


[Text Classification Data.csv](#)

```
import pandas as pd
import numpy as np
import seaborn as sb
from nltk.corpus import stopwords
import nltk
import math
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import LogisticRegression
```

```
# Input file name
input_file = "Text_Classification_Data.csv"
```

```
# Read data from csv into data variable
data = pd.read_csv(input_file, header = 0)
data.head()
```

	sms	label	
0	Go until jurong point, crazy.. Available only ...	0	
1	Ok lar... Joking wif u oni...\n	0	
2	Free entry in 2 a wkly comp to win FA Cup fina...	1	
3	U dun say so early hor... U c already then say...	0	
4	Nah I don't think he goes to usf, he lives aro...	0	

```
# Split training and testing data
X = data['sms']
y = data['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 500, test_size = 0.20, shuffle = True, stratify = y)
```

```
# convert to data frame
df_y = pd.DataFrame(y, columns=['label'])
```

```
# Create a graph showing the distribution of the target classes using seaborn
sb.catplot(x="label", kind='count', data=df_y)
```

```
<seaborn.axisgrid.FacetGrid at 0x7ff78ba4fdf0>
```

```
5000 |
```

This data set holds SMS labeled messages that have been collected for mobile phone spam research. Data labeled as '0' is Not Spam and data labeled as '1' is Spam.

The model should be able to predict whether a piece of text is Not Spam or is Spam.

```
|
```

▼ Naive Bayes

```
|
```

```
# Text preprocessing
nltk.download('stopwords')
stopwords = list(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words = stopwords)

# Use the tfidf vectorizer
X_train = vectorizer.fit_transform(X_train) #fit and transform the train data
X_test = vectorizer.transform(X_test) #transform the test data

# Perform Multinomial Naive Bayes
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# Calculate prior_p value
prior_p = sum(y_train == 1)/len(y_train)
print('prior spam:', prior_p, 'log of prior:', math.log(prior_p))

# Check prior value
naive_bayes.class_log_prior_[1]

# Plot log values
naive_bayes.feature_log_prob_

# Calculate predictions on testing data
pred = naive_bayes.predict(X_test)

# Print out confusion matrix
print(confusion_matrix(y_test, pred))

# Calculate baseline accuracy for guessing 'Not Spam' for everything
baseline = y_test[y_test == 0].shape[0] / y_test.shape[0]
print('baseline accuracy score: ', baseline)

# Print accuracy, precision, recall, and f1 scores
print('accuracy score: ', accuracy_score(y_test, pred))

print('\nprecision score (not spam): ', precision_score(y_test, pred, pos_label = 0))
print('precision score (spam): ', precision_score(y_test, pred))

print('\nrecall score: (not spam)', recall_score(y_test, pred, pos_label = 0))
print('recall score: (spam)', recall_score(y_test, pred))

print('\nf1 score: ', f1_score(y_test, pred))

# Display classification report
print(classification_report(y_test, pred))

# Print total number of 'Not Spam' data
print('spam size in test data:', y_test[y_test == 0].shape[0])
print('test size: ', len(y_test))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
prior spam: 0.13411078717201166 log of prior: -2.009089050676845
[[966  0]
 [ 25 124]]
baseline accuracy score: 0.8663677130044843
accuracy score: 0.9775784753363229

precision score (not spam): 0.9747729566094854
precision score (spam): 1.0

recall score: (not spam) 1.0
recall score: (spam) 0.8322147651006712
```

```
f1 score: 0.9084249084249084
      precision    recall  f1-score   support

         0         0.97      1.00      0.99      966
         1         1.00      0.83      0.91      149

 accuracy
macro avg      0.99      0.92      0.95      1115
weighted avg    0.98      0.98      0.98      1115

spam size in test data: 966
test size: 1115
```

▼ Logistic Regression

```
# tfidf vectorizer
vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=50000, min_df=2)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 500, test_size = 0.20, shuffle = True, stratify = y)
X_train = vectorizer.fit_transform(X_train) # fit and transform the train data
X_test = vectorizer.transform(X_test)      # transform only the test data

# Perform logistic regression on training data
clf = LogisticRegression(C=2.5, n_jobs=4, solver='lbfgs', random_state=30, verbose=1)
clf.fit(X_train, y_train)

# Generate predictions and confusion matrix
pred = clf.predict(X_test)
print(confusion_matrix(y_test, pred))

# Calculate accuracy of predictions
print(accuracy_score(y_test, pred))

print(precision_score(y_test, pred))

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[[965   1]
 [ 20 129]]
0.9811659192825112
0.9923076923076923
[Parallel(n_jobs=4)]: Done   1 out of   1 | elapsed:   1.0s finished
```

▼ Neural Network

```
# Redefine values
vectorizer = TfidfVectorizer(stop_words = stopwords, binary = True)

X = vectorizer.fit_transform(data.sms)
y = data.label

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 500, test_size = 0.2, shuffle = True, stratify = y)

classifier = MLPClassifier(solver = 'lbfgs', alpha = 1e-5, hidden_layer_sizes = (19, 2), random_state = 1)
classifier.fit(X_train, y_train)
pred = classifier.predict(X_test)

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

accuracy score: 0.9802690582959641
precision score: 0.950354609929078
recall score: 0.8993288590604027
f1 score: 0.9241379310344828
```

Analysis on Performance

All three classification approaches performed very well, with the lowest accuracy score being a .9776 from Naïve Bayes. The next best was the Neural Network with an accuracy score of .9802, followed closely by Logistic Regression, which had an accuracy score of .9812.

Generally, classification algorithms like Logistic Regression and Naïve Bayes are susceptible to overfitting and underfitting. This can be seen in Naïve Bayes performance compared to the others. Logistic Regression performed surprisingly well, however. This is likely because the dataset

is not complex enough to draw out the weaknesses of Logistic Regression as much. I believe the Neural Network did not perform better than the other algorithms because I used a relatively small data set. Neural Networks often perform better given a larger set of data and, when the data is more complex, it is able to shine in comparison to simpler algorithms like Logistic Regression and Naïve Bayes.

[Colab paid products](#) - [Cancel contracts here](#)

✓ 2s completed at 4:14 PM

