Email Text Classifier

This notebook creates a machine learning model that classifies emails into three categories: HR, Marketing, IT.

We have a .csv dataset that contains examples of emails for each category. The machine learning model does the prediction of the category of a given email (assume that the input will always be a text).

After that, the ML model is tested out with some examples (1 example per category) that are not in the dataset.

In [1]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfTransformer, CountVec
from sklearn.model_selection import train_test_split
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
```

Read and prepare the input data

In [2]:
```
df = pd.read_csv("coding_challenge_dataset.csv", names=["email", "cate
print(len(df))
df.head(10)
```

```
160
```

Out[2]:

	email	category
0	this is an it email and we are struggling with...	IT
1	can you guys check the network settings	IT
2	A computer virus is a software program that h...	IT
3	All versions of Microsoft Windows do not come...	IT
4	As with most computer errors, your first step...	IT
5	what is a computer virus? a computer virus is...	IT
6	Does Windows come with a virus protection prog...	IT
7	what do I do if my hard disk fails to work as ...	IT
8	My Computer does not power up?- Check that all...	IT
9	I can't delete a file because it is being used...	IT

Here I use the English stopwords, put them in a set.

In [3]:
```
def create_stopwords():
    with open('./stopwords.txt') as f:
        lines = f.read().splitlines()
    return lines

STOPWORDS = set(create_stopwords())
```

In this text cleaning step, I first convert the text to lower case, then I remove numbers. I use a tokenizer to remove whitespaces and punctuations by matching word characters.

Then I remove all the words in the stop word set.

For the TF-IDF in a later step, I convert the tokens to their stems to facilitate counting.

In [4]:
```
tokenizer = RegexpTokenizer(r'\w+')
p_stemmer = PorterStemmer()

def clean_text(raw):
    """ clean and tokenize document string """
    # Lowercase
    text = raw.lower()
    # Remove numbers
    text = ''.join([i for i in text if not i.isdigit()])
    # Remove whitespaces and punctuations
    tokens = tokenizer.tokenize(text)
    # Remove stop words in English
    stopped_tokens = [i for i in tokens if i not in STOPWORDS]
    # stem tokens
    stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]
    text = " ".join(stemmed_tokens)
    return text

df['email_cleaned'] = df['email'].apply(clean_text)
df['email_cleaned'].head(10)
```

Out[4]:
```
0                              it email struggl network
1                                 guy check network set
2    comput viru softwar program intent creat caus ...
3    version microsoft window come pre instal viru ...
4    comput error first step shut comput restart it...
5    comput viru comput viru softwar program intent...
6    window come viru protect program version micro...
7    hard disk fail work comput error first step sh...
8    comput power check cabl secur plug back machin...
9    delet file it use window close program run com...
Name: email_cleaned, dtype: object
```

Split the dataset into a training set and a test set with a 70/30 split.

In [5]:
```
X = df['email_cleaned']
y = df['category']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state = 0)
print(len(X_train), len(X_test))
X_train.head(), y_train.head()
```

```
112 48
```

Out[5]:
```
(118    promo code valid new one free ride begin end z...
 95     protect servic eeo categori encompass occup wo...
 55           staff trainer week question help first day
 109    get instant discount everyday item need save t...
 18     display monitor make sure monitor power light ...
 Name: email_cleaned, dtype: object,
 118    Marketing
 95            HR
 55            HR
 109    Marketing
 18            IT
 Name: category, dtype: object)
```

Vectorization

Here I use the TF-IDF transformer to convert texts to numerical feature vectors.

In [6]:
```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)

tfidf_vectorizer = TfidfTransformer(use_idf=True)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train_counts)
```

In [7]:
```
X_test_counts = count_vect.transform(X_test)
X_test_tfidf = tfidf_vectorizer.transform(X_test_counts)
```

Machine Learning Models

In this part I tried 3 most common ML models for text classification tasks: Naive Bayes, Stochastic Gradiant Descent, and Random Forest.

Naive Bayes

In [8]:
```
from sklearn.naive_bayes import MultinomialNB
clf_nb = MultinomialNB().fit(X_train_tfidf, y_train)
```

In [9]:
```
predicted_nb = clf_nb.predict(X_test_tfidf)
predicted_nb
```

Out[9]:
```
array(['Marketing', 'Marketing', 'IT', 'IT', 'IT', 'HR', 'IT', 'IT',
'HR',
       'HR', 'IT', 'HR', 'IT', 'Marketing', 'IT', 'HR', 'IT', 'HR',
       'Marketing', 'HR', 'IT', 'Marketing', 'HR', 'IT', 'IT', 'IT',
       'Marketing', 'HR', 'HR', 'Marketing', 'Marketing', 'IT', 'HR',
       'HR', 'IT', 'HR', 'Marketing', 'IT', 'IT', 'Marketing', 'IT',
'IT',
       'HR', 'Marketing', 'Marketing', 'Marketing', 'Marketing', 'IT'
],
      dtype='<U9')
```

In [10]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, predicted_nb))
```

```
              precision    recall  f1-score   support

          HR       1.00      0.82      0.90        17
          IT       0.75      1.00      0.86        15
   Marketing       0.93      0.81      0.87        16

    accuracy                           0.88        48
   macro avg       0.89      0.88      0.88        48
weighted avg       0.90      0.88      0.88        48
```

In [11]:
```
metrics.confusion_matrix(y_test, predicted_nb)
```

Out[11]:
```
array([[14,  2,  1],
       [ 0, 15,  0],
       [ 0,  3, 13]])
```

Stochastic Gradient Descent

In [12]:
```
from sklearn.linear_model import SGDClassifier
clf_sgd = SGDClassifier().fit(X_train_tfidf, y_train)
```

In [13]:
```
predicted_sgd = clf_sgd.predict(X_test_tfidf)
predicted_sgd
```

Out[13]:
```
array(['Marketing', 'Marketing', 'IT', 'IT', 'IT', 'HR', 'HR', 'HR',
'HR',
       'Marketing', 'IT', 'HR', 'IT', 'Marketing', 'Marketing', 'HR',
       'IT', 'HR', 'Marketing', 'HR', 'IT', 'Marketing', 'HR', 'IT',
'IT',
       'IT', 'Marketing', 'HR', 'HR', 'Marketing', 'Marketing', 'IT',
       'HR', 'HR', 'IT', 'HR', 'Marketing', 'IT', 'IT', 'Marketing',
'IT',
       'IT', 'HR', 'Marketing', 'Marketing', 'Marketing', 'Marketing'
,
       'IT'], dtype='<U9')
```

In [14]:
```
print(metrics.classification_report(y_test, predicted_sgd))
```

```
              precision    recall  f1-score   support

          HR       0.93      0.82      0.87        17
          IT       0.88      1.00      0.94        15
   Marketing       0.88      0.88      0.88        16

    accuracy                           0.90        48
   macro avg       0.90      0.90      0.90        48
weighted avg       0.90      0.90      0.89        48
```

In [15]:
```
metrics.confusion_matrix(y_test, predicted_sgd)
```

Out[15]:
```
array([[14,  1,  2],
       [ 0, 15,  0],
       [ 1,  1, 14]])
```

Random Forest

In [16]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [17]:
```
clf_rf = RandomForestClassifier(n_estimators=100, random_state=0)
clf_rf.fit(X_train_tfidf, y_train)
```

Out[17]:
```
RandomForestClassifier(random_state=0)
```

In [18]:
```
predicted_rf = clf_rf.predict(X_test_tfidf)
print(metrics.classification_report(y_test, predicted_rf))
```

```
              precision    recall  f1-score   support

          HR       0.68      1.00      0.81        17
          IT       1.00      0.73      0.85        15
   Marketing       1.00      0.75      0.86        16

    accuracy                           0.83        48
   macro avg       0.89      0.83      0.84        48
weighted avg       0.89      0.83      0.84        48
```

In [19]:
```
metrics.confusion_matrix(y_test, predicted_rf)
```

Out[19]:
```
array([[17,  0,  0],
       [ 4, 11,  0],
       [ 4,  0, 12]])
```

Grid Search

By comparing the results of the 3 classifiers, we can see that the SGD model has the best performance (90% accuracy). So I do a grid search to look for the best parameters for the SGD model.

In [20]:
```
from sklearn.model_selection import GridSearchCV
parameters = {'loss': ('hinge',),
              'penalty': ('l1', 'l2'),
              'alpha': (1e-3, 1e-4, 1e-5),
              'random_state': (0,)}
```

In [21]:
```
gs_clf = GridSearchCV(clf_sgd, parameters, n_jobs=-1)
gs_clf = gs_clf.fit(X_train_tfidf, y_train)
```

In [22]:
```
gs_clf.best_score_, gs_clf.best_params_
```

Out[22]:
```
(0.9648221343873518,
 {'alpha': 0.001, 'loss': 'hinge', 'penalty': 'l2', 'random_state': 0
})
```

With the best parameters, we can achieve around 96% accuracy.

Build the final model

For the best SGD model we found, fit the entire dataset. Then manually test out the model with 3 new emails from 3 categories.

In [23]:
```
X_counts = count_vect.transform(X)
X_tfidf = tfidf_vectorizer.transform(X_counts)
```

In [24]:
```
clf_sgd = SGDClassifier(alpha=0.001,
                        loss='hinge',
                        penalty='l2',
                        random_state=0).fit(X_tfidf, y)
```

In [25]:
```
# Three examples of IT, HR, and marketing
new_entry = pd.DataFrame(["Do you know how to open the attachment of a
                          "Dear Recruiter, I am applying my job applic
                          "Our special offer of 30 percent discount st
                          )
new_entry.head()

new_entry = new_entry[0].apply(clean_text)
```

In [26]:
```
new_entry_counts = count_vect.transform(new_entry)
new_entry_tfidf = tfidf_vectorizer.transform(new_entry_counts)
```

In [27]:
```
predicted_new_entry = clf_sgd.predict(new_entry_tfidf)
predicted_new_entry
```

Out[27]:
```
array(['IT', 'HR', 'Marketing'], dtype='<U9')
```

We can see that the classifier correctly classifies all 3 new emails: IT, HR, and marketing.