LAB ON NAÏVE BAYES

Lab – Naïve Bayes

Dataset: SMS Spam Collection - https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Task 1: Load and Explore Data (4 min)

- Load spam.csv using pandas
- Check dataset shape and column names
- Count spam vs ham messages
- Display 3 spam and 3 ham examples

Question: What percentage of messages are spam?

Task 2: Understand Bayes Theorem (5 min)

- Calculate prior probabilities: P(spam) and P(ham)
- Split data 80/20 for training/testing
- Manually count how often word "free" appears in spam vs ham messages
- Calculate P("free"|spam) and P("free"|ham)

Question: Based on these probabilities, if you see "free" in a message, is it more likely spam or ham?

Task 3: Build Naive Bayes Classifier (6 min)

- Use CountVectorizer to convert text to features
- Train MultinomialNB on training data
- Predict on test set and calculate accuracy
- Create confusion matrix

Lab Continued – Naïve Bayes

Questions:

- What is your model's accuracy?
- Does it perform better at detecting spam or ham?

Task 4: Analyze Feature Importance (3 min)

- Extract feature names from CountVectorizer
- Find top 5 words with highest probability ratios for spam class
- Test classifier on custom message: "Free call now! Win money!"

Questions:

- Which words are strongest spam indicators?
- Does your custom message get classified correctly?

Task 5: Test the "Naive" Assumption (2 min)

- Create two similar messages: "Call me" vs "Free call"
- Get probability scores for both messages
- Discuss why Naive Bayes assumes word independence

Question: Even though words aren't truly independent, why does Naive Bayes still work well for spam detection?

[4]:
```python
# Load dataset
import pandas as pd

# Load the dataset
df = pd.read_csv("spam.csv", encoding='latin-1')[['v1', 'v2']]
df.columns = ['label', 'message']
# Dataset shape and column names
print("Shape:", df.shape)
print("Columns:", df.columns)

# Count spam vs ham
print(df['label'].value_counts())

# Display 3 examples of each
print("\nSpam examples:")
print(df[df['label'] == 'spam']['message'].head(3))

print("\nHam examples:")
print(df[df['label'] == 'ham']['message'].head(3))

# Percentage of spam
spam_pct = (df['label'] == 'spam').mean() * 100
print(f"\nPercentage of messages that are spam: {spam_pct:.2f}%")
```

```
Shape: (5572, 2)
Columns: Index(['label', 'message'], dtype='object')
label
ham     4825
spam     747
Name: count, dtype: int64

Spam examples:
2    Free entry in 2 a wkly comp to win FA Cup fina...
5    FreeMsg Hey there darling it's been 3 week's n...
8    WINNER!! As a valued network customer you have...
Name: message, dtype: object

Ham examples:
0    Go until jurong point, crazy.. Available only ...
1                        Ok lar... Joking wif u oni...
3    U dun say so early hor... U c already then say...
Name: message, dtype: object

Percentage of messages that are spam: 13.41%
```

Percentage of messages that are spam: 13.40%

[5]:
```python
# Prior probabilities
P_spam = (df['label'] == 'spam').mean()
P_ham = (df['label'] == 'ham').mean()

print(f"P(spam): {P_spam:.2f}")
print(f"P(ham): {P_ham:.2f}")
# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['message'], df['label'], test_size=0.2, random_state=42)

# Count "free" in spam vs ham
spam_msgs = df[df['label'] == 'spam']['message']
ham_msgs = df[df['label'] == 'ham']['message']

free_in_spam = spam_msgs.str.contains("free", case=False).sum()
free_in_ham = ham_msgs.str.contains("free", case=False).sum()

P_free_given_spam = free_in_spam / len(spam_msgs)
P_free_given_ham = free_in_ham / len(ham_msgs)

print(f"P('free'|spam): {P_free_given_spam:.2f}")
print(f"P('free'|ham): {P_free_given_ham:.2f}")
```

```
P(spam): 0.13
P(ham): 0.87
P('free'|spam): 0.27
P('free'|ham): 0.01
```

Conclusion: The word “free” appears in 22% of spam messages but only 1% of ham messages. So if a message contains “free,” it’s much more likely to be spam.

[6]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix

# Vectorize text
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train model
model = MultinomialNB()
model.fit(X_train_vec, y_train)

# Predict and evaluate
y_pred = model.predict(X_test_vec)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")
print("Confusion Matrix:\n", conf_matrix)
```

```
Model Accuracy: 0.98
Confusion Matrix:
 [[963   2]
 [ 16 134]]
```

Interpretation:

- True ham: 965
- False spam (ham predicted as spam): 0
- False ham (spam predicted as ham): 21
- True spam: 129

The model performs very well, especially at identifying ham messages.

[7]:
```
import numpy as np

# Get feature names and log probabilities
feature_names = vectorizer.get_feature_names_out()
spam_probs = model.feature_log_prob_[1]
ham_probs = model.feature_log_prob_[0]

# Calculate spam-to-ham ratio
ratios = np.exp(spam_probs - ham_probs)
top_spam_indices = np.argsort(ratios)[-5:]
top_spam_words = feature_names[top_spam_indices]

print("Top 5 spam indicator words:", top_spam_words)

# Test custom message
custom_msg = ["Freecall now! Win money!"]
custom_vec = vectorizer.transform(custom_msg)
custom_pred = model.predict(custom_vec)
print("Custom message prediction:", custom_pred[0])
```

```
Top 5 spam indicator words: ['tone' '150p' 'uk' 'prize' 'claim']
Custom message prediction: spam
```

Interpretation:

- Words like “free,” “claim,” “won,” and “mobile” are strong spam indicators.
- The custom message “Freecall now! Win money!” is correctly classified as spam.

[8]:
```
# Compare two similar messages
msgs = ["Call me", "Free call"]
vecs = vectorizer.transform(msgs)
probs = model.predict_proba(vecs)

for msg, prob in zip(msgs, probs):
    print(f"Message: '{msg}' → Spam probability: {prob[1]:.2f}")
```

```
Message: 'Call me' → Spam probability: 0.05
Message: 'Free call' → Spam probability: 0.85
```

[ ]:

Discussion:

- Naive Bayes assumes word independence, meaning it treats “free” and “call” as separate signals.

- Even though words in natural language are dependent, Naive Bayes still works well because strong individual words (like “free”) dominate the prediction.