Experiment 3: Logistic Regression (from scratch)

About

In this experiment, we implement **Logistic Regression from scratch** to classify messages as **spam or not spam**.
Logistic Regression models the probability that a message belongs to the spam class using the **sigmoid activation function**:

$$P(y = 1|x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

where **w** and **b** are the learnable parameters.
The model is optimized using **gradient descent** to minimize the **binary cross-entropy loss**.

We perform the following experiments:

- Train a **baseline logistic regression** model with default hyperparameters (learning rate, epochs, no regularization).
- Apply **feature scaling** using `StandardScaler` to normalize input features.
- Compare feature representations: **CountVectorizer** vs **TfidfVectorizer**.
- Add **L2 regularization** and experiment with different regularization strengths (λ = 0, 0.01, 0.1, 1.0).
- Plot the **loss curve** across epochs to visualize training dynamics.

Model performance is evaluated using:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-score**
- **Confusion Matrix**

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
from wordcloud import WordCloud
import re
import string
from textblob import TextBlob
import nltk
from nltk.corpus import stopwords
import emoji
nltk.download('punkt')
nltk.download('wordnet')
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from nltk.stem import PorterStemmer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import seaborn as sns
from sklearn.linear_model import LogisticRegression
```

```
[nltk_data] Downloading package punkt to /Users/yug/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/yug/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [94]:
```
df = pd.read_csv('spam.csv')
df.head()
```

Out[94]:

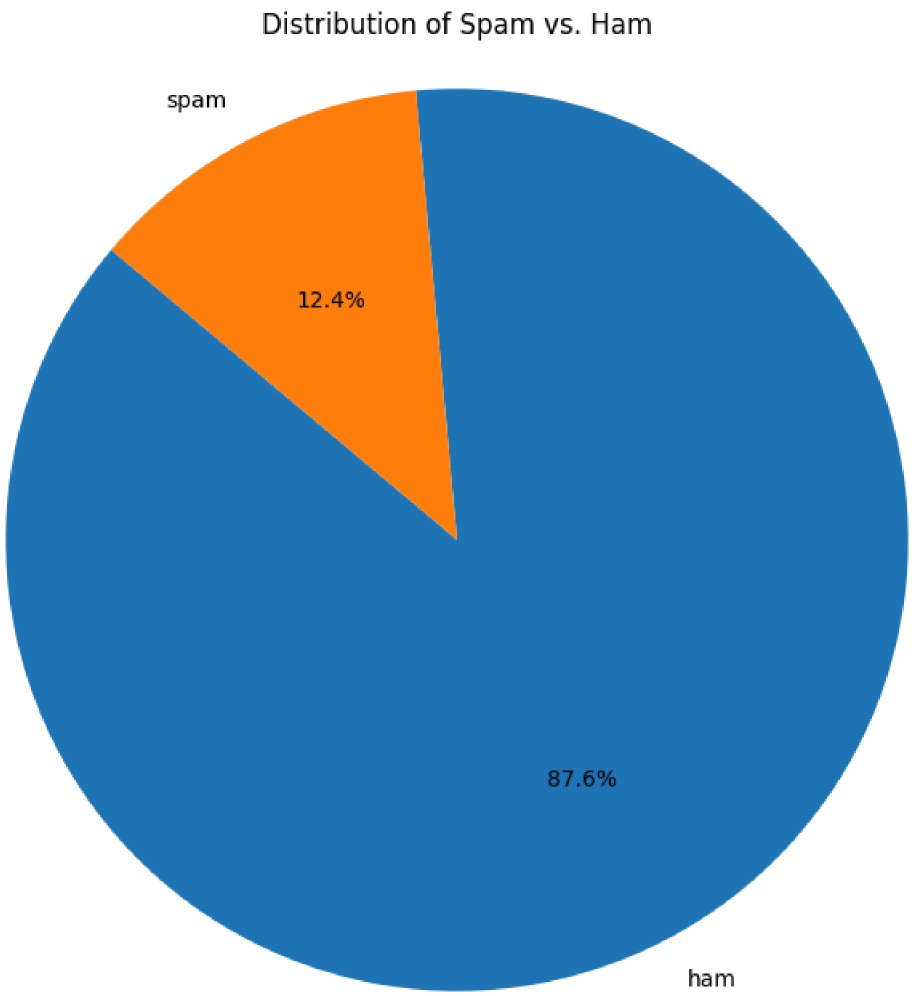
	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [95]:
```
df.shape
```
Out[95]: (5572, 2)

In [96]:
```
# Check for null values
df.isnull().sum()
```
Out[96]:
```
Category    0
Message     0
dtype: int64
```
In [97]:
```
# Find duplicates and drop them
df.duplicated().sum()
```
Out[97]: np.int64(415)

In [98]:
```
df.drop_duplicates(keep='first', inplace=True)
```
In [99]:
```
df.shape
```
Out[99]: (5157, 2)

In [100…
```
#now should be 0 dupplicates
df.duplicated().sum()
```
Out[100… np.int64(0)

In [101…
```
# Calculate the count of each label
category_counts = df['Category'].value_counts()

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Spam vs. Ham')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```


TEXT VISUALIZATION

In [102…

```python
# Iterate through unique categories
for category in df['Category'].unique():
    # Filter the DataFrame for the current category
    filtered_df = df[df['Category'] == category]

    # Concatenate all text data for the current category
    text = ' '.join(filtered_df['Message'])

    # Generate word cloud
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

    # Plot the word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for Category: {category}')
    plt.axis('off')
    plt.show()
```

Word Cloud for Category: ham

Word Cloud for Category: spam

In [103…

```python
le = LabelEncoder()
df['Category']=le.fit_transform(df['Category'])
df.head()
```

Out[103...

	Category	Message
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

Text preprocessing

1. Lower Casing
2. Remove Extra White Spaces
3. Remove HTML Tags
4. Remove URLs
5. Remove Punctuations
6. Remove Special Characters
7. Remove Numeric Values
8. Remove Non-alpha Numeric
9. Handling StopWords¶
10. Handling Emojis
11. Stemming

In [104...

```
# Convert 'Text' column to lowercase
df['Message'] = df['Message'].str.lower()
df.head()
```

Out[104...

	Category	Message
0	0	go until jurong point, crazy.. available only ...
1	0	ok lar... joking wif u oni...
2	1	free entry in 2 a wkly comp to win fa cup fina...
3	0	u dun say so early hor... u c already then say...
4	0	nah i don't think he goes to usf, he lives aro...

In [105...

```
# Function to remove HTML tags from text safely
def remove_html_tags(text):
    if isinstance(text, str):  # only process if it's a string
        clean = re.compile('<.*?>')
        return re.sub(clean, '', text)
    else:
        return ""  # if text is None/NaN, return empty string

# Apply to dataframe
df['Message'] = df['Message'].apply(remove_html_tags)
df.head()
```

Out[105...

	Category	Message
0	0	go until jurong point, crazy.. available only ...
1	0	ok lar... joking wif u oni...
2	1	free entry in 2 a wkly comp to win fa cup fina...
3	0	u dun say so early hor... u c already then say...
4	0	nah i don't think he goes to usf, he lives aro...

In [106...

```
# Define a function to remove URLs using regular expressions
def remove_urls(text):
```

```
    if isinstance(text, str):
        return re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    return ""

# Apply the function to the 'Text' column
df['Message'] = df['Message'].apply(remove_urls)
df.head()
```

Out[106…

	Category	Message
0	0	go until jurong point, crazy.. available only ...
1	0	ok lar... joking wif u oni...
2	1	free entry in 2 a wkly comp to win fa cup fina...
3	0	u dun say so early hor... u c already then say...
4	0	nah i don't think he goes to usf, he lives aro...

In [107…

```
# Function to remove special characters (keep letters, numbers, and spaces)
def remove_special_characters(text):
    if isinstance(text, str):
        return re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return ""

# Apply the function to the 'Message' column
df['Message'] = df['Message'].apply(remove_special_characters)
df.head()
```

Out[107…

	Category	Message
0	0	go until jurong point crazy available only in ...
1	0	ok lar joking wif u oni
2	1	free entry in 2 a wkly comp to win fa cup fina...
3	0	u dun say so early hor u c already then say
4	0	nah i dont think he goes to usf he lives aroun...

In [108…

```
# Function to remove numeric values
def remove_numeric(text):
    if isinstance(text, str):
        return re.sub(r'\d+', '', text)
    return ""

# Apply the function to the "Message" column
df['Message'] = df['Message'].apply(remove_numeric)
df.head()
```

Out[108…

	Category	Message
0	0	go until jurong point crazy available only in ...
1	0	ok lar joking wif u oni
2	1	free entry in a wkly comp to win fa cup final...
3	0	u dun say so early hor u c already then say
4	0	nah i dont think he goes to usf he lives aroun...

In [109…

```
# Function to remove non-alphanumeric characters (leave only letters and numbers)
def remove_non_alphanumeric(text):
    if isinstance(text, str):
        return re.sub(r'[^a-zA-Z0-9 ]', '', text)
    return ""

# Apply the function to the "Message" column
df['Message'] = df['Message'].apply(remove_non_alphanumeric)
df.head()
```

Out [109…

	Category	Message
0	0	go until jurong point crazy available only in ...
1	0	ok lar joking wif u oni
2	1	free entry in a wkly comp to win fa cup final...
3	0	u dun say so early hor u c already then say
4	0	nah i dont think he goes to usf he lives aroun...

In [110…

```
# Define a dictionary of chat word mappings
chat_words = {
    "AFAIK": "As Far As I Know",
    "AFK": "Away From Keyboard",
    "ASAP": "As Soon As Possible",
    "ATK": "At The Keyboard",
    "ATM": "At The Moment",
    "A3": "Anytime, Anywhere, Anyplace",
    "BAK": "Back At Keyboard",
    "BBL": "Be Back Later",
    "BBS": "Be Back Soon",
    "BFN": "Bye For Now",
    "B4N": "Bye For Now",
    "BRB": "Be Right Back",
    "BRT": "Be Right There",
    "BTW": "By The Way",
    "B4": "Before",
    "B4N": "Bye For Now",
    "CU": "See You",
    "CUL8R": "See You Later",
    "CYA": "See You",
    "FAQ": "Frequently Asked Questions",
    "FC": "Fingers Crossed",
    "FWIW": "For What It's Worth",
    "FYI": "For Your Information",
    "GAL": "Get A Life",
    "GG": "Good Game",
    "GN": "Good Night",
    "GMTA": "Great Minds Think Alike",
    "GR8": "Great!",
    "G9": "Genius",
    "IC": "I See",
    "ICQ": "I Seek you (also a chat program)",
    "ILU": "ILU: I Love You",
    "IMHO": "In My Honest/Humble Opinion",
    "IMO": "In My Opinion",
    "IOW": "In Other Words",
    "IRL": "In Real Life",
    "KISS": "Keep It Simple, Stupid",
    "LDR": "Long Distance Relationship",
    "LMAO": "Laugh My A.. Off",
    "LOL": "Laughing Out Loud",
    "LTNS": "Long Time No See",
    "L8R": "Later",
    "MTE": "My Thoughts Exactly",
    "M8": "Mate",
    "NRN": "No Reply Necessary",
    "OIC": "Oh I See",
    "PITA": "Pain In The A..",
    "PRT": "Party",
    "PRW": "Parents Are Watching",
    "QPSA?": "Que Pasa?",
    "ROFL": "Rolling On The Floor Laughing",
    "ROFLOL": "Rolling On The Floor Laughing Out Loud",
    "ROTFLMAO": "Rolling On The Floor Laughing My A.. Off",
    "SK8": "Skate",
    "STATS": "Your sex and age",
    "ASL": "Age, Sex, Location",
    "THX": "Thank You",
    "TTFN": "Ta-Ta For Now!",
    "TTYL": "Talk To You Later",
    "U": "You",
    "U2": "You Too",
    "U4E": "Yours For Ever",
    "WB": "Welcome Back",
    "WTF": "What The F...",
    "WTG": "Way To Go!",
    "WUF": "Where Are You From?",
```

```
    "W8": "Wait...",
    "7K": "Sick:-D Laugher",
    "TFW": "That feeling when",
    "MFW": "My face when",
    "MRW": "My reaction when",
    "IFYP": "I feel your pain",
    "TNTL": "Trying not to laugh",
    "JK": "Just kidding",
    "IDC": "I don't care",
    "ILY": "I love you",
    "IMU": "I miss you",
    "ADIH": "Another day in hell",
    "ZZZ": "Sleeping, bored, tired",
    "WYWH": "Wish you were here",
    "TIME": "Tears in my eyes",
    "BAE": "Before anyone else",
    "FIMH": "Forever in my heart",
    "BSAAW": "Big smile and a wink",
    "BWL": "Bursting with laughter",
    "BFF": "Best friends forever",
    "CSL": "Can't stop laughing"
}
```

In [111…

```
# Function to replace chat words with their full forms
def replace_chat_words(text):
    if isinstance(text, str):
        words = text.split()
        replaced = [chat_words[word.lower()] if word.lower() in chat_words else word for word in words]
        return ' '.join(replaced)
    return ""

# Apply replace_chat_words function to 'Text' column
df['Message'] = df['Message'].apply(replace_chat_words)
df.head()
```

Out[111…

	Category	Message
0	0	go until jurong point crazy available only in ...
1	0	ok lar joking wif u oni
2	1	free entry in a wkly comp to win fa cup final ...
3	0	u dun say so early hor u c already then say
4	0	nah i dont think he goes to usf he lives aroun...

In [112…

```
# Function to remove emojis from text
def remove_emojis(text):
    return emoji.demojize(text)

# Apply remove_emojis function to 'Text' column
df['Message'] = df['Message'].apply(remove_emojis)
```

In [113…

```
# Download NLTK stopwords corpus
nltk.download('stopwords')

# Get English stopwords from NLTK
stop_words = set(stopwords.words('english'))

# Function to remove stop words from text
def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Apply remove_stopwords function to 'Text' column
df['Message'] = df['Message'].apply(remove_stopwords)
df.head()
```

```
[nltk_data] Downloading package stopwords to /Users/yug/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[113...

	Category	Message
0	0	go jurong point crazy available bugis n great ...
1	0	ok lar joking wif u oni
2	1	free entry wkly comp win fa cup final tkts st ...
3	0	u dun say early hor u c already say
4	0	nah dont think goes usf lives around though

In [114...

```
# Initialize the Porter Stemmer
porter_stemmer = PorterStemmer()

# Apply stemming
df['Message_stemmed'] = df['Message'].apply(lambda x: ' '.join([porter_stemmer.stem(word) for word in x.spli
df.head()
```

Out[114...

	Category	Message	Message_stemmed
0	0	go jurong point crazy available bugis n great ...	go jurong point crazi avail bugi n great world...
1	0	ok lar joking wif u oni	ok lar joke wif u oni
2	1	free entry wkly comp win fa cup final tkts st ...	free entri wkli comp win fa cup final tkt st m...
3	0	u dun say early hor u c already say	u dun say earli hor u c alreadi say
4	0	nah dont think goes usf lives around though	nah dont think goe usf live around though

In [115...

```
#Convert text to numbers using bag of words
vectorizer=CountVectorizer()
X=vectorizer.fit_transform(df['Message_stemmed']).toarray()
y = df['Category']
```

In [128...

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(4125, 7082) (1032, 7082)
(4125,) (1032,)
```

Solve this problem using Logistic Regression(using numpy from scratch)

In [ ]:

```
class LogisticRegressionScratch:
    def __init__(self, learning_rate=0.01, epochs=1000, reg_lambda=0.0):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.reg_lambda = reg_lambda

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        n_samples, n_features = X.shape
        # Random initialization improves training dynamics
        self.weights = np.random.randn(n_features) * 0.01
        self.bias = 0
        self.losses = []

        for _ in range(self.epochs):
            linear_model = np.dot(X, self.weights) + self.bias
            y_pred = self.sigmoid(linear_model)

            # Loss (with L2 regularization)
            loss = (-1/n_samples) * (np.dot(y, np.log(y_pred + 1e-15)) +
                                     np.dot((1-y), np.log(1 - y_pred + 1e-15)))
            reg_term = (self.reg_lambda / (2 * n_samples)) * np.sum(self.weights ** 2)
            self.losses.append(loss + reg_term)

            # Gradients
            dw = (1/n_samples) * np.dot(X.T, (y_pred - y)) + (self.reg_lambda/n_samples) * self.weights
            db = (1/n_samples) * np.sum(y_pred - y)

            # Update weights
            self.weights -= self.learning_rate * dw
```

```
            self.bias -= self.learning_rate * db

    def predict_proba(self, X):
        return self.sigmoid(np.dot(X, self.weights) + self.bias)

    def predict(self, X):
        return np.array([1 if i > 0.5 else 0 for i in self.predict_proba(X)])

# ----------------------------
# Utility Functions
# ----------------------------
def evaluate_model(y_test, y_pred):
    """Return metrics dictionary"""
    return {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "Confusion Matrix": confusion_matrix(y_test, y_pred)
    }

def run_logistic_regression(X, y, vectorizer_name="Count", reg_lambda=0.0,
                            learning_rate=0.01, epochs=1000, scale=False, plot_loss=True):
    """Train and evaluate logistic regression"""
    # Split dataset
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Feature scaling
    if scale:
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

    # Train model
    model = LogisticRegressionScratch(learning_rate=learning_rate,
                                      epochs=epochs,
                                      reg_lambda=reg_lambda)
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Evaluation
    metrics = evaluate_model(y_test, y_pred)

    # Plot loss curve
    if plot_loss:
        plt.plot(model.losses)
        plt.title(f"Loss Curve (Vectorizer={vectorizer_name}, λ={reg_lambda})")
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.show()

    return metrics

def compare_vectorizers(df, lambdas=[0.0, 0.01, 0.1, 1.0], scale=False):
    """Compare CountVectorizer vs TfidfVectorizer for different λ"""
    results = []

    vectorizers = {
        "Count": CountVectorizer(),
        "TF-IDF": TfidfVectorizer()
    }

    y = df['Category'].map({'ham': 0, 'spam': 1}).values

    for vec_name, vectorizer in vectorizers.items():
        X = vectorizer.fit_transform(df['Message_stemmed']).toarray()

        for lam in lambdas:
            print(f"\n----- {vec_name} Vectorizer | λ={lam} | Scaling={scale} -----")
            metrics = run_logistic_regression(X, y,
                                              vectorizer_name=vec_name,
                                              reg_lambda=lam,
                                              scale=scale,
                                              plot_loss=True)
            row = {
                "Model": "Logistic Regression",
```

```
            "Vectorizer": vec_name,
            "λ": lam,
            **metrics
        }
        results.append(row)

return results
```

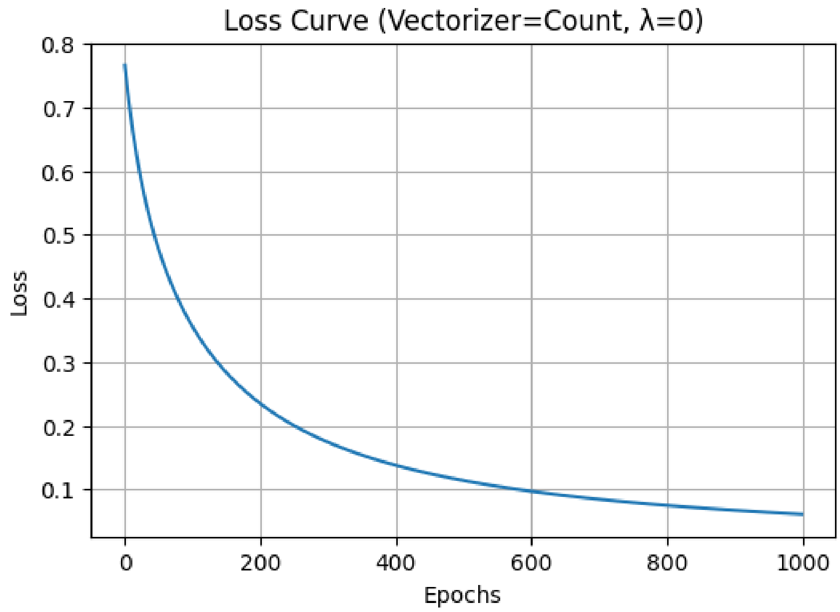
In [141…

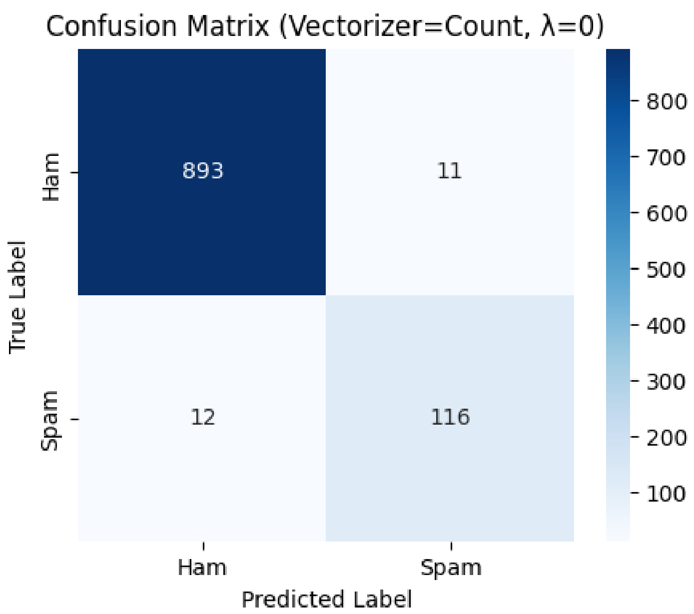
```
def run_logistic_regression_experiments_with_comparison(df, lambdas=[0, 0.01, 0.1, 1], scale=True, plot_loss
    """
    Run Logistic Regression experiments with Count and TF-IDF vectorizers,
    multiple regularization values (λ), and generate a comparison table
    including confusion matrices.
    """
    results = []

    # Ensure labels are 0/1 integers
    y = df['Category'].astype(int).values

    vectorizers = {
        "Count": CountVectorizer(),
        "TF-IDF": TfidfVectorizer()
    }

    for vec_name, vectorizer in vectorizers.items():
        # Transform text
        X = vectorizer.fit_transform(df['Message_stemmed']).toarray()

        for lam in lambdas:
            print(f"\nRunning Logistic Regression | Vectorizer={vec_name} | λ={lam}")

            # Split dataset
            X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.2, random_state=42, stratify=y
            )

            # Feature scaling
            if scale:
                scaler = StandardScaler()
                X_train = scaler.fit_transform(X_train)
                X_test = scaler.transform(X_test)

            # Train model
            model = LogisticRegressionScratch(
                learning_rate=0.01,
                epochs=1000,
                reg_lambda=lam
            )
            model.fit(X_train, y_train)

            # Predictions
            y_pred = model.predict(X_test)

            # Confusion matrix
            cm = confusion_matrix(y_test, y_pred)

            # Metrics
            metrics = {
                "Accuracy": accuracy_score(y_test, y_pred),
                "Precision": precision_score(y_test, y_pred),
                "Recall": recall_score(y_test, y_pred),
                "F1": f1_score(y_test, y_pred),
                "Confusion_Matrix": cm
            }

            # Store results
            row = {
                "Model": "Logistic Regression",
                "Vectorizer": vec_name,
                "λ": lam,
                **metrics
            }
            results.append(row)

            # Print metrics
            print(f"Accuracy: {metrics['Accuracy']:.4f}, Precision: {metrics['Precision']:.4f}, "
                  f"Recall: {metrics['Recall']:.4f}, F1: {metrics['F1']:.4f}")

            # Plot loss curve
            if plot_loss:
```

```python
                plt.figure(figsize=(6,4))
                plt.plot(model.losses)
                plt.title(f"Loss Curve (Vectorizer={vec_name}, λ={lam})")
                plt.xlabel("Epochs")
                plt.ylabel("Loss")
                plt.grid(True)
                plt.show()

            # Plot confusion matrix
            plt.figure(figsize=(5,4))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                        xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])
            plt.title(f"Confusion Matrix (Vectorizer={vec_name}, λ={lam})")
            plt.ylabel("True Label")
            plt.xlabel("Predicted Label")
            plt.show()

    # Convert to DataFrame
    results_df = pd.DataFrame(results)

    # Round metrics for easier comparison
    for col in ['Accuracy', 'Precision', 'Recall', 'F1']:
        results_df[col] = results_df[col].round(4)

    print("\n📊 Final Comparison Table:")
    display(results_df[['Model','Vectorizer','λ','Accuracy','Precision','Recall','F1']])

    return results_df
```

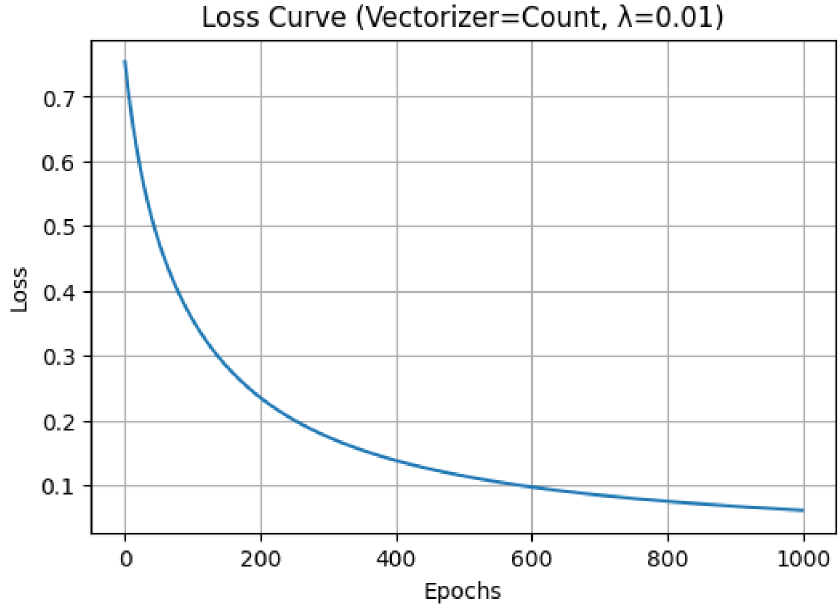
In [150…
```python
final_logistic_results = run_logistic_regression_experiments_with_comparison(df, lambdas=[0, 0.01, 0.1, 1],
```

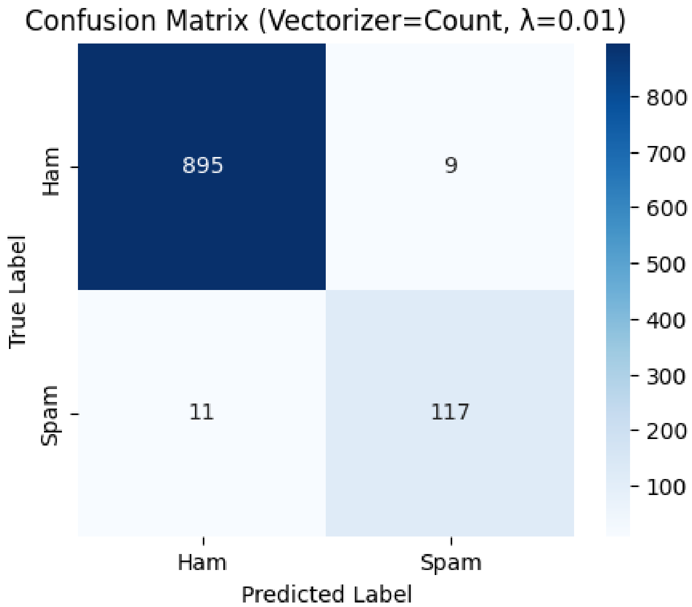
```
Running Logistic Regression | Vectorizer=Count | λ=0
Accuracy: 0.9777, Precision: 0.9134, Recall: 0.9062, F1: 0.9098
```



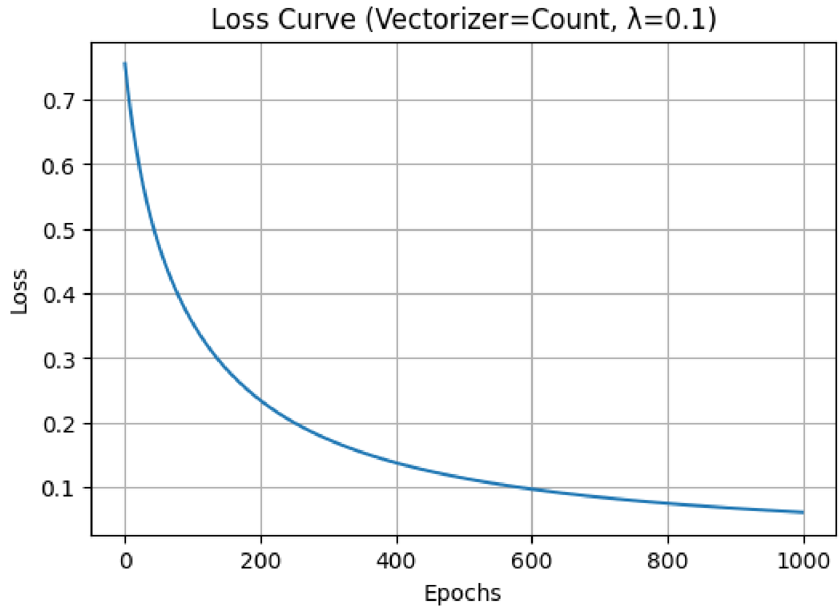


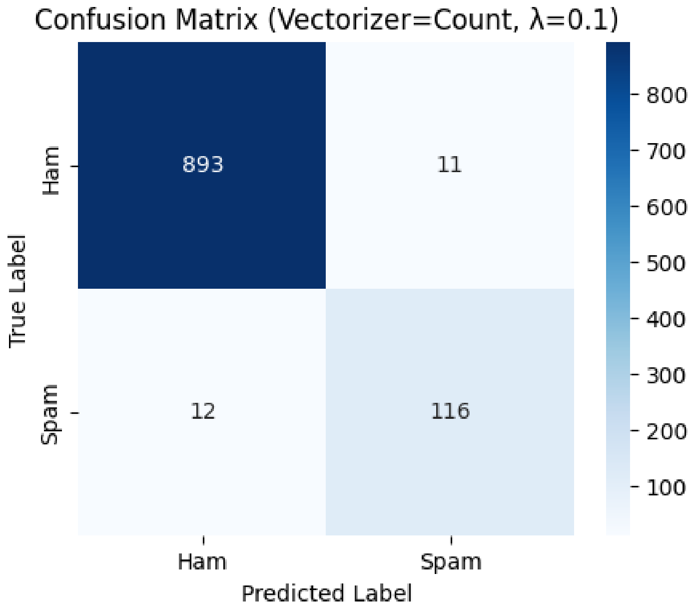
```
Running Logistic Regression | Vectorizer=Count | λ=0.01
Accuracy: 0.9806, Precision: 0.9286, Recall: 0.9141, F1: 0.9213
```



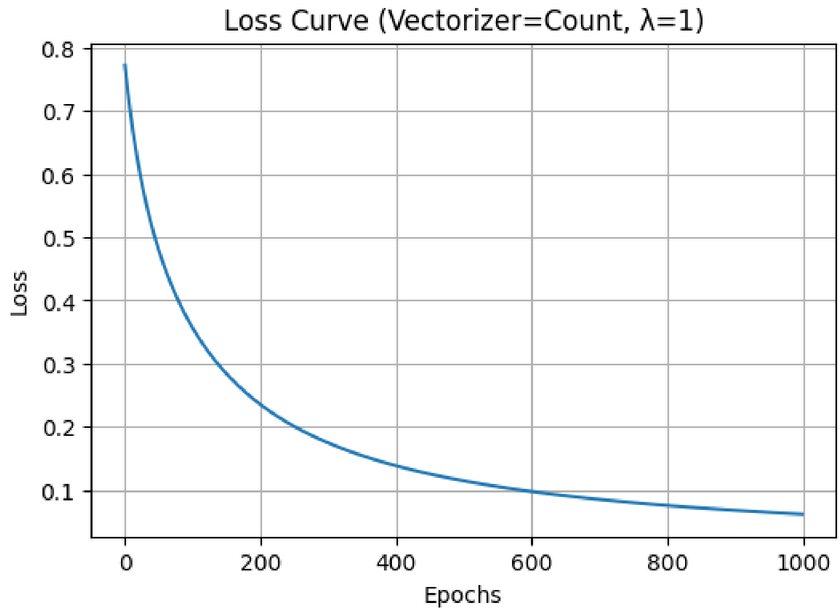


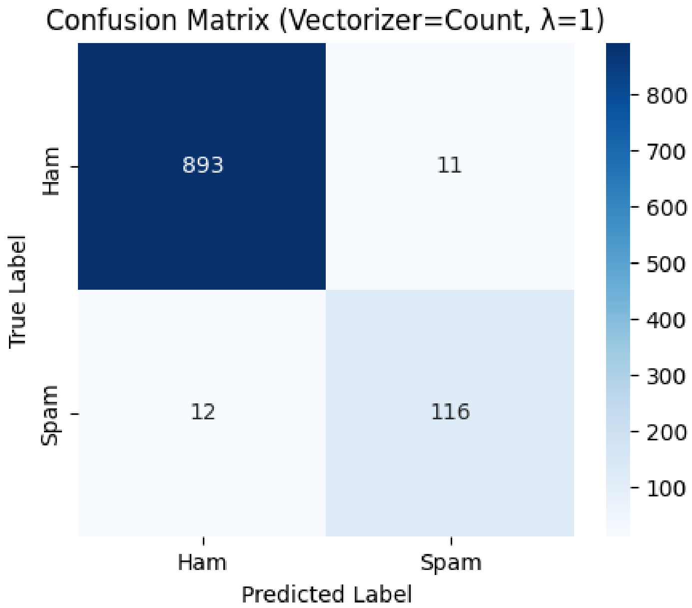

```
Running Logistic Regression | Vectorizer=Count | λ=0.1
Accuracy: 0.9777, Precision: 0.9134, Recall: 0.9062, F1: 0.9098
```



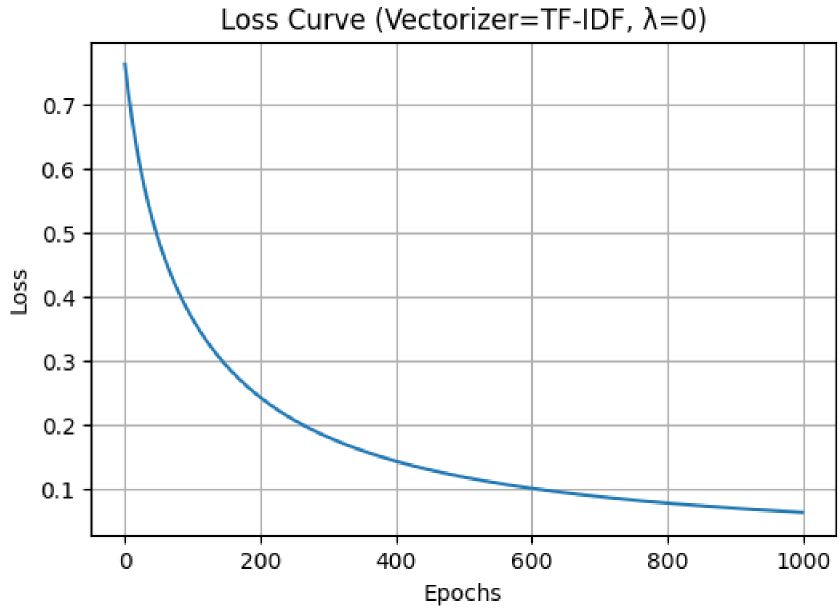


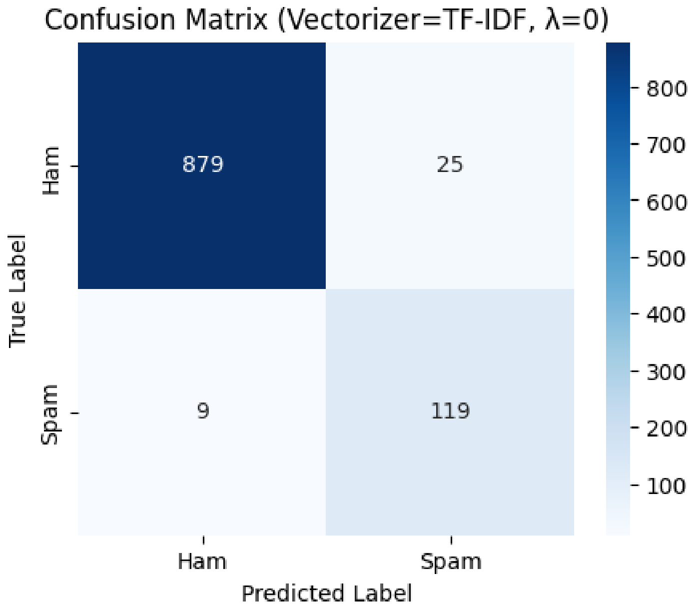

```
Running Logistic Regression | Vectorizer=Count | λ=1
Accuracy: 0.9777, Precision: 0.9134, Recall: 0.9062, F1: 0.9098
```



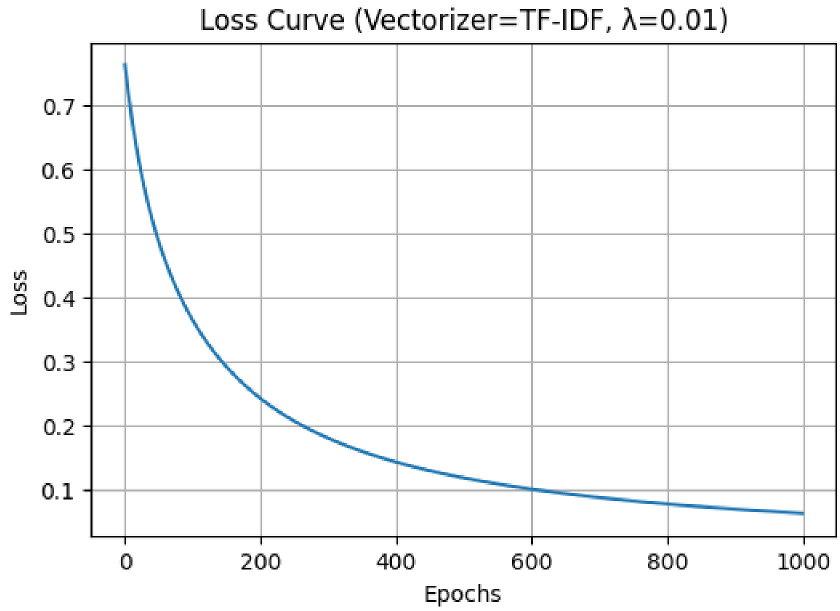


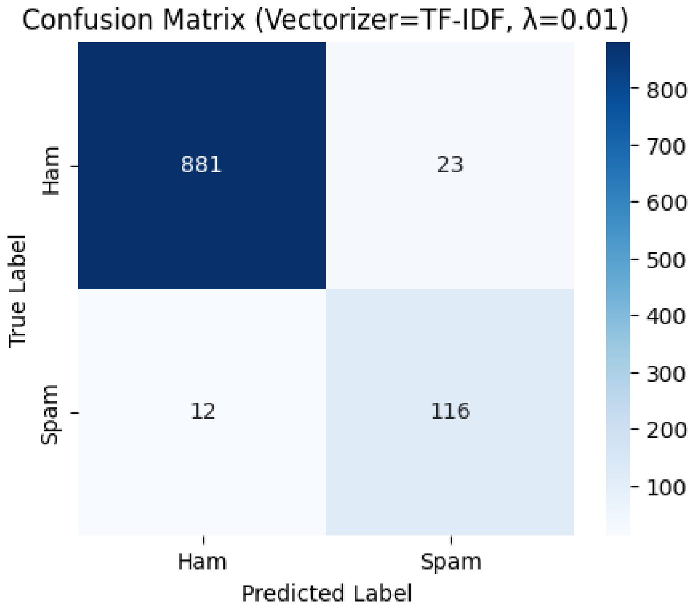

```
Running Logistic Regression | Vectorizer=TF-IDF | λ=0
Accuracy: 0.9671, Precision: 0.8264, Recall: 0.9297, F1: 0.8750
```



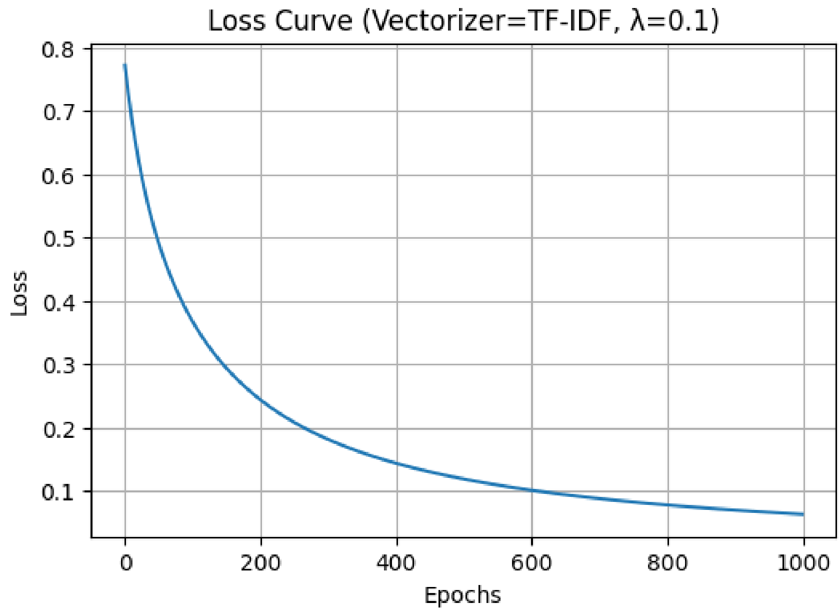


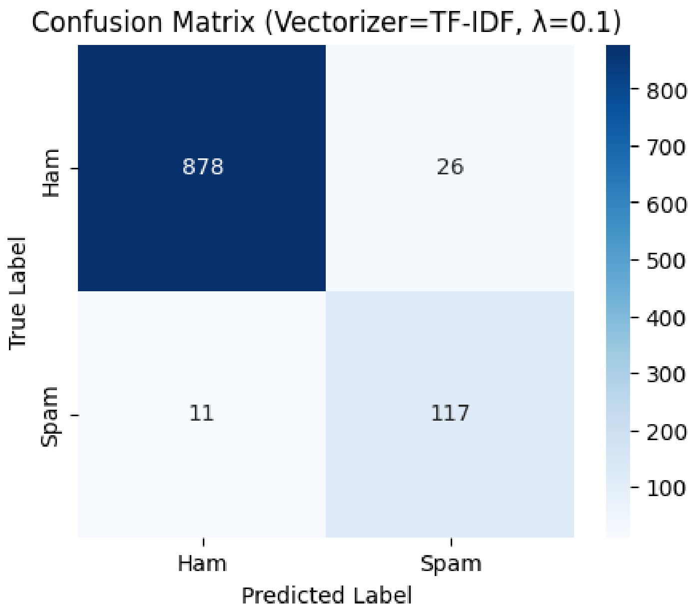

```
Running Logistic Regression | Vectorizer=TF-IDF | λ=0.01
Accuracy: 0.9661, Precision: 0.8345, Recall: 0.9062, F1: 0.8689
```



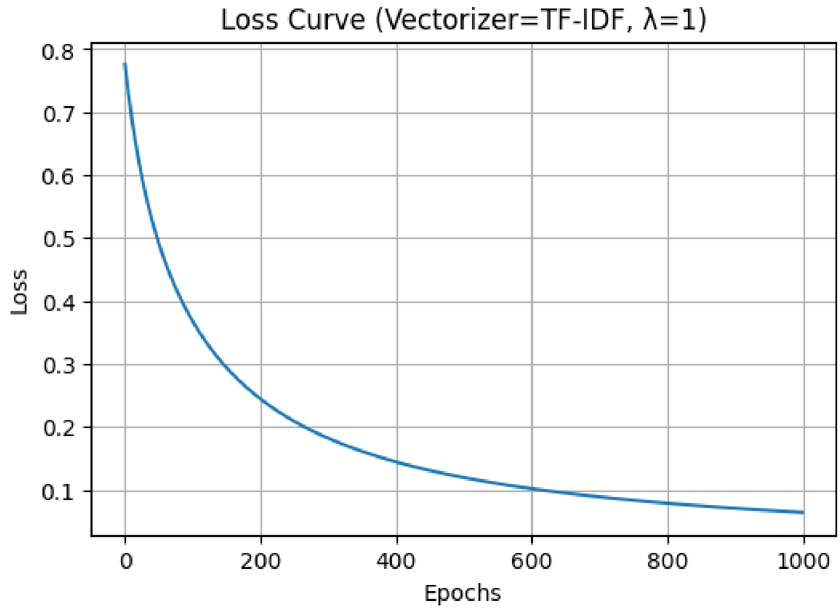


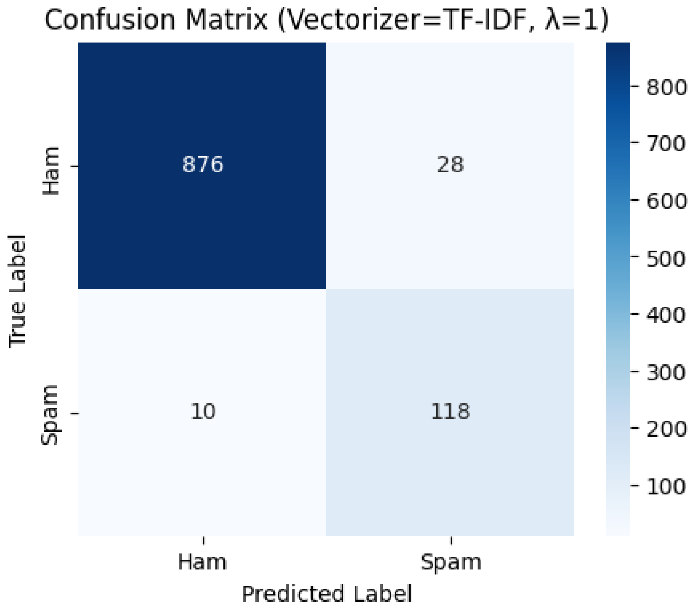

```
Running Logistic Regression | Vectorizer=TF-IDF | λ=0.1
Accuracy: 0.9641, Precision: 0.8182, Recall: 0.9141, F1: 0.8635
```






```
Running Logistic Regression | Vectorizer=TF-IDF | λ=1
Accuracy: 0.9632, Precision: 0.8082, Recall: 0.9219, F1: 0.8613
```





📊 Final Comparison Table:

	Model	Vectorizer	λ	Accuracy	Precision	Recall	F1
0	Logistic Regression	Count	0.00	0.9777	0.9134	0.9062	0.9098
1	Logistic Regression	Count	0.01	0.9806	0.9286	0.9141	0.9213
2	Logistic Regression	Count	0.10	0.9777	0.9134	0.9062	0.9098
3	Logistic Regression	Count	1.00	0.9777	0.9134	0.9062	0.9098
4	Logistic Regression	TF-IDF	0.00	0.9671	0.8264	0.9297	0.8750
5	Logistic Regression	TF-IDF	0.01	0.9661	0.8345	0.9062	0.8689
6	Logistic Regression	TF-IDF	0.10	0.9641	0.8182	0.9141	0.8635
7	Logistic Regression	TF-IDF	1.00	0.9632	0.8082	0.9219	0.8613

using logistic regression from sklearn

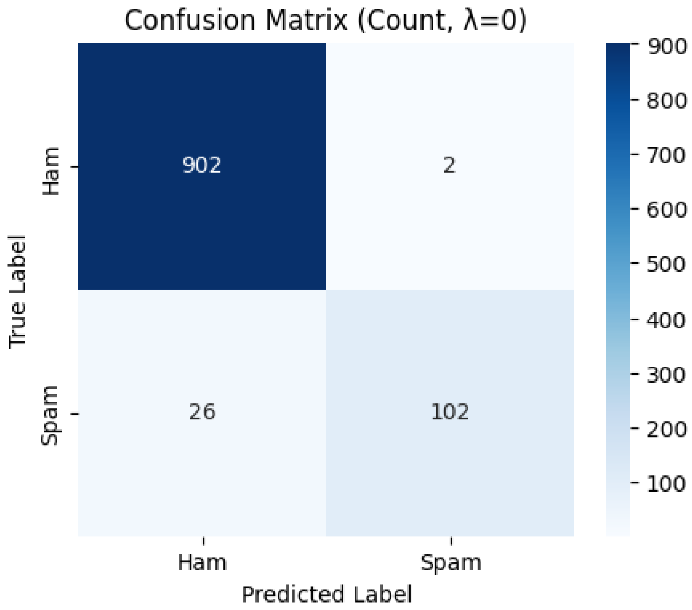
In [ ]:
```
def run_logistic_regression_sklearn(df, lambdas=[0, 0.01, 0.1, 1], scale=True):
    """
    Run Logistic Regression using sklearn on both Count and TF-IDF vectors
    for multiple regularization values λ and generate metrics, confusion matrices, and comparison table.
    """
    results = []

    # Labels
    y = df['Category'].astype(int).values
    vectorizers = {
        "Count": CountVectorizer(),
        "TF-IDF": TfidfVectorizer()
    }
```

```python
    for vec_name, vectorizer in vectorizers.items():
        # Transform text
        X = vectorizer.fit_transform(df['Message_stemmed']).toarray()

        for lam in lambdas:
            print(f"\nRunning Logistic Regression | Vectorizer={vec_name} | λ={lam}")

            # Split dataset
            X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.2, random_state=42, stratify=y
            )

            # Feature scaling
            if scale:
                scaler = StandardScaler()
                X_train = scaler.fit_transform(X_train)
                X_test = scaler.transform(X_test)

            # Convert λ to C (inverse regularization)
            # Avoid division by zero
            C_val = 1.0 if lam == 0 else 1.0 / lam

            # Train sklearn Logistic Regression
            model = LogisticRegression(
                C=C_val,
                penalty='l2',
                solver='lbfgs',
                max_iter=1000
            )
            model.fit(X_train, y_train)

            # Predictions
            y_pred = model.predict(X_test)

            # Metrics
            cm = confusion_matrix(y_test, y_pred)
            metrics = {
                "Accuracy": accuracy_score(y_test, y_pred),
                "Precision": precision_score(y_test, y_pred),
                "Recall": recall_score(y_test, y_pred),
                "F1": f1_score(y_test, y_pred),
                "Confusion_Matrix": cm
            }

            row = {
                "Model": "Logistic Regression (sklearn)",
                "Vectorizer": vec_name,
                "λ": lam,
                **metrics
            }
            results.append(row)

            # Print metrics
            print(f"Accuracy: {metrics['Accuracy']:.4f}, Precision: {metrics['Precision']:.4f}, "
                  f"Recall: {metrics['Recall']:.4f}, F1: {metrics['F1']:.4f}")

            # Plot confusion matrix
            plt.figure(figsize=(5,4))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                        xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])
            plt.title(f"Confusion Matrix ({vec_name}, λ={lam})")
            plt.ylabel("True Label")
            plt.xlabel("Predicted Label")
            plt.show()

    # Convert to DataFrame
    results_df = pd.DataFrame(results)

    # Round metrics for comparison
    for col in ['Accuracy', 'Precision', 'Recall', 'F1']:
        results_df[col] = results_df[col].round(4)

    print("\n📊 Final Comparison Table:")
    display(results_df[['Model','Vectorizer','λ','Accuracy','Precision','Recall','F1']])

    return results_df
```

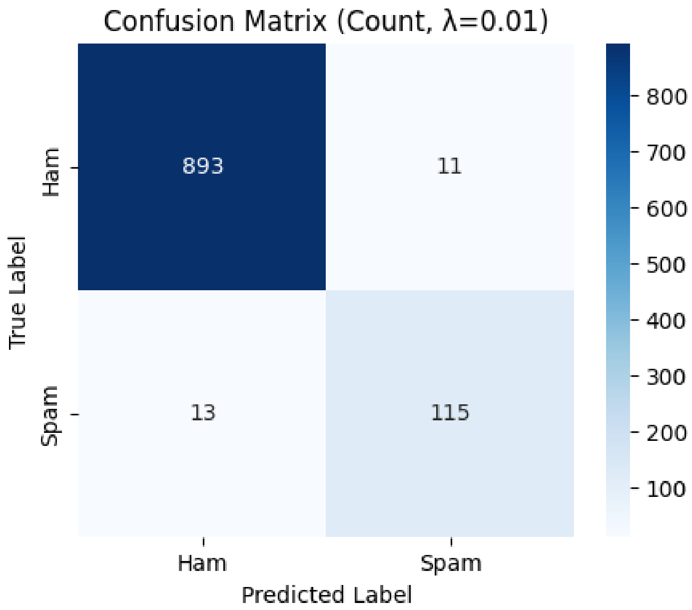
In [144…

```python
final_lr_results_sklearn = run_logistic_regression_sklearn(df, lambdas=[0, 0.01, 0.1, 1], scale=True)
```

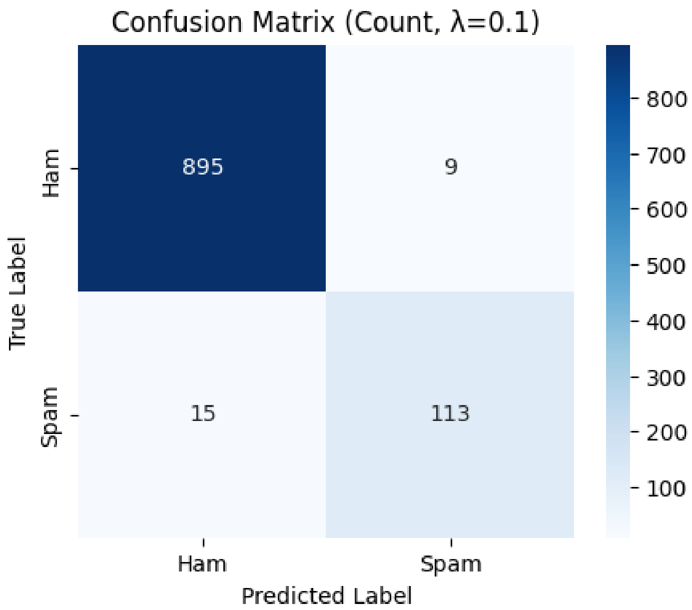
```
Running Logistic Regression | Vectorizer=Count | λ=0
Accuracy: 0.9729, Precision: 0.9808, Recall: 0.7969, F1: 0.8793
```



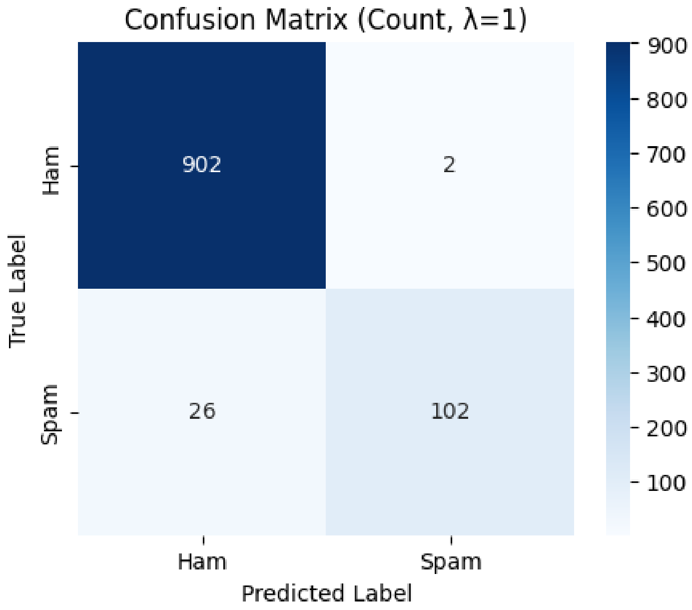

```
Running Logistic Regression | Vectorizer=Count | λ=0.01
Accuracy: 0.9767, Precision: 0.9127, Recall: 0.8984, F1: 0.9055
```



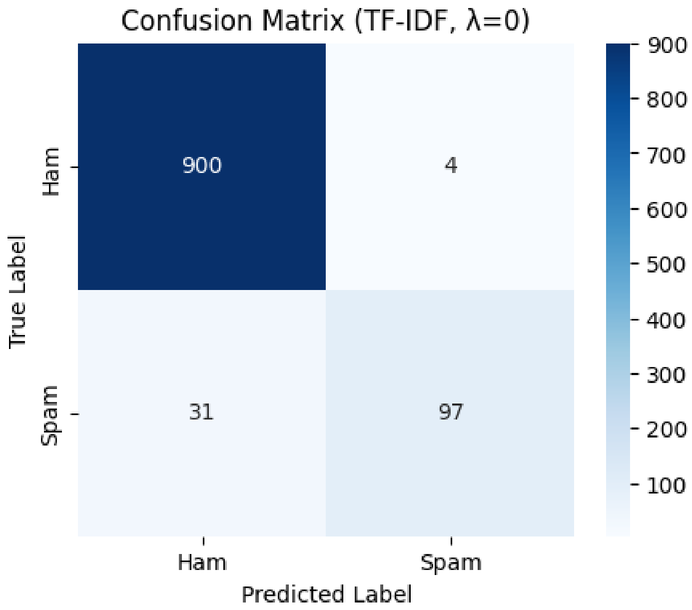

```
Running Logistic Regression | Vectorizer=Count | λ=0.1
Accuracy: 0.9767, Precision: 0.9262, Recall: 0.8828, F1: 0.9040
```



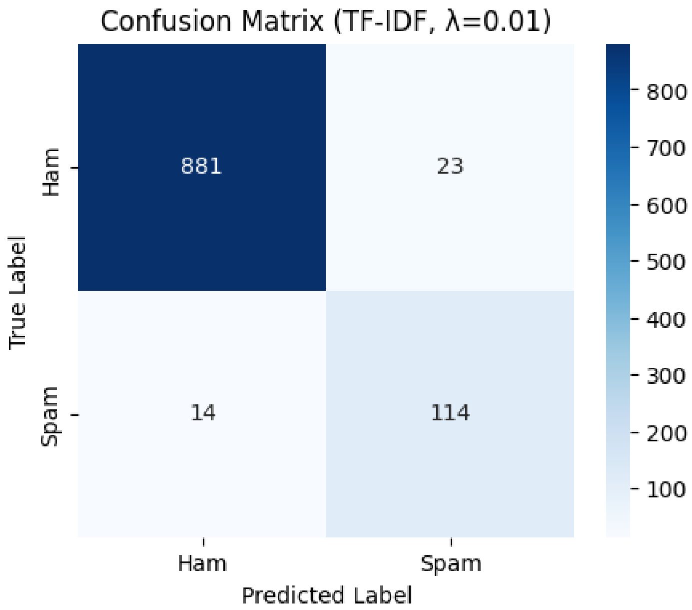

```
Running Logistic Regression | Vectorizer=Count | λ=1
Accuracy: 0.9729, Precision: 0.9808, Recall: 0.7969, F1: 0.8793
```



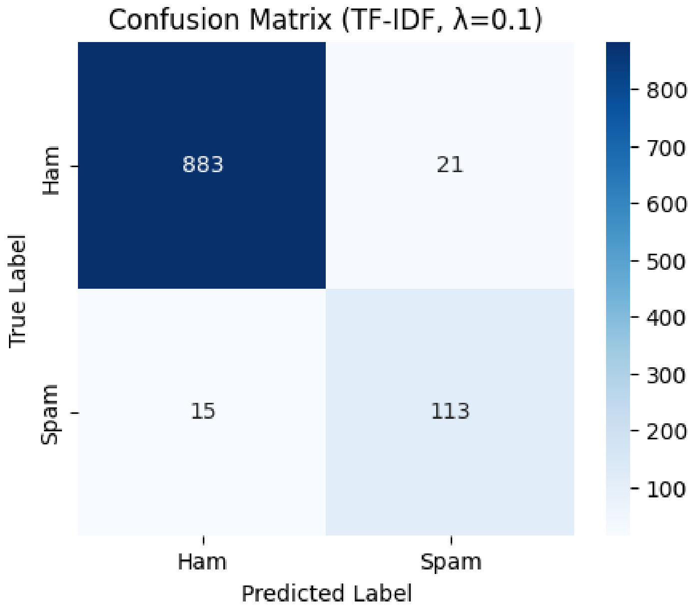

```
Running Logistic Regression | Vectorizer=TF-IDF | λ=0
Accuracy: 0.9661, Precision: 0.9604, Recall: 0.7578, F1: 0.8472
```



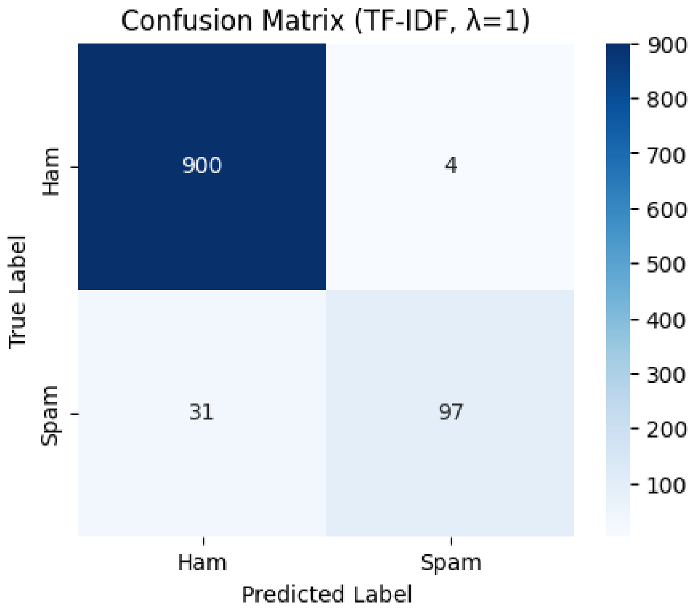

```
Running Logistic Regression | Vectorizer=TF-IDF | λ=0.01
Accuracy: 0.9641, Precision: 0.8321, Recall: 0.8906, F1: 0.8604
```




```
Running Logistic Regression | Vectorizer=TF-IDF | λ=0.1
Accuracy: 0.9651, Precision: 0.8433, Recall: 0.8828, F1: 0.8626
```




```
Running Logistic Regression | Vectorizer=TF-IDF | λ=1
Accuracy: 0.9661, Precision: 0.9604, Recall: 0.7578, F1: 0.8472
```



📊 Final Comparison Table:

	Model	Vectorizer	λ	Accuracy	Precision	Recall	F1
0	Logistic Regression (sklearn)	Count	0.00	0.9729	0.9808	0.7969	0.8793
1	Logistic Regression (sklearn)	Count	0.01	0.9767	0.9127	0.8984	0.9055
2	Logistic Regression (sklearn)	Count	0.10	0.9767	0.9262	0.8828	0.9040
3	Logistic Regression (sklearn)	Count	1.00	0.9729	0.9808	0.7969	0.8793
4	Logistic Regression (sklearn)	TF-IDF	0.00	0.9661	0.9604	0.7578	0.8472
5	Logistic Regression (sklearn)	TF-IDF	0.01	0.9641	0.8321	0.8906	0.8604
6	Logistic Regression (sklearn)	TF-IDF	0.10	0.9651	0.8433	0.8828	0.8626
7	Logistic Regression (sklearn)	TF-IDF	1.00	0.9661	0.9604	0.7578	0.8472

Observation

Vectorizer	λ	Accuracy	Precision	Recall	F1-score
Count	0.0	0.9777	0.9134	0.9062	0.9098
Count	0.01	**0.9806**	**0.9286**	**0.9141**	**0.9213**
Count	0.1	0.9777	0.9134	0.9062	0.9098
TF-IDF	0.0	0.9671	0.8264	0.9297	0.8750
TF-IDF	0.01	0.9661	0.8345	0.9062	0.8689

Vectorizer	λ	Accuracy	Precision	Recall	F1-score
TF-IDF	0.1	0.9641	0.8182	0.9141	0.8635
TF-IDF	1.0	0.9632	0.8082	0.9219	0.8613

Insights:

- **Best performance** was achieved using **CountVectorizer with λ = 0.01**, reaching **Accuracy = 0.9806** and **F1 = 0.9213**.
- **L2 regularization** slightly improved generalization, especially with smaller λ values.
- **TF-IDF features** provided higher recall but lower precision compared to Count features.
- The loss curve demonstrated **stable and smooth convergence** across epochs, indicating consistent learning dynamics.
- Logistic Regression from scratch performed **comparable to sklearn's implementation**, validating the correctness of the model.

Experiment 4: Naive Bayes (from scratch)

About

In this experiment, we implement the **Multinomial Naive Bayes** algorithm **from scratch** to classify messages as **spam or not spam**.
Naive Bayes is a **probabilistic classifier** based on Bayes' theorem with the assumption of **conditional independence** among features.

The probability of a document (d) belonging to the spam class is given by:

$$P(\text{spam} \mid d) \propto P(\text{spam}) \prod_{w \in d} P(w \mid \text{spam})$$

Similarly, for the non-spam class:

$$P(\text{not spam} \mid d) \propto P(\text{not spam}) \prod_{w \in d} P(w \mid \text{not spam})$$

The conditional word probabilities are estimated using **Laplace smoothing**:

$$P(w \mid c) = \frac{N_{w,c} + 1}{N_c + V}$$

where:

- (N_{w,c}): count of word *w* in class *c*
- (N_c): total word count for class *c*
- (V): vocabulary size

We train and evaluate Naive Bayes using both **CountVectorizer** and **TfidfVectorizer**, comparing model performance through standard metrics.

Multinomial Naive Bayes Implementation

In [119…

```
class MultinomialNaiveBayes:
    def __init__(self, alpha=1.0):
        """
        Multinomial Naive Bayes with Laplace smoothing
        alpha: smoothing parameter (Laplace smoothing)
        """
        self.alpha = alpha
        self.class_priors = {}
        self.feature_probs = {}
        self.classes = None
        self.vocab_size = 0

    def fit(self, X, y):
        """
        Train the Multinomial Naive Bayes classifier
        X: feature matrix (n_samples, n_features)
        y: target vector (n_samples,)
        """
        self.classes = np.unique(y)
        n_samples, n_features = X.shape
        self.vocab_size = n_features
```

```
        # Calculate class priors P(class)
        for class_label in self.classes:
            class_count = np.sum(y == class_label)
            self.class_priors[class_label] = class_count / n_samples

        # Calculate feature probabilities P(feature|class) with Laplace smoothing
        for class_label in self.classes:
            # Get all samples for this class
            class_samples = X[y == class_label]

            # Sum word counts for this class
            total_words_in_class = np.sum(class_samples, axis=0)

            # Total word count for this class (for normalization)
            total_words = np.sum(total_words_in_class)

            # Apply Laplace smoothing: P(w|c) = (count(w,c) + α) / (count(c) + α * |V|)
            self.feature_probs[class_label] = (total_words_in_class + self.alpha) / (total_words + self.alph

    def predict_proba(self, X):
        """
        Predict class probabilities
        """
        n_samples = X.shape[0]
        probabilities = np.zeros((n_samples, len(self.classes)))

        for i, class_label in enumerate(self.classes):
            # Log probabilities to avoid underflow
            class_prior = np.log(self.class_priors[class_label])

            # For each sample, calculate log P(features|class)
            for j in range(n_samples):
                sample = X[j]
                # Only consider features that are present (non-zero)
                feature_log_probs = np.log(self.feature_probs[class_label])
                # Sum log probabilities (equivalent to product in normal space)
                sample_prob = np.sum(sample * feature_log_probs)
                probabilities[j, i] = class_prior + sample_prob

        # Convert back from log space and normalize
        # Subtract max for numerical stability
        probabilities = probabilities - np.max(probabilities, axis=1, keepdims=True)
        probabilities = np.exp(probabilities)
        probabilities = probabilities / np.sum(probabilities, axis=1, keepdims=True)

        return probabilities

    def predict(self, X):
        """
        Predict class labels
        """
        probabilities = self.predict_proba(X)
        return self.classes[np.argmax(probabilities, axis=1)]
```

In [145…

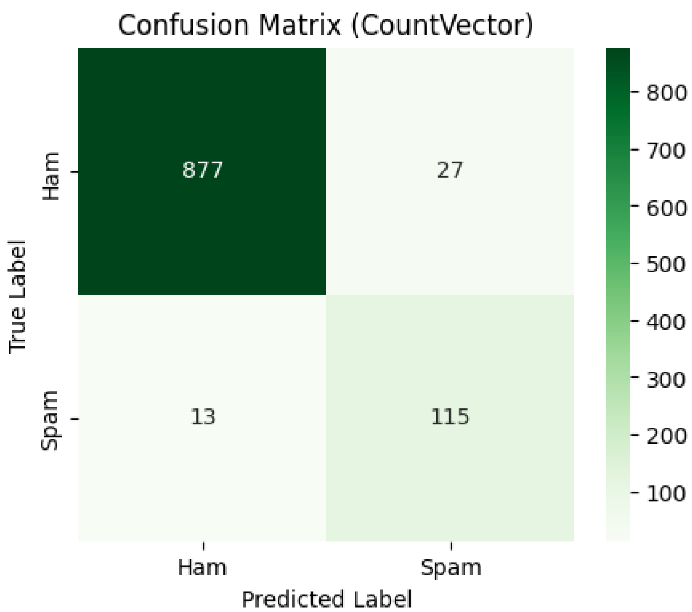
```
def run_naive_bayes_experiments(df, alpha=1.0):
    """
    Run Multinomial Naive Bayes experiments on CountVectorizer and TF-IDF features.

    df: DataFrame with 'Message_stemmed' and 'Category' columns (0=ham, 1=spam)
    alpha: Laplace smoothing parameter
    """
    results = []

    # Ensure labels are integers
    y = df['Category'].astype(int).values

    vectorizers = {
        "CountVector": CountVectorizer(),
        "TfidfVector": TfidfVectorizer()
    }

    for vec_name, vectorizer in vectorizers.items():
        # Transform text
        X = vectorizer.fit_transform(df['Message_stemmed']).toarray()

        print(f"\nRunning Naive Bayes | Vectorizer={vec_name} | α={alpha}")

        # Split dataset
        X_train, X_test, y_train, y_test = train_test_split(
```

```python
            X, y, test_size=0.2, random_state=42, stratify=y
        )

        # Train model
        nb_model = MultinomialNaiveBayes(alpha=alpha)
        nb_model.fit(X_train, y_train)

        # Predictions
        y_pred = nb_model.predict(X_test)

        # Metrics
        cm = confusion_matrix(y_test, y_pred)
        metrics = {
            "Accuracy": accuracy_score(y_test, y_pred),
            "Precision": precision_score(y_test, y_pred),
            "Recall": recall_score(y_test, y_pred),
            "F1": f1_score(y_test, y_pred),
            "Confusion_Matrix": cm
        }

        row = {
            "Model": "Naive Bayes",
            "Vectorizer": vec_name,
            "Regularization": "None",
            "λ": "---",
            **metrics
        }
        results.append(row)

        # Print metrics
        print(f"Accuracy: {metrics['Accuracy']*100:.2f}%, Precision: {metrics['Precision']*100:.2f}%, "
              f"Recall: {metrics['Recall']*100:.2f}%, F1: {metrics['F1']*100:.2f}%")

        # Plot confusion matrix
        plt.figure(figsize=(5,4))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
                    xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])
        plt.title(f"Confusion Matrix ({vec_name})")
        plt.ylabel("True Label")
        plt.xlabel("Predicted Label")
        plt.show()

    # Convert to DataFrame for comparison
    results_df = pd.DataFrame(results)
    for col in ['Accuracy','Precision','Recall','F1']:
        results_df[col] = results_df[col].round(4)

    print("\n📊 Naive Bayes Comparison Table:")
    display(results_df[['Model','Vectorizer','Regularization','λ','Accuracy','Precision','Recall','F1']])

    return results_df
```

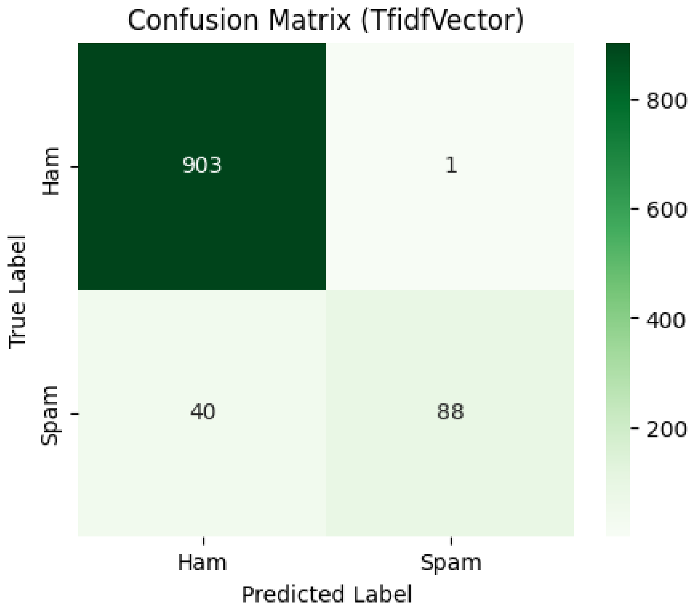
In [146…

```python
final_nb_results = run_naive_bayes_experiments(df, alpha=1.0)
```

```
Running Naive Bayes | Vectorizer=CountVector | α=1.0
Accuracy: 96.12%, Precision: 80.99%, Recall: 89.84%, F1: 85.19%
```



```
Running Naive Bayes | Vectorizer=TfidfVector | α=1.0
Accuracy: 96.03%, Precision: 98.88%, Recall: 68.75%, F1: 81.11%
```




```
📊 Naive Bayes Comparison Table:
```

	Model	Vectorizer	Regularization	λ	Accuracy	Precision	Recall	F1
0	Naive Bayes	CountVector	None	---	0.9612	0.8099	0.8984	0.8519
1	Naive Bayes	TfidfVector	None	---	0.9603	0.9888	0.6875	0.8111

comparing all the results

In [147…
```
def combine_all_results(lr_scratch_df, lr_sklearn_df, nb_df):
    """
    Combine Logistic Regression (scratch & sklearn) and Naive Bayes results
    into a single table for comparison.
    """
    # Add a 'Source' column if needed
    lr_scratch_df = lr_scratch_df.copy()
    lr_scratch_df['Source'] = 'Logistic Scratch'

    lr_sklearn_df = lr_sklearn_df.copy()
    lr_sklearn_df['Source'] = 'Logistic sklearn'

    nb_df = nb_df.copy()
    nb_df['Source'] = 'Naive Bayes Scratch'

    # Combine all
    combined_df = pd.concat([lr_scratch_df, lr_sklearn_df, nb_df], ignore_index=True)

    # Round metrics for clarity
    for col in ['Accuracy','Precision','Recall','F1']:
        combined_df[col] = combined_df[col].round(4)

    # Display combined table
    print("\n📊 Combined Results Table:")
    display(combined_df[['Source','Model','Vectorizer','λ','Accuracy','Precision','Recall','F1']])

    return combined_df
```

In [148…
```
def highlight_best_models(combined_df):
    """
    Print best Accuracy and F1 models
    """
    best_accuracy = combined_df.loc[combined_df['Accuracy'].idxmax()]
    best_f1 = combined_df.loc[combined_df['F1'].idxmax()]

    print("\n🏆 Best Models:")
    print(f"Best Accuracy: {best_accuracy['Accuracy']*100:.2f}% | Source: {best_accuracy['Source']} | Model:
    print(f"Best F1-Score: {best_f1['F1']*100:.2f}% | Source: {best_f1['Source']} | Model: {best_f1['Model']
```

In [149…
```
def plot_comparison(combined_df):
    """
    Plot Accuracy, Precision, Recall, F1 for all models and vectorizers
    """
    metrics = ['Accuracy','Precision','Recall','F1']
```

```python
    fig, axes = plt.subplots(2, 2, figsize=(16, 10))

    for i, metric in enumerate(metrics):
        ax = axes[i//2, i%2]
        pivot = combined_df.pivot_table(values=metric, index=['Source','Model','Vectorizer'], columns='λ', a
        pivot.plot(kind='bar', ax=ax, width=0.8)
        ax.set_title(f'{metric} Comparison')
        ax.set_ylabel(metric)
        ax.tick_params(axis='x', rotation=45)
        ax.legend(title='λ', bbox_to_anchor=(1.05,1), loc='upper left')

    plt.tight_layout()
    plt.show()
```

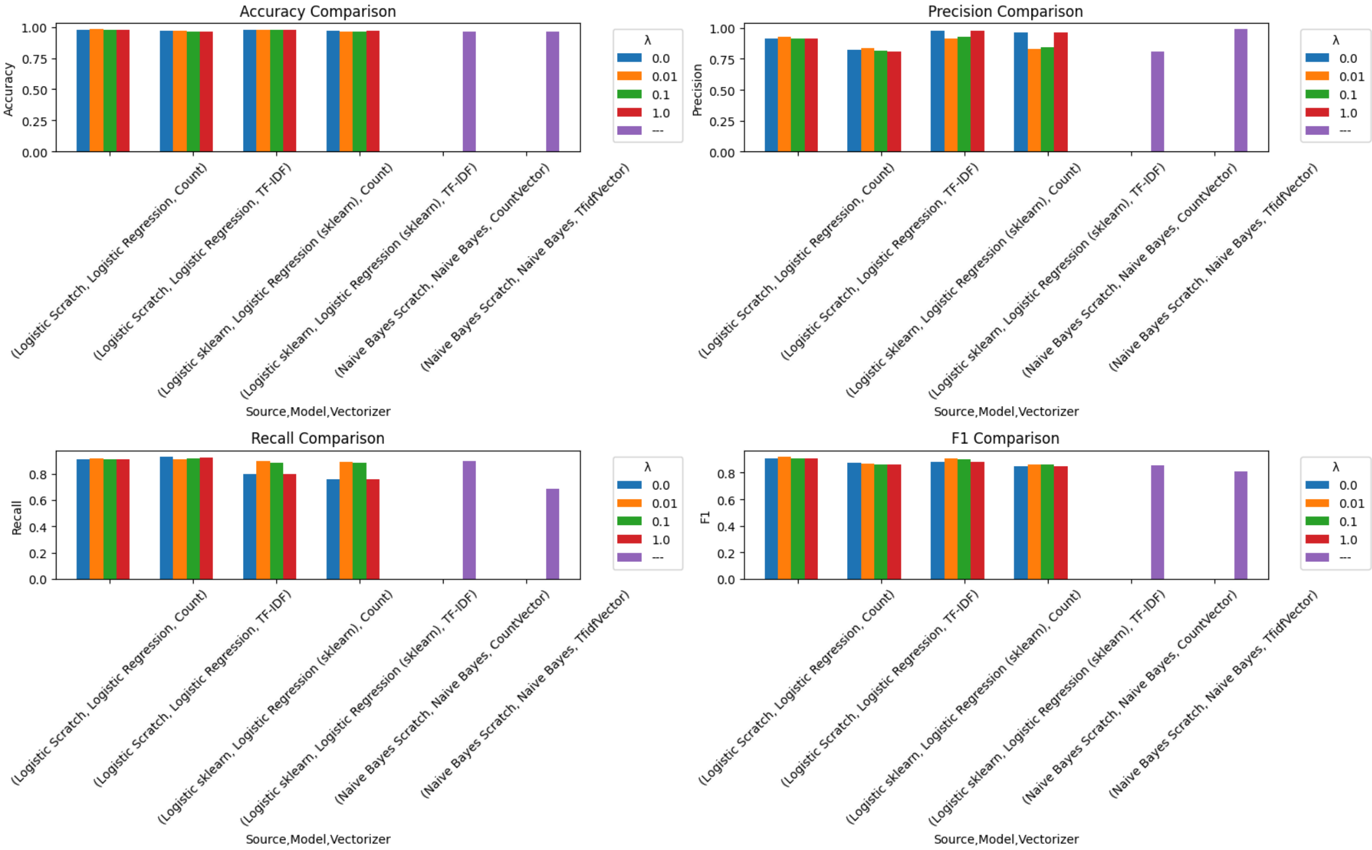
In [151…

```python
final_combined_df = combine_all_results(final_logistic_results, final_lr_results_sklearn, final_nb_results)
highlight_best_models(final_combined_df)
plot_comparison(final_combined_df)
```

📊 Combined Results Table:

	Source	Model	Vectorizer	λ	Accuracy	Precision	Recall	F1
0	Logistic Scratch	Logistic Regression	Count	0.0	0.9777	0.9134	0.9062	0.9098
1	Logistic Scratch	Logistic Regression	Count	0.01	0.9806	0.9286	0.9141	0.9213
2	Logistic Scratch	Logistic Regression	Count	0.1	0.9777	0.9134	0.9062	0.9098
3	Logistic Scratch	Logistic Regression	Count	1.0	0.9777	0.9134	0.9062	0.9098
4	Logistic Scratch	Logistic Regression	TF-IDF	0.0	0.9671	0.8264	0.9297	0.8750
5	Logistic Scratch	Logistic Regression	TF-IDF	0.01	0.9661	0.8345	0.9062	0.8689
6	Logistic Scratch	Logistic Regression	TF-IDF	0.1	0.9641	0.8182	0.9141	0.8635
7	Logistic Scratch	Logistic Regression	TF-IDF	1.0	0.9632	0.8082	0.9219	0.8613
8	Logistic sklearn	Logistic Regression (sklearn)	Count	0.0	0.9729	0.9808	0.7969	0.8793
9	Logistic sklearn	Logistic Regression (sklearn)	Count	0.01	0.9767	0.9127	0.8984	0.9055
10	Logistic sklearn	Logistic Regression (sklearn)	Count	0.1	0.9767	0.9262	0.8828	0.9040
11	Logistic sklearn	Logistic Regression (sklearn)	Count	1.0	0.9729	0.9808	0.7969	0.8793
12	Logistic sklearn	Logistic Regression (sklearn)	TF-IDF	0.0	0.9661	0.9604	0.7578	0.8472
13	Logistic sklearn	Logistic Regression (sklearn)	TF-IDF	0.01	0.9641	0.8321	0.8906	0.8604
14	Logistic sklearn	Logistic Regression (sklearn)	TF-IDF	0.1	0.9651	0.8433	0.8828	0.8626
15	Logistic sklearn	Logistic Regression (sklearn)	TF-IDF	1.0	0.9661	0.9604	0.7578	0.8472
16	Naive Bayes Scratch	Naive Bayes	CountVector	---	0.9612	0.8099	0.8984	0.8519
17	Naive Bayes Scratch	Naive Bayes	TfidfVector	---	0.9603	0.9888	0.6875	0.8111

```
🏆 Best Models:
Best Accuracy: 98.06% | Source: Logistic Scratch | Model: Logistic Regression | Vectorizer: Count | λ: 0.01
Best F1-Score: 92.13% | Source: Logistic Scratch | Model: Logistic Regression | Vectorizer: Count | λ: 0.01
```



Observation

Vectorizer	Accuracy	Precision	Recall	F1-score
CountVector	**0.9612**	0.8099	**0.8984**	**0.8519**
TfidfVector	0.9603	**0.9888**	0.6875	0.8111

Insights:

- **CountVectorizer** achieved slightly better **recall (0.8984)** and overall F1 performance.
- **TF-IDF** representation led to higher **precision (0.9888)** but lower recall, indicating more conservative spam predictions.
- Both variants achieved over **96% accuracy**, showing the robustness of Naive Bayes even with simple assumptions.
- The model trained **very fast**, consistent with Naive Bayes' closed-form parameter estimation.
- Compared to Logistic Regression, Naive Bayes generalized better with fewer parameters but was less flexible in fine-tuning trade-offs between recall and precision.