SMS Spam Classification

Basavaraj A arabhanvi (Email:arabhanviba@gmail.com)

> Data Preparation: Obtain a dataset containing SMS messages labeled as spam or ham (non-spam).
>
> Text Preprocessing: Clean and preprocess the text data by removing punctuation, stopwords, and converting text to lowercase.
>
> Feature Extraction: Convert text data into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency).
>
> Train-Test Split: Split the dataset into training and testing sets.
>
> KNN Model Training: Train a KNN classifier on the training data.
>
> Model Evaluation: Evaluate the trained model's performance using metrics like accuracy, precision, recall, and F1-score on the testing data.
>
> Hyperparameter Tuning: Experiment with different values of k (number of neighbors) to optimize the model's performance.
>
> Deployment: Deploy the trained model to classify new SMS messages as spam or ham.

In [1]:

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [2]:

```
df = pd.read_csv("E:\Pg Studies\Intership\Bharath intren\spam.csv")
df.head()
```

Out[2]:

	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...

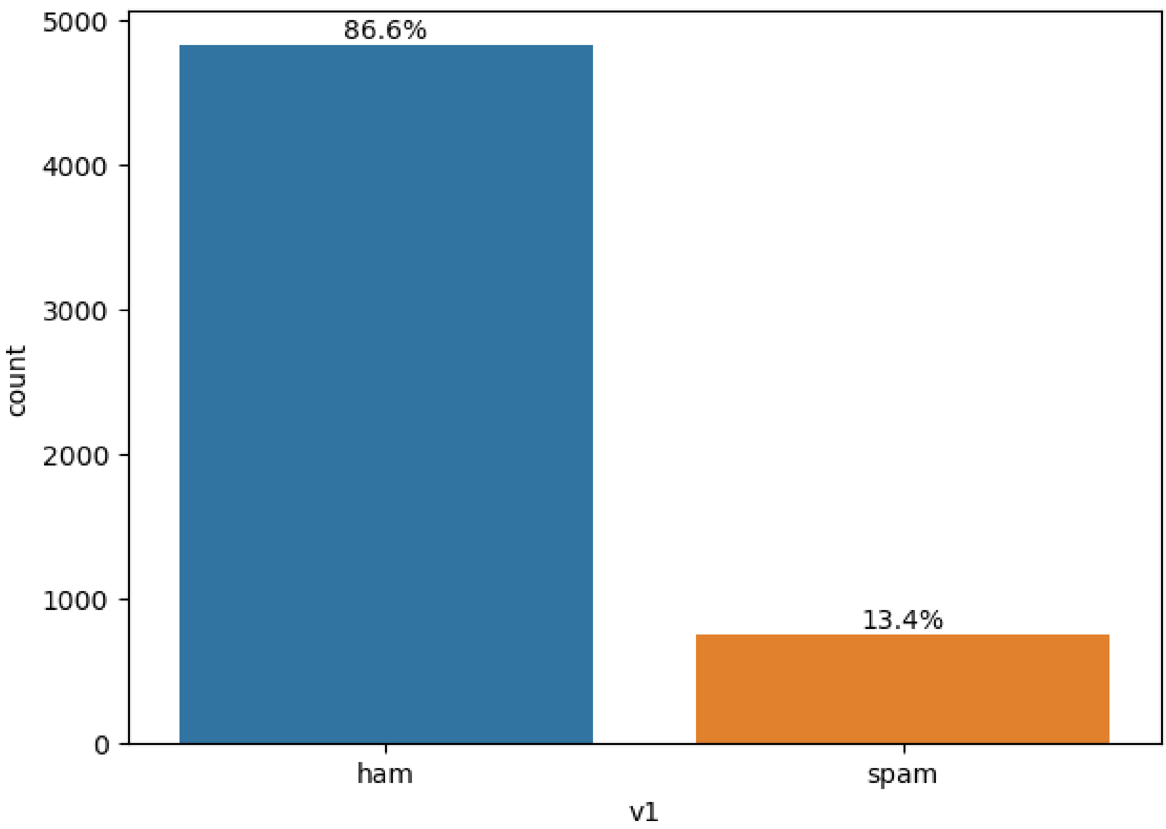
Split the data into train and test set

In [3]:
```
X = df.iloc[:, 1:2]
y = df.iloc[:, 0:1]
```

Visualize the class distribution

In [4]:
```
fig, ax = plt.subplots(figsize=(7, 5))
sns.countplot(x="v1", data=df)

for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/len(X))
    x_countplot = p.get_x() + p.get_width()/2
    y_countplot = p.get_height()+ 50
    ax.annotate(percentage, (x_countplot, y_countplot), ha='center')
plt.show()
```



We can see that there's huge differences in class distribution, where the majority of data is ham (86.6%) and only 13.4% are spam

Split the training and testing set

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X.values.ravel(),
    y.values.ravel(),
    test_size=0.20,
    random_state=42)
```

Feature Extraction

In [6]:
```
count_vectorizer = CountVectorizer()
tfidf_vectorizer = TfidfVectorizer()
```

Model Training

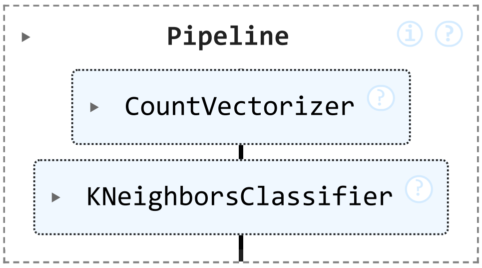
Create the pipeline using Count Vectorizer

In [7]:
```
clf_method = KNeighborsClassifier()

clf_count_vectorizer = Pipeline([
     ('vectorizer', count_vectorizer),
     ('classifier', clf_method)
 ])

clf_count_vectorizer.fit(X_train, y_train)
```

Out[7]:



In [20]:
```
y_train_pred_cvect = clf_count_vectorizer.predict(X_train)
print(f"Train Accuracy using Count Vectorizer: {accuracy_score(y_train, y_train_pre
```

```
Train Accuracy using Count Vectorizer: 0.973
```

In [21]:
```
print(classification_report(y_train, y_train_pred_cvect))
```

```
              precision    recall  f1-score   support

         ham       0.97      1.00      0.98      3860
        spam       0.99      0.81      0.89       597

    accuracy                           0.97      4457
   macro avg       0.98      0.90      0.94      4457
weighted avg       0.97      0.97      0.97      4457
```

Model Evaluation

In [25]:
```
y_test_pred_cvect = clf_count_vectorizer.predict(X_test)

print(f"Test Accuracy using Count Vectorizer: {accuracy_score(y_test, y_test_pred_c
```

```
Test Accuracy using Count Vectorizer: 0.968
```

In [26]:
```
print(classification_report(y_test, y_test_pred_cvect))
```

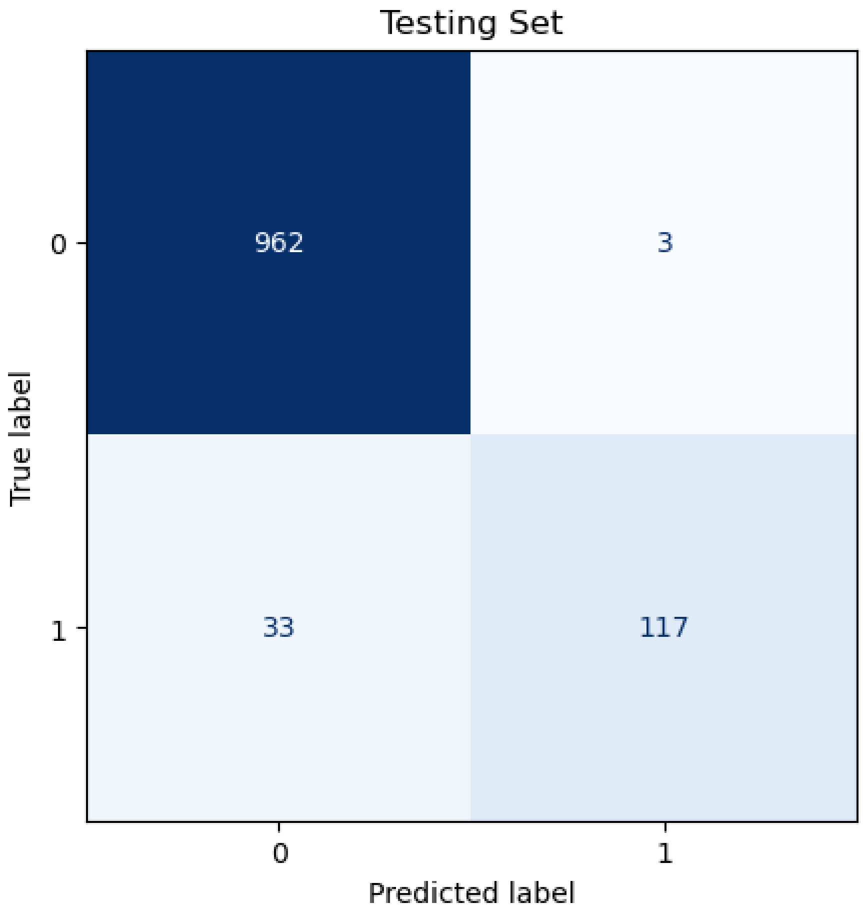
```
              precision    recall  f1-score   support

         ham       0.97      1.00      0.98       965
        spam       0.97      0.78      0.87       150

    accuracy                           0.97      1115
   macro avg       0.97      0.89      0.92      1115
weighted avg       0.97      0.97      0.97      1115
```

In [27]:
```
conf_mat_train = ConfusionMatrixDisplay(confusion_matrix(y_test, y_test_pred_cvect)

fig, ax = plt.subplots(figsize=(5, 5))
ax.set_title('Testing Set')
conf_mat_train.plot(cmap=plt.cm.Blues, ax=ax, colorbar=False);
```



Conclusion

The model achieved an accuracy of 0.973 on the training set and 0.968 on the test set, compared to an accuracy of 0.920 on the training set and 0.916 on the test set achieved by the TF-IDF Vectorizer with KNN. Therefore, it can be concluded that the CountVectorizer with KNN model is more reliable and accurate in predicting the outcome of the given dataset.