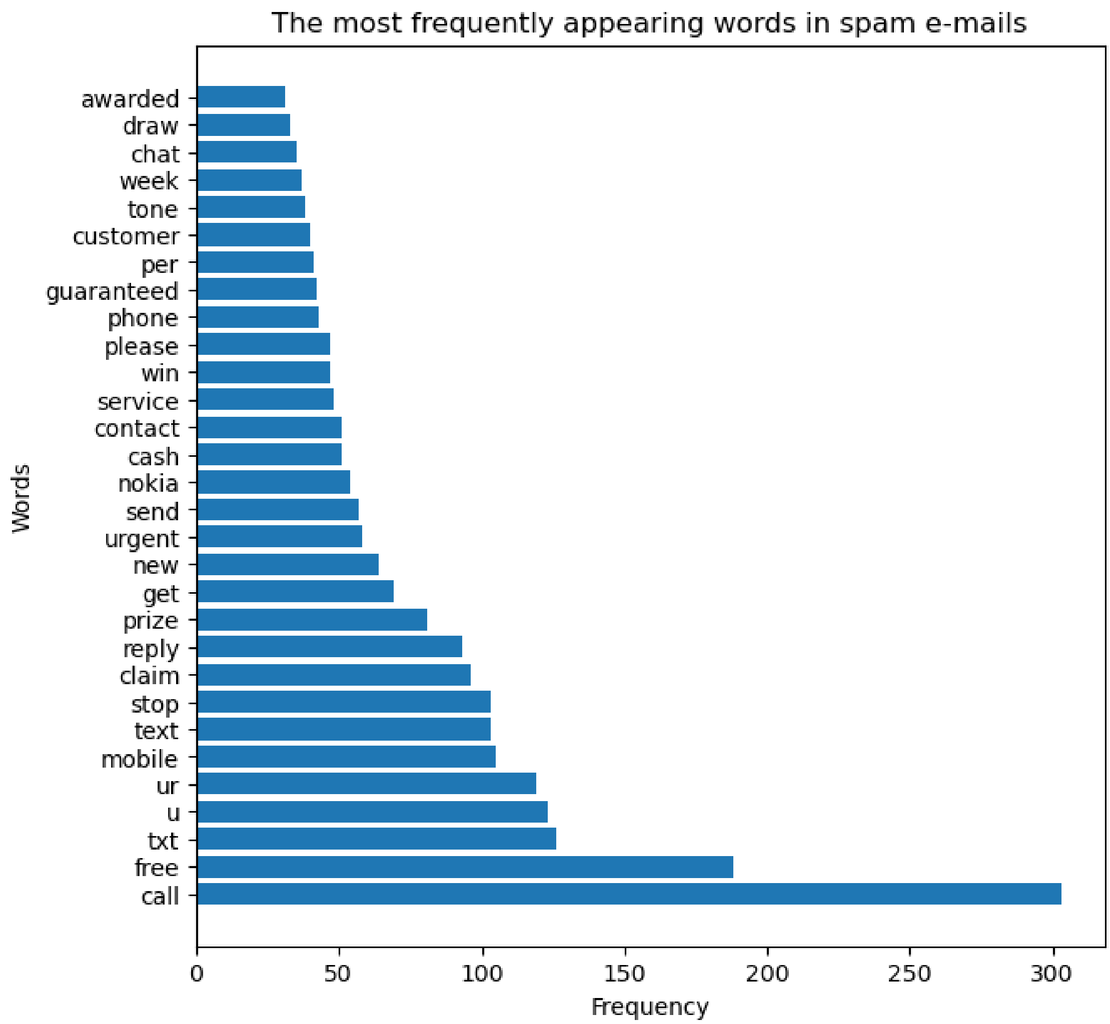
[69]:
```
# Let's now extract the features from data's. This is where TfidfVectorizer
 ↪(Term Frequency - Inverse Document Frequency) comes in.
# This allows us to assess the quantity and quality of words in emails and help
 ↪us determine the spams.
# min_df ensures, that extremely infrequent words are cut-off of our equation,
 ↪which makes these words more unique, making them less likely a spam
# Frequently appearing words can be assumed to be part of a spam message
features = TfidfVectorizer(min_df = 1, stop_words = 'english', lowercase = True)
X_train = features.fit_transform(X_train)
X_test = features.transform(X_test)
X_val = features.transform(X_val)

y_val = y_val.astype(int)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
```

[70]:
```
# Let's see what kind of words appear most frequently in scams
spam_most_used_words = []
stop_words = set(stopwords.words('english'))
for sentence in dataframe[dataframe['Label'] == 1]['E-mail'].tolist():
    words = sentence.split()
    filtered_words = [word for word in words if word.lower() not in stop_words
 ↪and word.isalpha()]
    spam_most_used_words.extend(filtered_words)

filter_df = pd.DataFrame(Counter(spam_most_used_words).most_common(30))
```

[71]:
```
plt.figure(figsize=(7, 7))
plt.barh(filter_df[0], filter_df[1])
plt.xlabel('Frequency')
plt.ylabel('Words')
plt.title('The most frequently appearing words in spam e-mails')
plt.show()
```



[72]:
```python
print(X_train)
```

```
  (0, 5190)	0.4521241713824104
  (0, 5793)	0.3721282557866962
  (0, 3353)	0.1896790597744401
  (0, 1766)	0.4152584200119519
  (0, 1545)	0.4152584200119519
  (0, 4899)	0.32458962544662484
  (0, 3836)	0.22214069331416178
  (0, 5930)	0.34862892281720176
  (1, 6762)	0.2927895555121226
  (1, 2141)	0.2927895555121226
  (1, 3704)	0.2927895555121226
  (1, 7021)	0.2549505306455541
  (1, 5676)	0.2927895555121226
  (1, 5574)	0.2927895555121226
```

```
(1, 6834)     0.2927895555121226
(1, 1213)     0.27882429728000246
(1, 2895)     0.2927895555121226
(1, 5482)     0.2927895555121226
(1, 1384)     0.2927895555121226
(1, 7037)     0.2927895555121226
(2, 1470)     0.4966035959316888
(2, 3151)     0.4652464853142722
(2, 3024)     0.44545358337764496
(2, 7095)     0.4874947093614859
(2, 2248)     0.3175698187973922
:     :
(3614, 3237)  0.47750925337263866
(3614, 3737)  0.5333301936875326
(3614, 6544)  0.4117250459123929
(3614, 1104)  0.43184393400337695
(3614, 6514)  0.36268046635799
(3615, 6512)  0.3956700211092583
(3615, 4978)  0.3767976470253049
(3615, 6967)  0.30188635690203547
(3615, 1790)  0.31632251851959775
(3615, 7230)  0.33736016680322134
(3615, 1355)  0.32075873098598895
(3615, 5510)  0.24584744086271954
(3615, 2896)  0.29340019975909737
(3615, 1036)  0.23814357106780987
(3615, 6372)  0.23509034545306035
(3615, 4609)  0.18727237215508966
(3616, 1999)  0.664184765345535
(3616, 5468)  0.578348015438433
(3616, 6893)  0.3392430085479584
(3616, 3973)  0.3305848630422339
(3617, 2593)  0.5759501518340481
(3617, 2487)  0.5013270511240697
(3617, 7032)  0.4202853532449395
(3617, 2992)  0.3669515613171994
(3617, 2039)  0.3250528940160777
```

[73]:
```
def generate_confusion_matrix(y_true, y_pred, title_main):
    # visualize the confusion matrix
    ax = plt.subplot()
    c_mat = confusion_matrix(y_true, y_pred)
    sns.heatmap(c_mat, annot=True, fmt='g', ax=ax)

    ax.set_xlabel('Predicted labels', fontsize=15)
    ax.set_ylabel('True labels', fontsize=15)
    ax.set_title(title_main, fontsize=15)
```

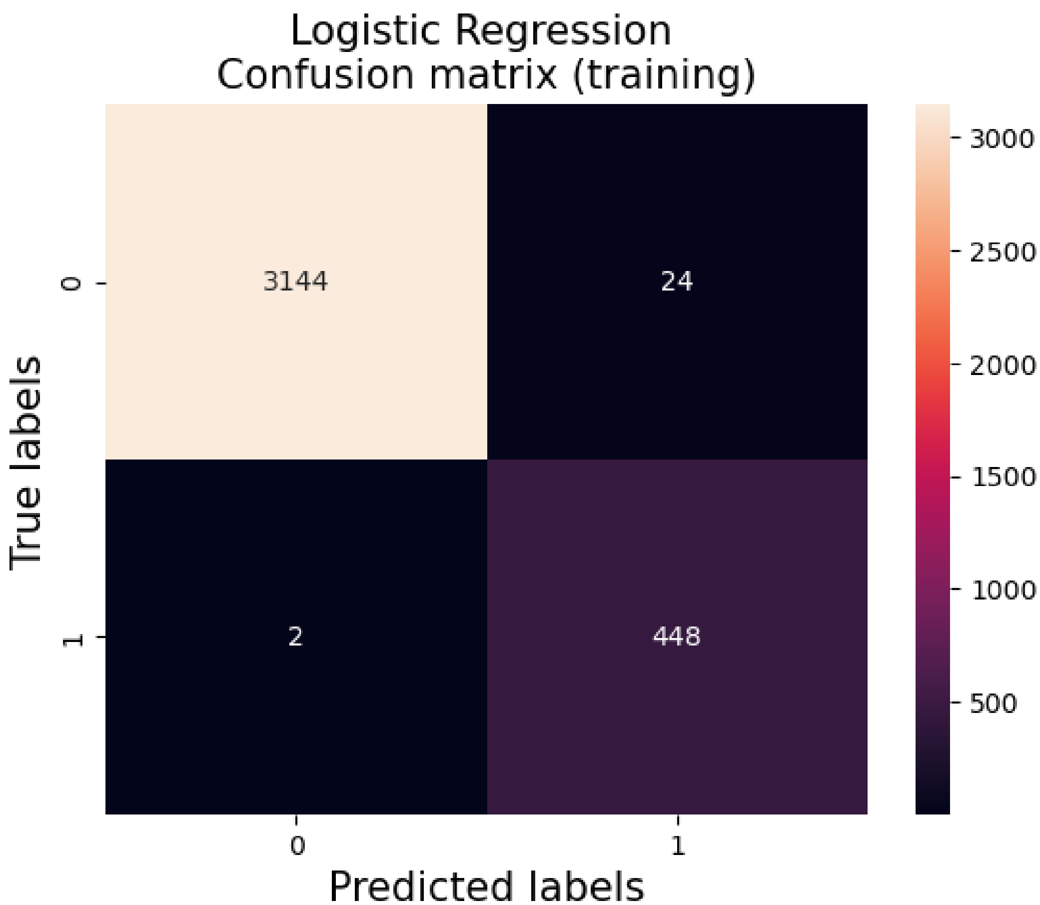
[74]:
```
# Let's train our model and then do validation
clf_1 = LogisticRegression(solver = 'liblinear', class_weight = 'balanced').
  ↪fit(X_train, y_train)
```

[75]:
```
y_pred_train_lr = clf_1.predict(X_train)
tr_error = log_loss(y_train,y_pred_train_lr)
accuracy = accuracy_score(y_train, y_pred_train_lr)
precision = precision_score(y_train, y_pred_train_lr)
```

[76]:
```
print("Error:",tr_error)
print("Accuracy:",accuracy)
print("Precision:",precision)
```

```
Error: 0.2590201736089129
Accuracy: 0.9928137092316197
Precision: 0.9491525423728814
```

[77]:
```
generate_confusion_matrix(y_train, y_pred_train_lr, 'Logistic Regression␣
  ↪\nConfusion matrix (training)' )
```

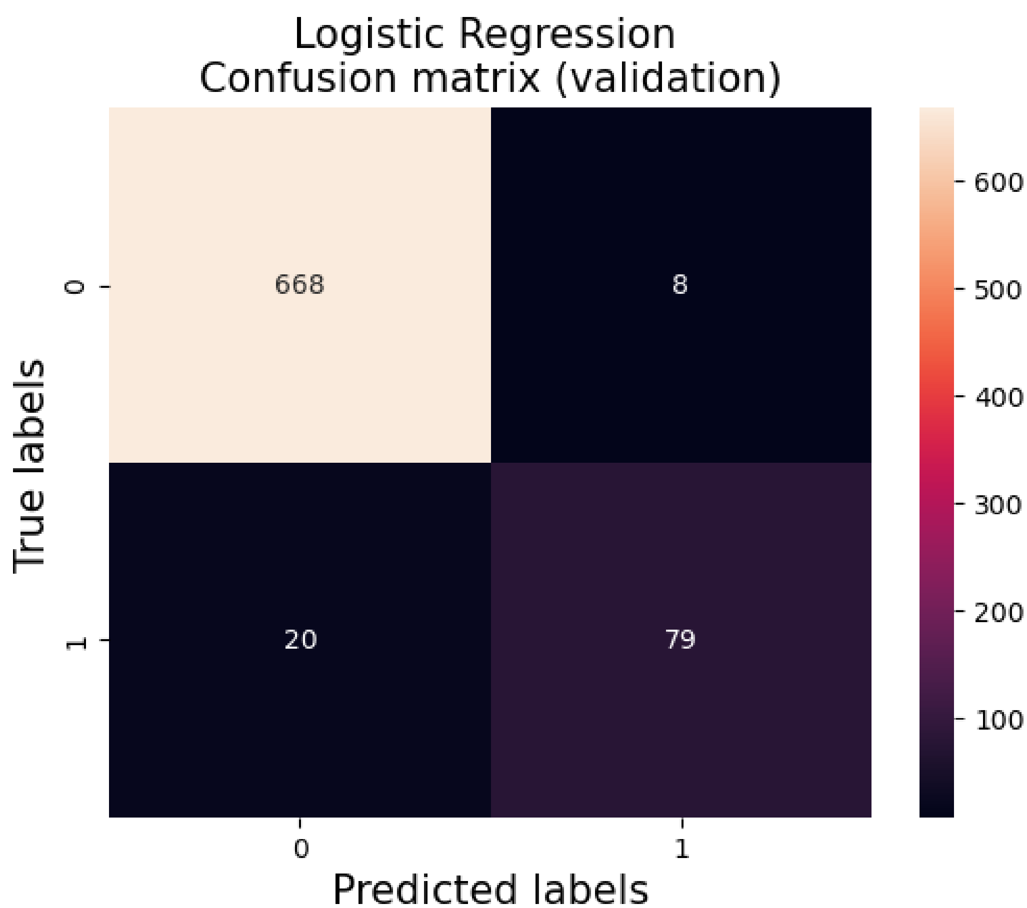


[78]:
```python
# Let's use our validation set
y_pred_val_lr = clf_1.predict(X_val)
val_error = log_loss(y_val ,y_pred_val_lr)
accuracy = accuracy_score(y_val, y_pred_val_lr)
precision = precision_score(y_val, y_pred_val_lr)
```

[79]:
```python
print("Error:",val_error)
print("Accuracy:",accuracy)
print("Precision:",precision)
```

```
Error: 1.3022223159939101
Accuracy: 0.9638709677419355
Precision: 0.9080459770114943
```

[80]:
```python
generate_confusion_matrix(y_val, y_pred_val_lr, 'Logistic Regression␣
↪\nConfusion matrix (validation)' )
```

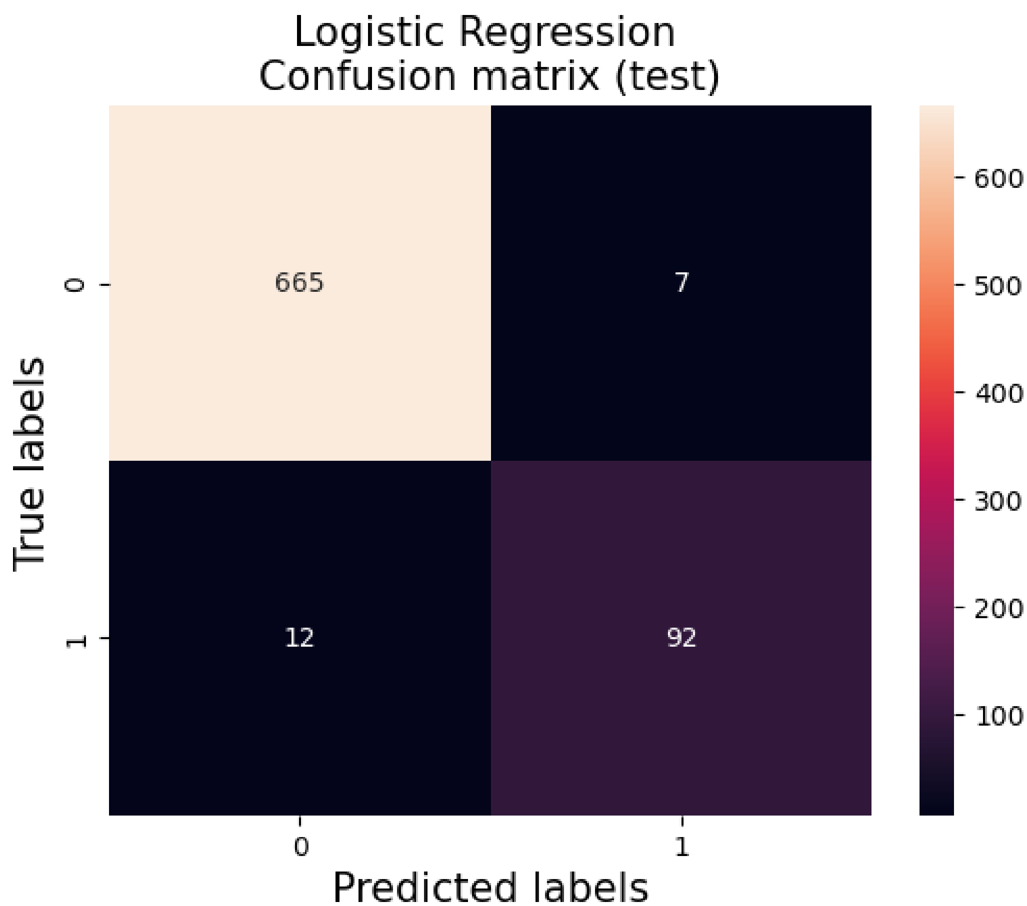


[81]:
```
# Let's also use our test set
y_pred_tst_lr = clf_1.predict(X_test)
tst_error = log_loss(y_test,y_pred_tst_lr)
accuracy = accuracy_score(y_test, y_pred_tst_lr)
precision = precision_score(y_test, y_pred_tst_lr)
```

[82]:
```
print("Error:",tst_error)
print("Accuracy:",accuracy)
print("Precision:",precision)
```

```
Error: 0.8825121319500335
Accuracy: 0.9755154639175257
Precision: 0.9292929292929293
```

[83]:
```
generate_confusion_matrix(y_test, y_pred_tst_lr, 'Logistic Regression␣
↪\nConfusion matrix (test)')
```



[84]:
```
# Let's see how well can we do with another ML-method, such as random forest␣
 ↪classifier
# Let's find the optimal parameters to best estimation
n_estimators = [10, 50, 100, 200]
max_depth = [None, 2, 3, 5]

# Create a parameter grid: map the parameter names to the values that should be␣
 ↪searched
param_grid = {'n_estimators': n_estimators, 'max_depth': max_depth}

# Instantiate the grid
grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)

# Fit the grid with data
grid.fit(X_train, y_train)

# Examine the best model
print(grid.best_score_)
print(grid.best_params_)
```

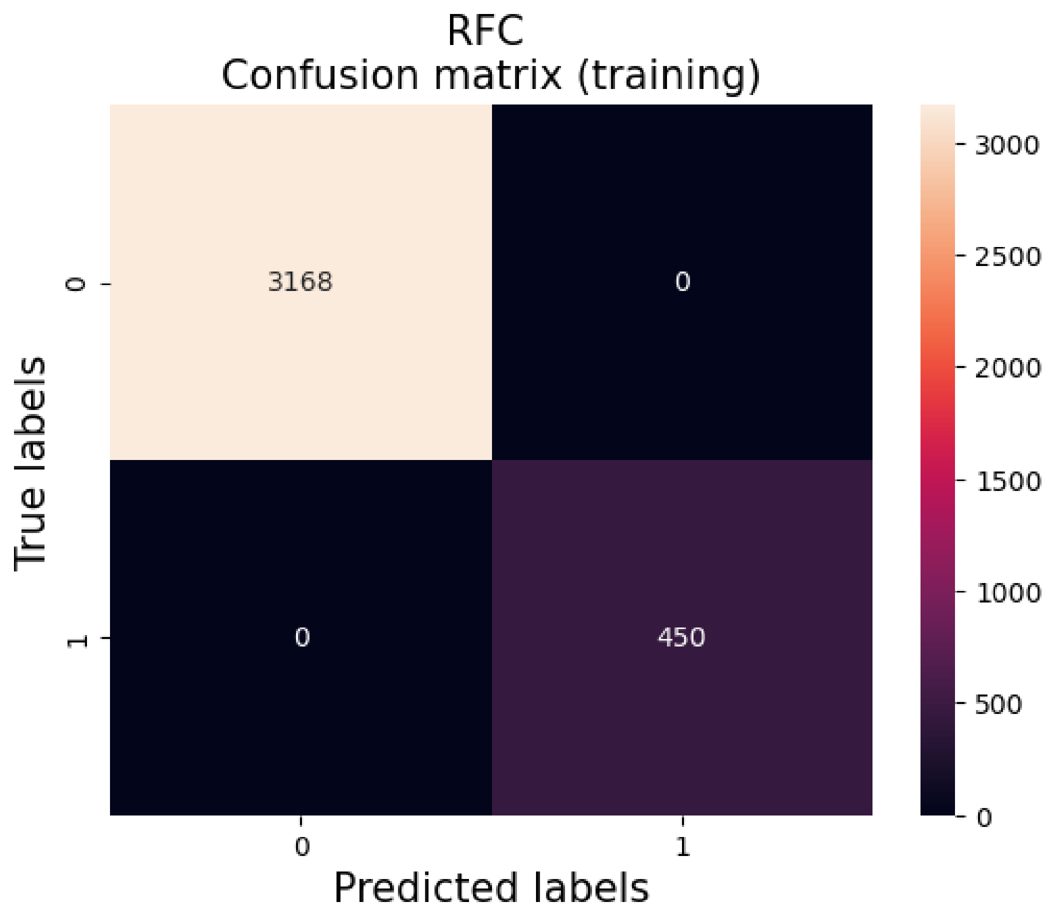
```
0.9690443440850356
{'max_depth': None, 'n_estimators': 100}
100
```

[85]:
```
# Now let's build our model
rfc = RandomForestClassifier(n_estimators = grid.best_params_['n_estimators'],␣
 ↪random_state = 2, criterion = 'gini', max_depth = grid.
 ↪best_params_['max_depth'], class_weight = 'balanced')
# Train our model and see the performance
rfc.fit(X_train, y_train)
y_pred_train_rfc = rfc.predict(X_train)
y_accuracy_train = accuracy_score(y_train, y_pred_train_rfc)
precision_train = precision_score(y_train, y_pred_train_rfc)
```

[86]:
```
print("Accuracy: ", y_accuracy_train)
print("Precision: ", precision_train)
```

```
Accuracy:  1.0
Precision:  1.0
```

[87]:
```
generate_confusion_matrix(y_train, y_pred_train_rfc, 'RFC \nConfusion matrix␣
 ↪(training)')
```

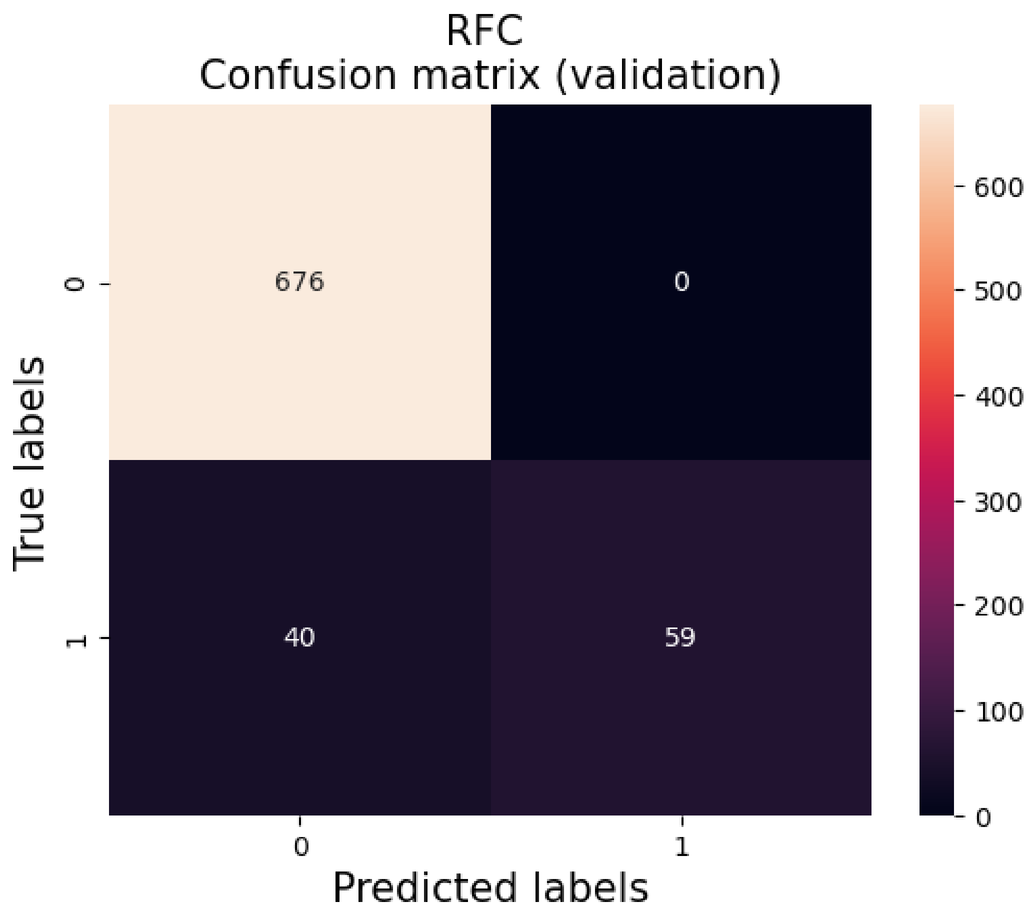


[88]:
```
# How well does the validation set perform
y_pred_val_rfc = rfc.predict(X_val)
y_accuracy_val = accuracy_score(y_val, y_pred_val_rfc)
precision_val = precision_score(y_val, y_pred_val_rfc)
```

[89]:
```
print("Accuracy: ", y_accuracy_val)
print("Precision: ", precision_val)
```

```
Accuracy:  0.9483870967741935
Precision:  1.0
```

[90]:
```
generate_confusion_matrix(y_val, y_pred_val_rfc, 'RFC \nConfusion matrix␣
↪(validation)')
```



[91]:

```
y_pred_tst_rfc = rfc.predict(X_test)
y_accuracy_tst = accuracy_score(y_test, y_pred_tst_rfc)
precision_tst = precision_score(y_test, y_pred_tst_rfc)
```

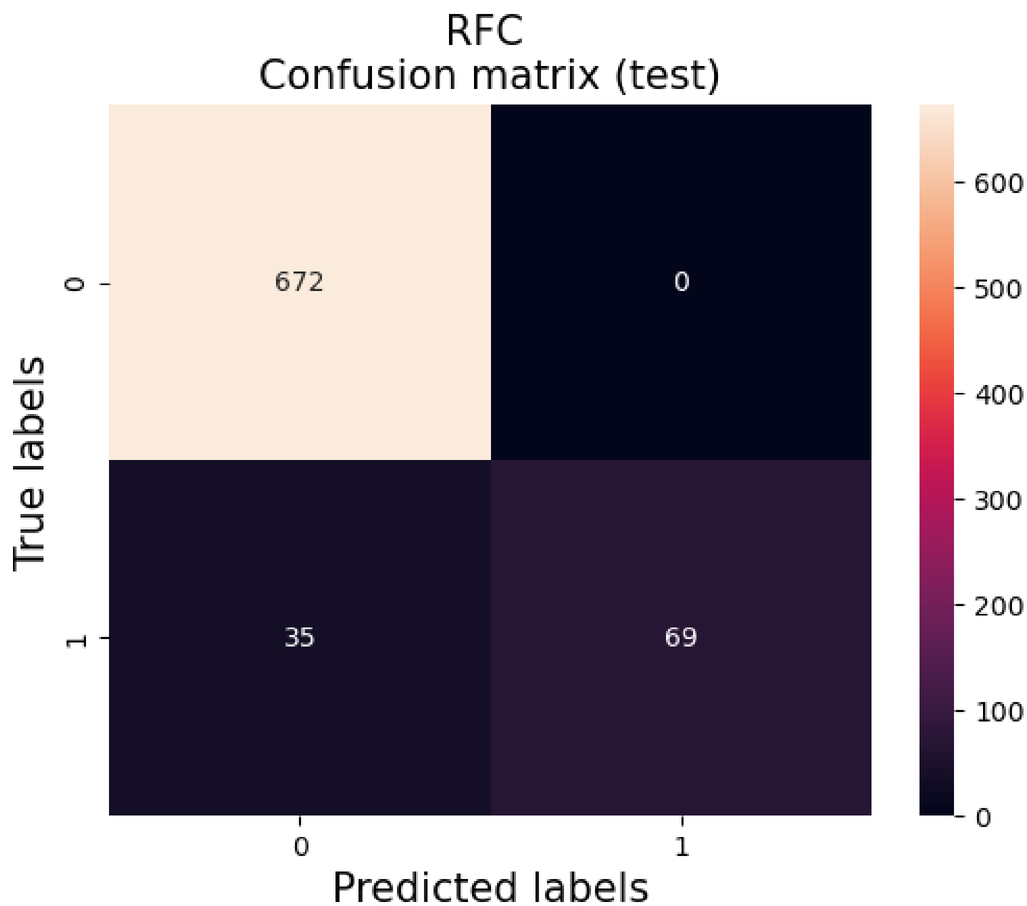
[92]:

```
print("Accuracy: ", y_accuracy_tst)
print("Precision: ", precision_tst)
```

```
Accuracy:  0.9548969072164949
Precision:  1.0
```

[93]:

```
generate_confusion_matrix(y_test, y_pred_tst_rfc, 'RFC \nConfusion matrix␣
↪(test)')
```



[94]:
```
print("\033[1m" + "Logistic Regression classification report of training␣
↪data\n", "\033[0m" + classification_report(y_train, y_pred_train_lr))
print("\033[1m" + "Logistic Regression classification report of validation␣
↪data\n", "\033[0m" + classification_report(y_val, y_pred_val_lr))
print("\033[1m" + "Logistic Regression classification report of test data\n",␣
↪"\033[0m" + classification_report(y_test, y_pred_tst_lr))
print("-----------------------------------------------------------------")
print("\033[1m" + "RFC classification report for training data\n", "\033[0m" +␣
↪classification_report(y_train, y_pred_train_rfc))
print("\033[1m" + "RCF classification report of validation data\n", "\033[0m" +␣
↪classification_report(y_val, y_pred_val_rfc))
print("\033[1m" + "RCF classification report of test data\n", "\033[0m" +␣
↪classification_report(y_test, y_pred_tst_rfc))
```

```
Logistic Regression classification report of training data

              precision    recall  f1-score   support

           0       1.00      0.99      1.00      3168
```

1	0.95	1.00	0.97	450
accuracy			0.99	3618
macro avg	0.97	0.99	0.98	3618
weighted avg	0.99	0.99	0.99	3618

Logistic Regression classification report of validation data

	precision	recall	f1-score	support
0	0.97	0.99	0.98	676
1	0.91	0.80	0.85	99
accuracy			0.96	775
macro avg	0.94	0.89	0.91	775
weighted avg	0.96	0.96	0.96	775

Logistic Regression classification report of test data

	precision	recall	f1-score	support
0	0.98	0.99	0.99	672
1	0.93	0.88	0.91	104
accuracy			0.98	776
macro avg	0.96	0.94	0.95	776
weighted avg	0.98	0.98	0.98	776

\---

RFC classification report for training data

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3168
1	1.00	1.00	1.00	450
accuracy			1.00	3618
macro avg	1.00	1.00	1.00	3618
weighted avg	1.00	1.00	1.00	3618

RCF classification report of validation data

	precision	recall	f1-score	support
0	0.94	1.00	0.97	676
1	1.00	0.60	0.75	99
accuracy			0.95	775
macro avg	0.97	0.80	0.86	775

```
weighted avg       0.95      0.95      0.94       775
```

RCF classification report of test data

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       672
           1       1.00      0.66      0.80       104

    accuracy                           0.95       776
   macro avg       0.98      0.83      0.89       776
weighted avg       0.96      0.95      0.95       776
```

[ ]:

```

```