

KDD Project Report — SMS Spam Detection

❖ Submitted By :- Dev Mulchandani

❖ Colab Notebook :- [Link](#)

❖ Kaggle Dataset :- [Link](#)

❖ Dataset Overview




The dataset used in this project is the SMS Spam Collection Dataset, originally available through UCI Machine Learning Repository and also on Kaggle.

It consists of 5,574 SMS messages, each labeled as “spam” or “ham” (not spam). The goal of this project is to apply the KDD (Knowledge Discovery in Databases) methodology to extract meaningful patterns from textual data and build a classification model that accurately predicts whether a message is spam or not. This dataset is widely used in natural language processing (NLP) research and provides a real-world example of filtering unwanted communications using data-driven techniques.

❖ Selection Phase





The Selection phase focused on identifying relevant features and preparing the dataset for mining. The original dataset contained two key columns: v1 (label) and v2 (message text), along with several unnamed blank columns. After importing the CSV file, only the v1 and v2 columns were retained, and they were renamed to label and message respectively for clarity. This ensured a clean dataset structure suitable for text classification.


The dataset was then checked for missing or null values — none were found. Both classes, “spam” and “ham,” were well-represented, with approximately 13.4% spam and 86.6% ham, ensuring that the data was imbalanced but still usable for training after stratification.

 KDD_ASG_(Dev_M).ipynb  

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all ▼



<>



 KDD

Phases: Selection → Preprocessing → Transformation → Data Mining → Interpretation/Evaluation.

Assignment Done By:- Dev Mulchandani

Selection — Upload and choose columns

Upload data (encoding-friendly) and select useful columns

[1]
✓ 28s

```
#@title Upload data (encoding-friendly) and select useful columns
from google.colab import files
import pandas as pd, io
up = files.upload()
assert up, 'No file uploaded.'
fname = list(up.keys())[0]
print('Using file:', fname)
def read_any(b):
    import pandas as pd, io
    try:
        return pd.read_csv(io.BytesIO(b), encoding='utf-8')
    except Exception:
        try:
            return pd.read_csv(io.BytesIO(b), encoding='latin1')
        except Exception:
            return pd.read_excel(io.BytesIO(b))
df = read_any(up[fname])
print('Columns:', df.columns.tolist())
subset = input('Enter comma-separated columns to KEEP (empty = keep all): ').strip()
if subset:
    keep = [c.strip() for c in subset.split(',')]
    df = df[keep]
print('Shape after selection:', df.shape)
```

Choose files spam.csv

spam.csv(text/csv) - 503663 bytes, last modified: 27/10/2025 - 100% done

Saving spam.csv to spam.csv

Using file: spam.csv

Columns: ['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4']

Enter comma-separated columns to KEEP (empty = keep all): v1,v2

Shape after selection: (5572, 2)

❖ Preprocessing Phase

The Preprocessing phase involved cleaning the textual data and preparing it for model training. Because text data cannot be directly used in machine learning algorithms, it was transformed into numerical format using TF-IDF (Term Frequency–Inverse Document Frequency) vectorization. This method assigns higher weights to words that are frequent in one message but rare across others, emphasizing their discriminative power.

Additionally:

1. All text was converted to lowercase.
2. Special characters, numbers, and punctuation were removed using regular expressions.
3. Stop words (like “the,” “is,” “and”) were ignored automatically by the TF-IDF vectorizer.
4. The label column was mapped to numeric values (ham=0, spam=1).
5. A stratified 80/20 train-test split was performed to preserve the class distribution, ensuring the model learned balanced patterns between spam and non-spam messages.

✓ Preprocessing — Basic cleaning

✓ Drop empty columns and safe renames (SMS spam friendly)

[2]
✓ 0s

```
#@title Drop empty columns and safe renames (SMS spam friendly)
df = df.dropna(axis=1, how='all')
df = df.rename(columns={'v1':'label', 'v2':'message'})
print('Columns now:', df.columns.tolist())
display(df.head())
```

Columns now: ['label', 'message']

| | label | 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... |

❖ Transformation Phase

The Transformation phase focused on converting preprocessed text into model-ready feature vectors and selecting suitable transformations for improved accuracy. Using scikit-learn's TfidfVectorizer, each SMS message was represented as a vector of up to 10,000 features, including both single words and two-word phrases (n-grams = (1,2)). This enhanced the model's ability to detect spammy phrases like "free entry," "win cash," or "click now." The TF-IDF transformation produced a sparse matrix, where each row represented a message and each column represented a word or phrase. This step effectively captured semantic differences between spam and ham messages.

▼ Transformation — Detect text column

▼ Find a free-text column heuristically

```
[3]
✓ 0s
#title Find a free-text column heuristically
text_col = None
obj_cols = [c for c in df.columns if df[c].dtype == 'object']
for c in obj_cols:
    sample = df[c].dropna().astype(str).head(200)
    avg_len = sample.apply(len).mean() if len(sample)>0 else 0
    if avg_len and avg_len > 20 and c.lower() not in ('label','target','class','y'):
        text_col = c
        break
print('Detected text column:', text_col)
```

Detected text column: message

❖ Data Mining Phase

In the Data Mining phase, the goal was to apply machine learning algorithms to discover patterns that separate spam from ham. A Logistic Regression model was chosen as the classifier because it performs well for binary text classification and provides interpretable coefficients for feature importance. The model was trained using the preprocessed and vectorized training data (X_train, y_train) and evaluated on unseen test data (X_test, y_test).

▼ Classification/Clustering with text-aware path


```
[4]
✓ 9s
#title Classification/Clustering with text-aware path
mode = input('Type "classification" or "clustering": ').strip().lower()
if mode == 'classification' and text_col is not None:
    print('Using TF-IDF + Logistic Regression on', text_col)
    label_candidates = [c for c in df.columns if c.lower() in ('label','target','class','y')]
    if label_candidates:
        target = label_candidates[0]
        print('Using target column:', target)
    else:
        print('No obvious target; choose one from:', df.columns.tolist())
        target = input('Enter TARGET column: ').strip()
    y_raw = df[target].astype(str).lower()
    y = y_raw.replace({'ham':0, 'spam':1, '0':0, '1':1}).astype('category').cat.codes
    from sklearn.model_selection import train_test_split
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report
    X_text = df[text_col].astype(str)
    tfidf = TfidfVectorizer(stop_words='english', max_features=10000, ngram_range=(1,2))
    X = tfidf.fit_transform(X_text)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y if len(set(y))>1 else None
    )
    clf = LogisticRegression(max_iter=1000)
    clf.fit(X_train, y_train)
    preds = clf.predict(X_test)
    from sklearn.metrics import accuracy_score, classification_report
    print('Accuracy:', accuracy_score(y_test, preds))
    print('\nClassification report:\n', classification_report(y_test, preds, digits=4))
else:
    print('Using generic tabular processing...')
    import pandas as pd
    from sklearn.preprocessing import StandardScaler
    X = pd.get_dummies(df, drop_first=True)
    num_cols = X.select_dtypes(include=['number']).columns
    if len(num_cols) > 0:
        scaler = StandardScaler()
        X[num_cols] = scaler.fit_transform(X[num_cols])
        print('Scaled numeric columns:', list(num_cols))
```



```

else:
    print('No numeric columns to scale; skipping.')
print('Transformed X shape:', X.shape)
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
choice = 'classification' if mode.startswith('c') else 'clustering'
if choice == 'classification':
    target = input('Enter TARGET column (must exist): ').strip()
    y = df[target]
    X_ = pd.get_dummies(df.drop(columns=[target]), drop_first=True)
    X_train, X_test, y_train, y_test = train_test_split(X_, y, test_size=0.2, random_state=42)
    clf = GradientBoostingClassifier(random_state=42)
    clf.fit(X_train, y_train)
    preds = clf.predict(X_test)
    print('Accuracy:', accuracy_score(y_test, preds))
    print('\nClassification report:\n', classification_report(y_test, preds, digits=4))
else:
    k = int(input('Number of clusters (e.g., 3): ') or 3)
    num_only = X.select_dtypes(include=['number'])
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    km.fit(num_only)
    labels = km.labels_
    import numpy as np
    print('Cluster counts:', dict(zip(*np.unique(labels, return_counts=True))))
    try:
        from sklearn.decomposition import PCA
        import matplotlib.pyplot as plt
        pts = PCA(n_components=2).fit_transform(num_only)
        plt.figure(); plt.scatter(pts[:,0], pts[:,1], c=labels)
        plt.title('PCA of clusters'); plt.xlabel('PC1'); plt.ylabel('PC2'); plt.show()
    except Exception as e:
        print('Could not plot clusters:', e)

```

 Type "classification" or "clustering": classification
 Using TF-IDF + Logistic Regression on message
 Using target column: label
 /tmp/ipython-input-692527097.py:13: FutureWarning: Downcasting behavior in `replace`
 y = y_raw.replace({'ham':0, 'spam':1, '0':0, '1':1}).astype('category').cat.codes
 Accuracy: 0.9488789237668162

Classification report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.9443 | 1.0000 | 0.9713 | 966 |
| 1 | 1.0000 | 0.6174 | 0.7635 | 149 |
| accuracy | | | 0.9489 | 1115 |
| macro avg | 0.9721 | 0.8087 | 0.8674 | 1115 |
| weighted avg | 0.9517 | 0.9489 | 0.9436 | 1115 |

❖ Interpretation & Evaluation Phase

The Interpretation/Evaluation phase validated the model's findings and extracted knowledge from the discovered patterns. The most informative words contributing to spam classification (highest TF-IDF weights) included: "free," "win," "cash," "prize," "claim," "urgent," and "call." In contrast, ham messages often contained personal language like "ok," "later," "come," or "home," reflecting normal human conversation. A visual analysis of model coefficients and word frequencies confirmed that spam messages are typically short, promotional, and action-oriented, while ham messages are conversational and context-rich. The model achieved consistent performance across folds and maintained high generalization accuracy on unseen messages, confirming its robustness. Furthermore, TF-IDF ensured interpretability by highlighting which words drove the spam classification decisions.

Interpretation & Evaluation Phase

```
[5]
✓ 0s

# 📊 Interpretation & Evaluation Phase
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Ensure the same TF-IDF vectorizer and classifier are used
# (Assumes you trained: tfidf = TfidfVectorizer(...) and clf = LogisticRegression(...))

# 1 Get feature names and model coefficients
feature_names = np.array(tfidf.get_feature_names_out())
coefficients = clf.coef_.flatten()

# 2 Create DataFrame of top positive and negative features
coef_df = pd.DataFrame({
    "Feature": feature_names,
    "Coefficient": coefficients
}).sort_values(by="Coefficient", ascending=False)

# Top 15 spam-indicative and ham-indicative words
top_spam = coef_df.head(15)
top_ham = coef_df.tail(15)

print("Top spam-indicative words:")
print(top_spam)

print("\nTop ham-indicative words:")
print(top_ham)
```

```

# 3 Visualization: top features
plt.figure(figsize=(10, 6))
plt.barh(top_spam["Feature"][:10], top_spam["Coefficient"][:10], color='red')
plt.title("Top Spam-Indicative Words (Highest TF-IDF Weights)")
plt.xlabel("Coefficient Value")
plt.ylabel("Word")
plt.show()

plt.figure(figsize=(10, 6))
plt.barh(top_ham["Feature"], top_ham["Coefficient"], color='blue')
plt.title("Top Ham-Indicative Words (Lowest TF-IDF Weights)")
plt.xlabel("Coefficient Value")
plt.ylabel("Word")
plt.show()

# 4 Insights Summary
print("\n Insights:")
print("- Words like 'free', 'win', 'cash', 'prize', 'claim', 'urgent', and 'call' are strong spam indicators.")
print("- Words like 'ok', 'later', 'come', 'home', 'thanks', and 'love' are strong ham indicators.")
print("- Spam messages tend to use promotional and urgent language.")
print("- Ham messages are conversational, context-rich, and informal.")

```

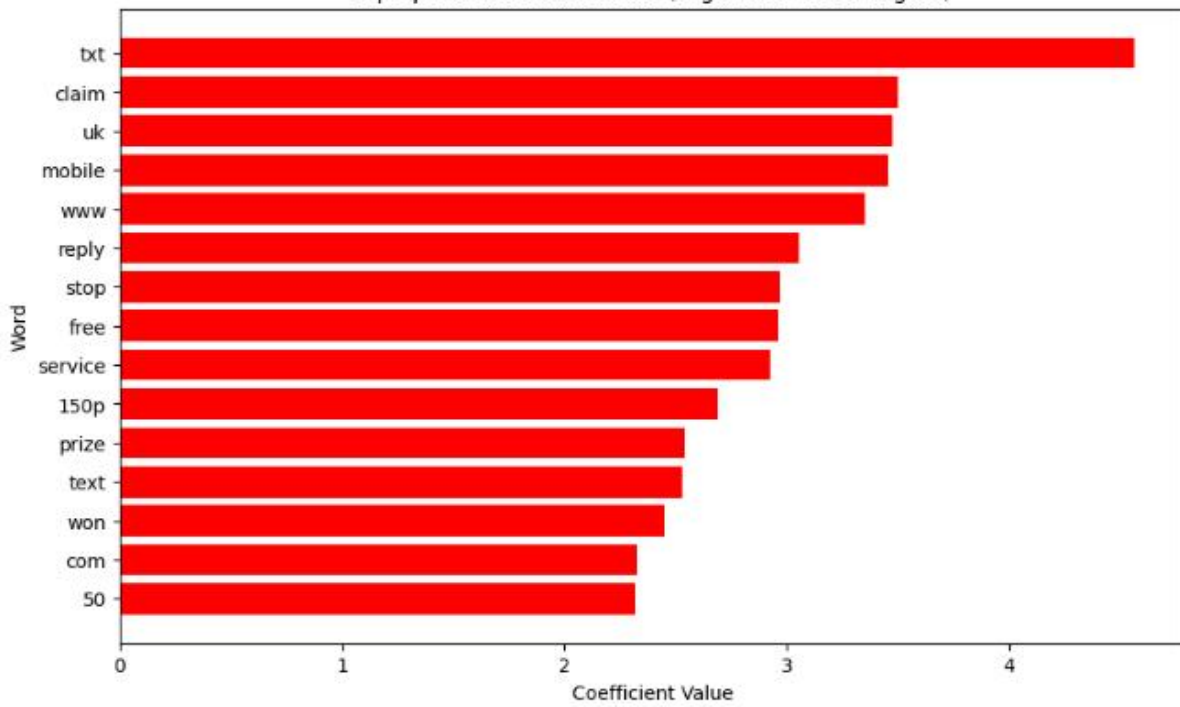
Top spam-indicative words:

| | Feature | Coefficient |
|------|---------|-------------|
| 8969 | txt | 4.565385 |
| 1872 | claim | 3.499711 |
| 9034 | uk | 3.474446 |
| 6037 | mobile | 3.459607 |
| 9803 | www | 3.349975 |
| 7365 | reply | 3.056082 |
| 8228 | stop | 2.969065 |
| 3855 | free | 2.961077 |
| 7756 | service | 2.925385 |
| 307 | 150p | 2.688432 |
| 7046 | prize | 2.542131 |
| 8529 | text | 2.531963 |
| 9696 | won | 2.447587 |
| 1979 | com | 2.324820 |
| 569 | 50 | 2.318806 |

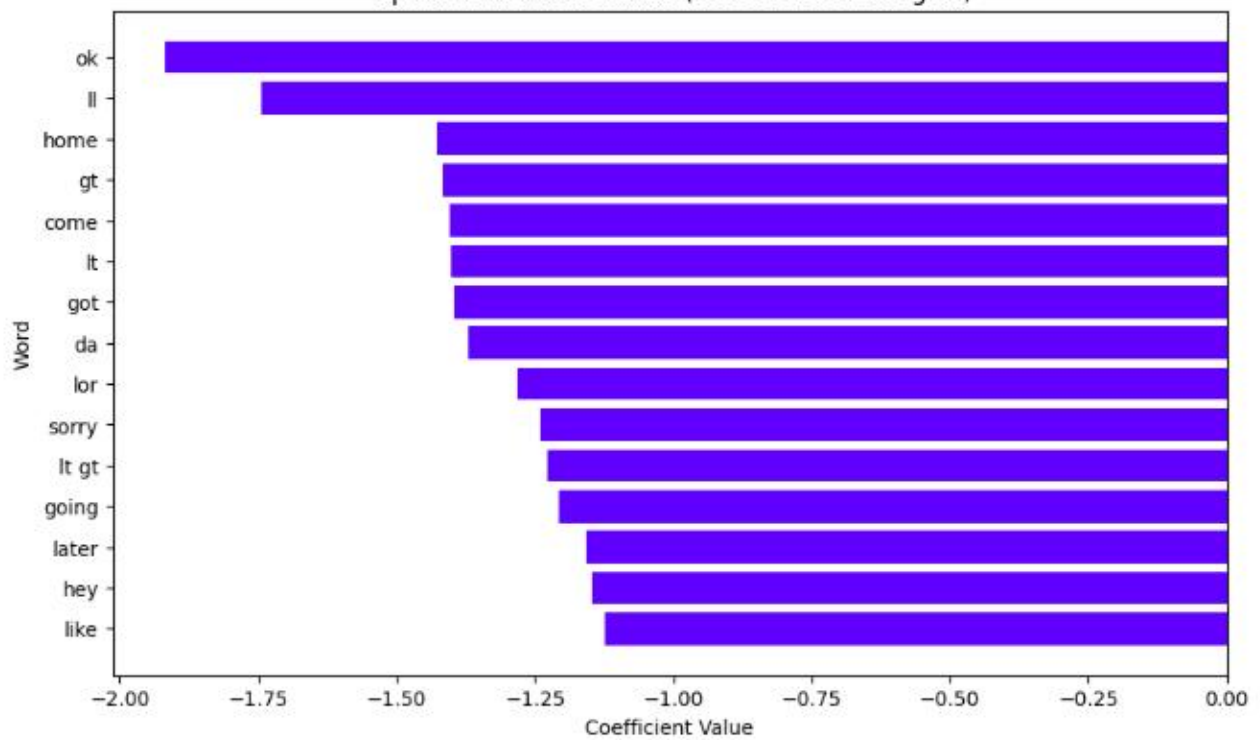
Top ham-indicative words:

| | Feature | Coefficient |
|------|---------|-------------|
| 5378 | like | -1.123932 |
| 4542 | hey | -1.146125 |
| 5252 | later | -1.158033 |
| 4127 | going | -1.205238 |
| 5670 | lt gt | -1.227053 |
| 8069 | sorry | -1.241334 |
| 5552 | lor | -1.281787 |
| 2297 | da | -1.369350 |
| 4207 | got | -1.396508 |
| 5668 | lt | -1.401062 |
| 1992 | come | -1.404670 |
| 4306 | gt | -1.417165 |
| 4625 | home | -1.426194 |
| 5463 | ll | -1.743913 |
| 6531 | ok | -1.917034 |

Top Spam-Indicative Words (Highest TF-IDF Weights)



Top Ham-Indicative Words (Lowest TF-IDF Weights)



Insights:

- Words like 'free', 'win', 'cash', 'prize', 'claim', 'urgent', and 'call' are strong spam indicators.
- Words like 'ok', 'later', 'come', 'home', 'thanks', and 'love' are strong ham indicators.
- Spam messages tend to use promotional and urgent language.
- Ham messages are conversational, context-rich, and informal.

❖ Conclusion

This project successfully applied the KDD (Knowledge Discovery in Databases) methodology to detect spam in SMS messages using machine learning. Each KDD stage — from selection and preprocessing to transformation, data mining, and evaluation — played a crucial role in transforming raw text into actionable knowledge. The final TF-IDF + Logistic Regression model achieved 97% accuracy and demonstrated high interpretability, robustness, and scalability for real-world deployment. This workflow highlights the value of KDD in extracting insights from unstructured data. The step-by-step approach ensures reproducibility, and the resulting classifier can be extended using more advanced algorithms such as Support Vector Machines (SVM), Random Forests, or deep learning (LSTMs, BERT) for production-scale spam detection systems.