SVM_Nov_Adv_Usecase

February 28, 2022

[1]:
```
# Importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
from sklearn import feature_extraction, model_selection, naive_bayes, metrics,␣
↪svm
from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Libraries for text processing
import re, nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sci\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\sci\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

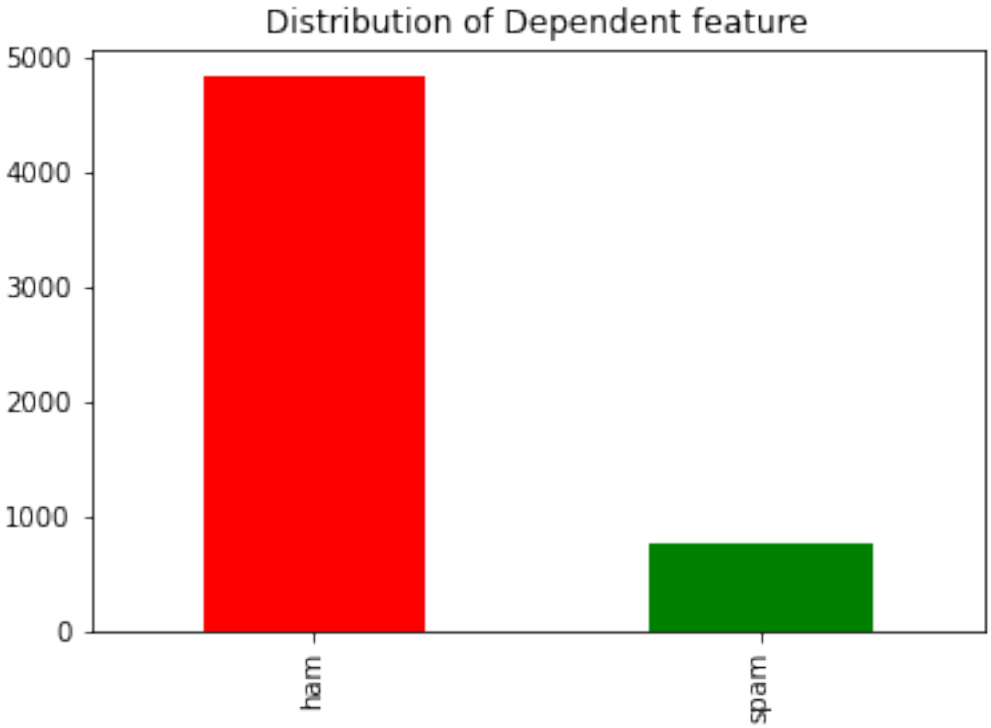
dataset link https://drive.google.com/file/d/1UEc9IY2HgIAYOsm4qpXdqUO5a1ZyeyTV/view?usp=sharing

[2]:
```
df = pd.read_csv("spam_clean.csv", encoding='latin-1')
df.head()
```

[2]:

	type	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…

[3]:
```
freq = pd.value_counts(df["type"], sort= True)
freq.plot(kind= 'bar', color= ["red", "green"])
plt.title('Distribution of Dependent feature')
plt.show()
```



[4]:
```
def clean_tokenized_sentence(s):
    """Performs basic cleaning of a tokenized sentence"""
    cleaned_s = ""  # Create empty string to store processed sentence.
    words = nltk.word_tokenize(s)
    for word in words:
        # Convert to lowercase #
        c_word = word.lower()
        # Remove punctuations #
        c_word = re.sub(r'[^\w\s]', '', c_word)
        # Remove stopwords #
        if c_word != '' and c_word not in stopwords.words('english'):
```

```
            cleaned_s = cleaned_s + " " + c_word    # Append processed words to␣
↪new list.
    return(cleaned_s.strip())
```

[5]:
```
df["cleaned_message"] = df["message"].apply(clean_tokenized_sentence)
```

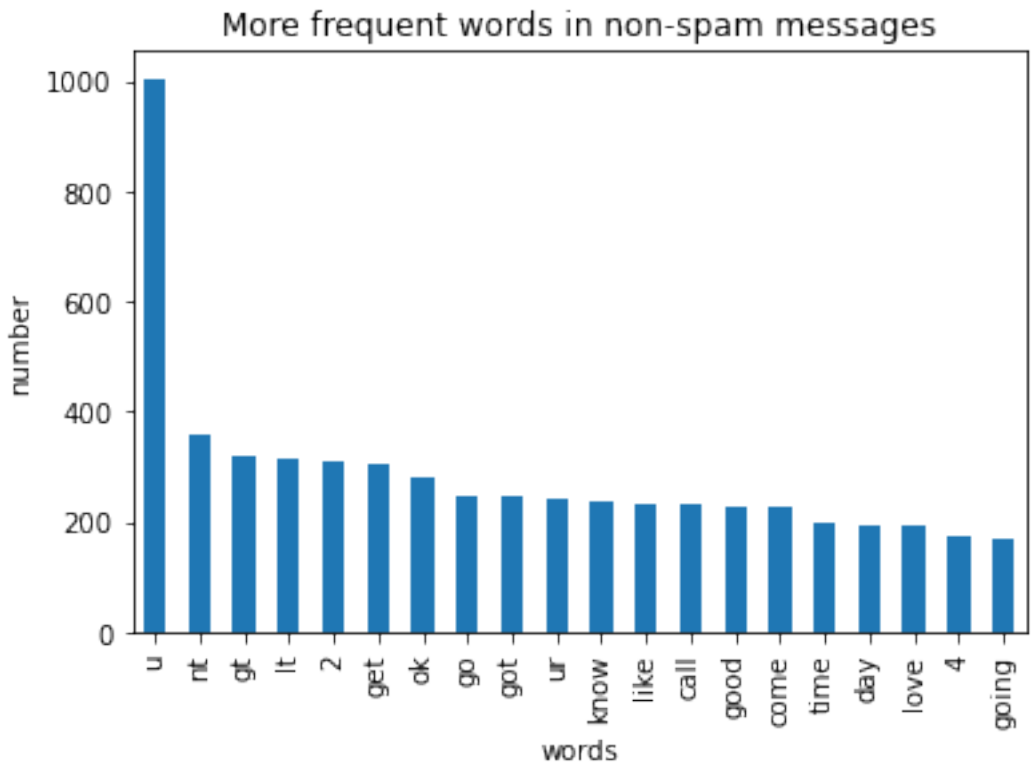
[6]:
```
df.head(10)
```

[6]:
```
   type                                                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…
5  spam  FreeMsg Hey there darling it's been 3 week's n…
6   ham  Even my brother is not like to speak with me. …
7   ham  As per your request 'Melle Melle (Oru Minnamin…
8  spam  WINNER!! As a valued network customer you have…
9  spam  Had your mobile 11 months or more? U R entitle…

                                  cleaned_message
0  go jurong point crazy available bugis n great …
1                            ok lar joking wif u oni
2  free entry 2 wkly comp win fa cup final tkts 2…
3                u dun say early hor u c already say
4          nah nt think goes usf lives around though
5  freemsg hey darling 3 week word back like fun …
6     even brother like speak treat like aids patent
7  per request melle melle oru minnaminunginte nu…
8  winner valued network customer selected receiv…
9  mobile 11 months u r entitled update latest co…
```

[7]:
```
counter_ham = Counter(" ".join(df[df['type']=='ham']["cleaned_message"]).
↪split()).most_common(20)
df_ham = pd.DataFrame.from_dict(counter_ham)
df_ham = df_ham.rename(columns={0:"words in non-spam", 1:"count"})

df_ham.plot.bar(legend = False)
y_pos = np.arange(len(df_ham["words in non-spam"]))
plt.xticks(y_pos, df_ham["words in non-spam"])
plt.title('More frequent words in non-spam messages')
plt.xlabel('words')
plt.ylabel('number')
plt.show()
```



[8]:
```
counter_spam = Counter(" ".join(df[df['type']=='spam']["cleaned_message"]).
 ↪split()).most_common(20)
df_spam = pd.DataFrame.from_dict(counter_spam)
df_spam = df_spam.rename(columns={0:"words in spam", 1:"count_"})

df_spam.plot.bar(legend = False, color = 'orange')
y_pos = np.arange(len(df_spam["words in spam"]))
plt.xticks(y_pos, df_spam["words in spam"])
plt.title('More frequent words in spam messages')
plt.xlabel('words')
plt.ylabel('number')
plt.show()
```

[9]:
```
f = feature_extraction.text.CountVectorizer()
X = f.fit_transform(df["cleaned_message"])
```

[10]:
```
df["type"] = df["type"].map({'spam':1,'ham':0})

X_train, X_test, y_train, y_test = train_test_split(X, df['type'], test_size=0.
 ↪25, random_state=42)
print([np.shape(X_train), np.shape(X_test)])
```

```
[(4179, 8951), (1393, 8951)]
```

[11]:
```
perf = {}

perf['algorithm_name'] = []
perf['cv_score_f1'] = []
perf['cv_std_f1'] = []
```

[12]:
```
clf = DecisionTreeClassifier(max_depth=5)
cv_scores = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=5,␣
 ↪scoring='f1')
```

[13]:
```
perf['algorithm_name'].append('Decision Trees')
perf['cv_score_f1'].append(round(cv_scores.mean()*100, 1))
perf['cv_std_f1'].append(round(cv_scores.std(), 2))
```

[14]:
```
clf = RandomForestClassifier(n_estimators=50, max_depth=20)
cv_scores = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=5,␣
↪scoring='f1')
```

[15]:
```
perf['algorithm_name'].append('Random Forest')
perf['cv_score_f1'].append(round(cv_scores.mean()*100, 1))
perf['cv_std_f1'].append(round(cv_scores.std(), 2))
```

[16]:
```
clf = SVC(kernel='linear')
cv_scores = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=5,␣
↪scoring='f1')
```

[17]:
```
perf['algorithm_name'].append('SVM Linear')
perf['cv_score_f1'].append(round(cv_scores.mean()*100, 1))
perf['cv_std_f1'].append(round(cv_scores.std(), 2))
```

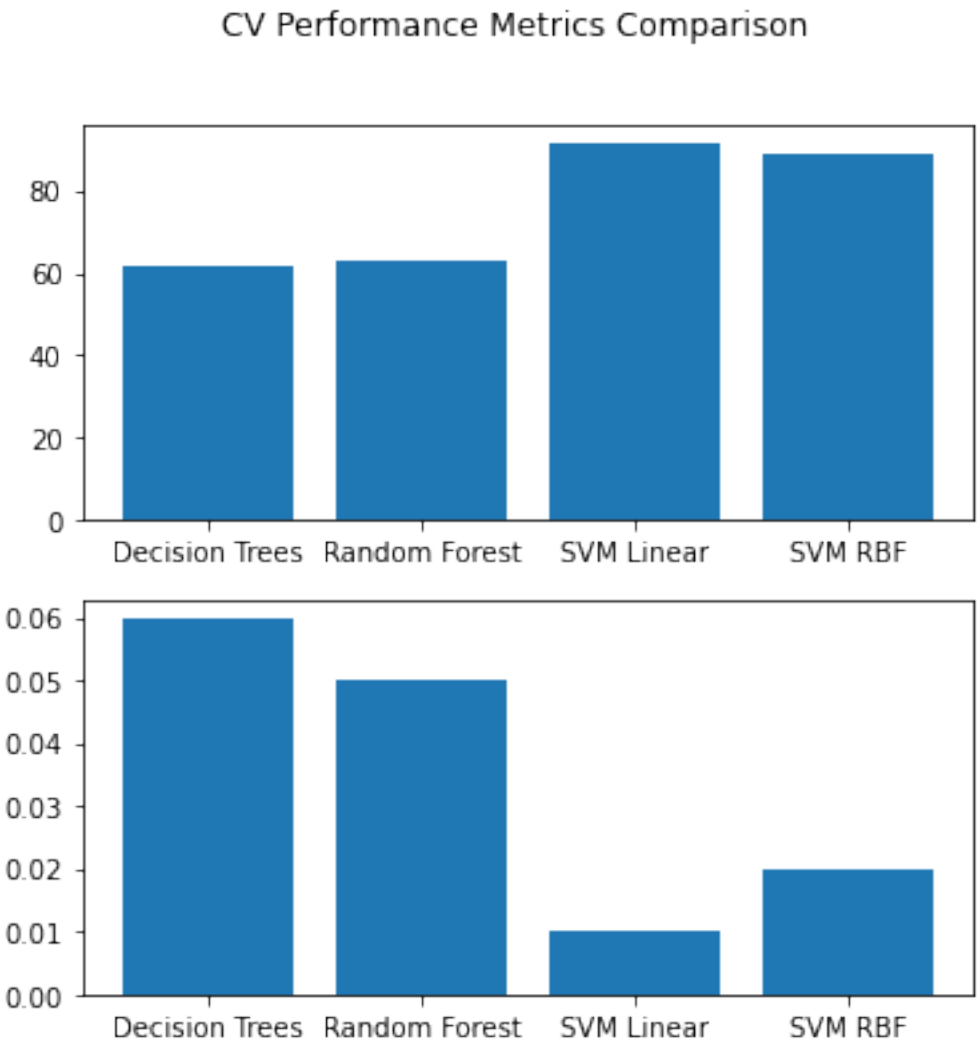
[18]:
```
clf = SVC(kernel='rbf')
cv_scores = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=5,␣
↪scoring='f1')
```

[19]:
```
perf['algorithm_name'].append('SVM RBF')
perf['cv_score_f1'].append(round(cv_scores.mean()*100, 1))
perf['cv_std_f1'].append(round(cv_scores.std(), 2))
```

[20]:
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(6,6))
fig.suptitle('CV Performance Metrics Comparison')

ax1.bar(perf['algorithm_name'], perf['cv_score_f1'])
ax2.bar(perf['algorithm_name'], perf['cv_std_f1'])
```

[20]:
```
<BarContainer object of 4 artists>
```



[21]:
```
svc = svm.SVC(kernel='linear')
svc.fit(X_train, y_train)
y_predict = svc.predict(X_test)
```

[22]:
```
metrics.f1_score(y_test, y_predict)
```

[22]: 0.8997134670487107

[25]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

[24]:
```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      1202
```

```
           1       0.99      0.82      0.90       191

    accuracy                           0.97      1393
   macro avg       0.98      0.91      0.94      1393
weighted avg       0.98      0.97      0.97      1393
```

[26]:
```
confusion_matrix(y_test, y_predict)
```

[26]:
```
array([[1201,    1],
       [  34,  157]], dtype=int64)
```

[ ]:
```

```