Spam Detection

Author: *Cennen Del Rosario*

Date: *August 19, 2020*

Read in text

In [1]:
```
import pandas as pd
import re
import string
import nltk
from matplotlib import pyplot as plt
pd.set_option('display.max_colwidth', 100)

stopwords = nltk.corpus.stopwords.words('english')
ps = nltk.PorterStemmer()
wn = nltk.WordNetLemmatizer()

data = pd.read_csv("SMSSpamCollection.tsv", sep='\t')
data.columns = ['label', 'body_text']
```

Text Cleaning and Lemmatization

In [2]:
```
def clean_text(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    tokens = re.split('\W+', text)
    #text = " ".join([wn.lemmatize(word) for word in tokens if word not in stopwords])
    text = [wn.lemmatize(word) for word in tokens if word not in stopwords]
    return text

data['cleaned_text'] = data['body_text'].apply(lambda x: clean_text(x))
```

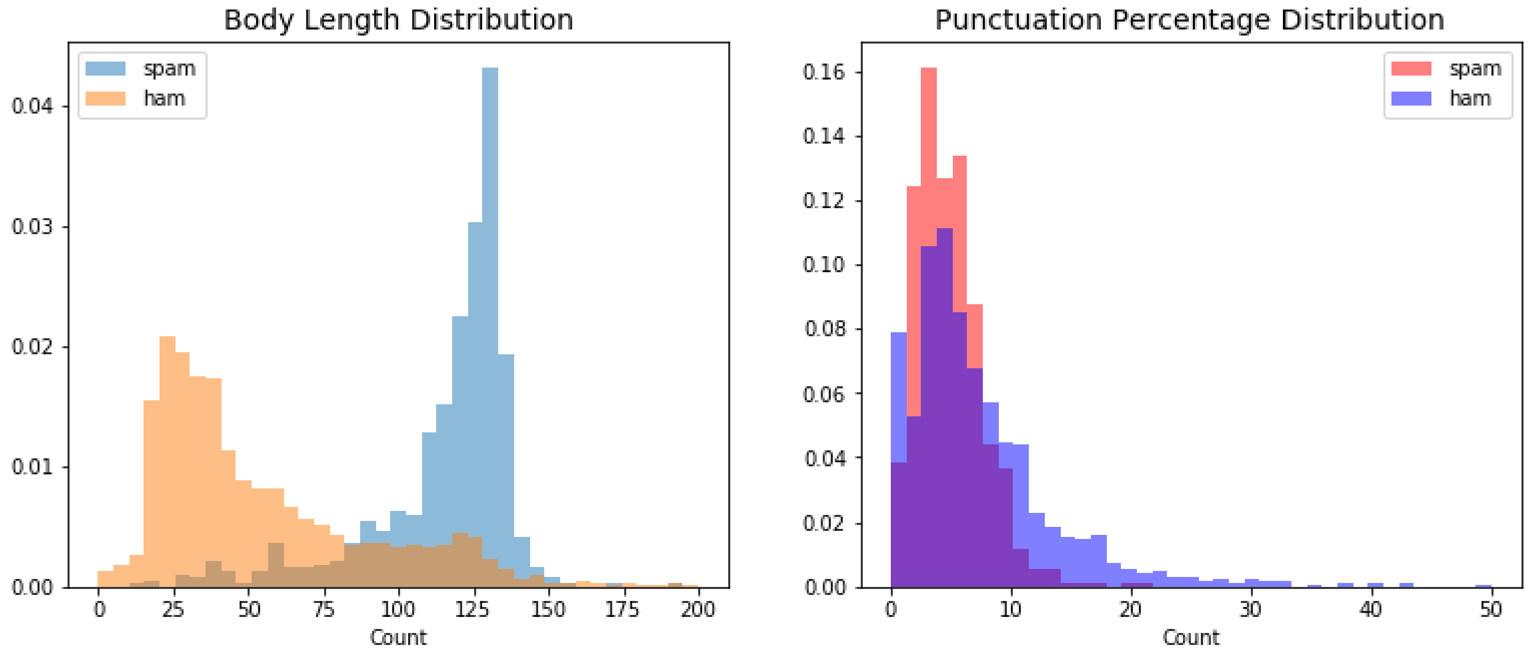
Feature Engineering

In [3]:
```
import string

def count_punct(text):
    count = sum([1 for char in text if char in string.punctuation])
    return round(count/(len(text) - text.count(" ")), 3)*100

data['body_len'] = data['body_text'].apply(lambda x: len(x) - x.count(" "))
data['punct%'] = data['body_text'].apply(lambda x: count_punct(x))
data.head()
```

Out[3]:

	label	body_text	cleaned_text	body_len	punct%
0	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ...	[free, entry, 2, wkly, comp, win, fa, cup, final, tkts, 21st, may, 2005, text, fa, 87121, receiv...	128	4.7
1	ham	Nah I don't think he goes to usf, he lives around here though	[nah, dont, think, go, usf, life, around, though]	49	4.1
2	ham	Even my brother is not like to speak with me. They treat me like aids patent.	[even, brother, like, speak, treat, like, aid, patent]	62	3.2
3	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	[date, sunday]	28	7.1
4	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your call...	[per, request, melle, melle, oru, minnaminunginte, nurungu, vettam, set, callertune, caller, pre...	135	4.4

In [4]:
```
## from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline

colors = ['#4889BF', '#ECAE3F', '#008489', '#66BFA1', '#E65656']
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13,5))

bins1 = np.linspace(0, 200, 40)

ax1.hist(data[data['label']=='spam']['body_len'], bins1, alpha=0.5, density=True, label='spam')
ax1.hist(data[data['label']=='ham']['body_len'], bins1, alpha=0.5, density=True, label='ham')
ax1.legend(loc='upper left')
ax1.set_title('Body Length Distribution', fontsize=14)
ax1.set_xlabel('Count')

bins2 = np.linspace(0, 50, 40)
ax2.hist(data[data['label']=='spam']['punct%'], bins2, alpha=0.5, density=True, label='spam', color='red')
ax2.hist(data[data['label']=='ham']['punct%'], bins2, alpha=0.5, density=True, label='ham', color='blue')
ax2.legend(loc='upper right')
ax2.set_title('Punctuation Percentage Distribution', fontsize=14)
ax2.set_xlabel('Count')

plt.show()
```



Split into train/test

In [5]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data[['body_text', 'cleaned_text', 'body_len', 'punct%']], data['l
abel'], test_size=0.2)
```

Apply TfidfVectorizer

In [6]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(analyzer=clean_text)
tfidf_vect_fit = tfidf_vect.fit(X_train['body_text'])

tfidf_train = tfidf_vect_fit.transform(X_train['body_text'])
tfidf_test = tfidf_vect_fit.transform(X_test['body_text'])

X_train_vect = pd.concat([X_train[['body_len', 'punct%']].reset_index(drop=True),
           pd.DataFrame(tfidf_train.toarray())], axis=1)
X_test_vect = pd.concat([X_test[['body_len', 'punct%']].reset_index(drop=True),
           pd.DataFrame(tfidf_test.toarray())], axis=1)

X_train_vect.iloc[0:5, 0:20]
```

Out[6]:

	body_len	punct%	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	33	3.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	91	7.7	0.125411	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	53	5.7	0.185754	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	145	4.1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	51	3.9	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Random Forest Model

In [8]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.model_selection import GridSearchCV
import time
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [111]:
```
rf = RandomForestClassifier()
param = {'n_estimators': [10, 150, 300],
        'max_depth': [30, 60, 90, None]}

gs = GridSearchCV(rf, param, cv=5, n_jobs=-1)
gs_fit = gs.fit(X_train_vect, y_train)
pd.DataFrame(gs_fit.cv_results_).sort_values('mean_test_score', ascending=False)[0:5]
```

Out[111]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_score	split1_t
7	18.260384	0.646578	0.295610	0.015561	90	150	{'max_depth': 90, 'n_estimators': 150}	0.978676	
8	37.095831	0.889115	0.408907	0.011336	90	300	{'max_depth': 90, 'n_estimators': 300}	0.979798	
11	32.918206	0.967842	0.276660	0.060859	None	300	{'max_depth': None, 'n_estimators': 300}	0.979798	
10	19.884642	1.118057	0.293415	0.016871	None	150	{'max_depth': None, 'n_estimators': 150}	0.978676	
6	1.941210	0.094751	0.186103	0.015790	90	10	{'max_depth': 90, 'n_estimators': 10}	0.978676	

In [9]:
```
rf = RandomForestClassifier(n_estimators=150, max_depth=None, n_jobs=-1)

rf_model = rf.fit(X_train_vect, y_train)
y_pred_rf = rf_model.predict(X_test_vect)

precision, recall, fscore, train_support = score(y_test, y_pred_rf, pos_label='spam', average='binary')
print('Precision: {} / Recall: {} / Accuracy: {}'.format(
    round(precision, 3), round(recall, 3), round((y_pred_rf==y_test).sum()/len(y_pred_rf), 3)))
```

```
Precision: 1.0 / Recall: 0.788 / Accuracy: 0.97
```

Gradient Boosting Model

In [113]:
```
gb = GradientBoostingClassifier()
param = {
    'n_estimators': [50, 100, 150],
    'max_depth': [7, 11, 15],
    'learning_rate': [0.1]
}

clf = GridSearchCV(gb, param, cv=5, n_jobs=-1)
cv_fit = clf.fit(X_train_vect, y_train)
pd.DataFrame(cv_fit.cv_results_).sort_values('mean_test_score', ascending=False)[0:5]
```

In [10]:
```
gb = GradientBoostingClassifier(n_estimators=150, max_depth=11)

gb_model = gb.fit(X_train_vect, y_train)
y_pred_gb = gb_model.predict(X_test_vect)

precision, recall, fscore, train_support = score(y_test, y_pred_gb, pos_label='spam', average='binary')
print('Precision: {} / Recall: {} / Accuracy: {}'.format(
    round(precision, 3), round(recall, 3), round((y_pred_gb==y_test).sum()/len(y_pred_gb), 3)))
```

```
Precision: 0.918 / Recall: 0.788 / Accuracy: 0.961
```

Naive Bayes Model

In [11]:
```
import sklearn
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()
classifier = classifier.fit(X_train_vect, y_train)

y_pred_nb = classifier.predict(X_test_vect)

precision, recall, fscore, train_support = score(y_test, y_pred_nb, pos_label='spam', average='binary')
print('Precision: {} / Recall: {} / Accuracy: {}'.format(
    round(precision, 3), round(recall, 3), round((y_pred_nb==y_test).sum()/len(y_pred_nb), 3)))
```

```
Precision: 0.597 / Recall: 0.865 / Accuracy: 0.899
```

Logistic Regression

In [20]:
```
from sklearn.linear_model import LogisticRegression

y_train_bool = [1 if y.lower() == 'spam' else 0 for y in y_train]
log_reg = LogisticRegression(penalty='none', max_iter=100000, #l1_ratio=0.1,
                             random_state=0, solver='lbfgs').fit(X_train_vect, y_train_bool)
y_pred_lg = ['spam' if y == 1 else 'ham' for y in log_reg.predict(X_test_vect)]

precision, recall, fscore, train_support = score(y_test, y_pred_lg, pos_label='spam', average='binary')
print('Precision: {} / Recall: {} / Accuracy: {}'.format(
    round(precision, 3), round(recall, 3), round((y_pred_lg==y_test).sum()/len(y_pred_lg), 3)))
```

```
Precision: 0.965 / Recall: 0.891 / Accuracy: 0.98
```

Logistic Lasso Regression

In [18]:
```
from sklearn.linear_model import LogisticRegression

y_train_bool = [1 if y.lower() == 'spam' else 0 for y in y_train]
log_reg_l1 = LogisticRegression(penalty='l1', max_iter=100000, #l1_ratio=0.1,
                             random_state=0, solver='liblinear').fit(X_train_vect, y_train_bool)
y_pred_lg_l1 = ['spam' if y == 1 else 'ham' for y in log_reg_l1.predict(X_test_vect)]

precision, recall, fscore, train_support = score(y_test, y_pred_lg_l1, pos_label='spam', average='binary')
print('Precision: {} / Recall: {} / Accuracy: {}'.format(
    round(precision, 3), round(recall, 3), round((y_pred_lg_l1==y_test).sum()/len(y_pred_lg_l1), 3)))
```

```
Precision: 0.953 / Recall: 0.782 / Accuracy: 0.964
```

Logistic Ridge Regression

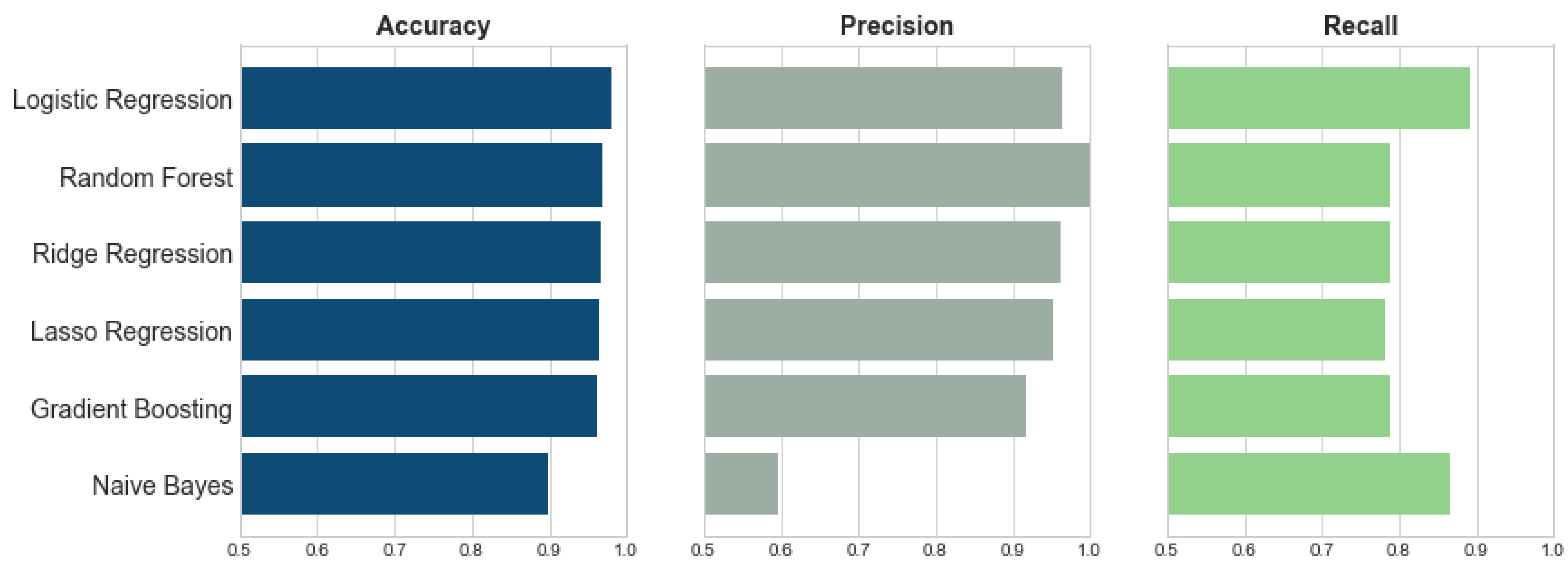
In [17]:
```
from sklearn.linear_model import LogisticRegression

y_train_bool = [1 if y.lower() == 'spam' else 0 for y in y_train]
log_reg_l2 = LogisticRegression(penalty='l2', max_iter=100000, #l1_ratio=0.1,
                             random_state=0, solver='lbfgs').fit(X_train_vect, y_train_bool)
y_pred_lg_l2 = ['spam' if y == 1 else 'ham' for y in log_reg_l2.predict(X_test_vect)]

precision, recall, fscore, train_support = score(y_test, y_pred_lg_l2, pos_label='spam', average='binary')
print('Precision: {} / Recall: {} / Accuracy: {}'.format(
    round(precision, 3), round(recall, 3), round((y_pred_lg_l2==y_test).sum()/len(y_pred_lg_l2), 3)))
```

```
Precision: 0.961 / Recall: 0.788 / Accuracy: 0.966
```

Summary and Conclusion

In [120]:
```
models = ['Random Forest', 'Gradient Boosting', 'Naive Bayes',
          'Logistic Regression', 'Lasso Regression', 'Ridge Regression']
precisions = [1, 0.918, 0.597, 0.965, 0.953, 0.961]
recalls = [0.788, 0.788, 0.865, 0.891, 0.782, 0.788]
accuracys = [0.97, 0.961, 0.899, 0.98, 0.964, 0.966]
summary = pd.DataFrame({'Model': models, 'Accuracy': accuracys, 'Precision':precisions, 'Recall':recalls})
summary = summary.sort_values('Accuracy', ascending=False).reset_index(drop=True)
```

In [119]:
```
plt.style.use('seaborn-whitegrid')
fig, axes = plt.subplots(1, 3, figsize=(13,5))
axes = axes.ravel()

columns = summary.columns
y_pos = np.arange(len(summary['Model']))
colors = ['#0F4C75', '#9CADA4', '#91D18B']

for idx, ax in enumerate(axes):
    h_value = summary[columns[idx + 1]]
    barlist = ax.barh(y_pos, h_value, color=colors[idx])
    if idx == 0:
        ax.set_yticks(y_pos)
        ax.set_yticklabels(summary['Model'], fontsize=14)
    else:
        ax.set_yticklabels('')
    ax.invert_yaxis()
    ax.set_title(columns[idx + 1], fontsize=14, fontweight='bold')
    ax.set_xlim((0.5,1))
    ax.grid(b=None, axis='y')

plt.show()
```



The **best performing model** is the **Logistic Regression**. It achieved the highest accuracy and Recall, and relative high precision that is just second to Random Forest. Here, it was demonstrated that the simplest model has the best result.