Shubhan Bhat
Kai Sun
UMBC
10/6/2025
IS 428

Lab 5 Report

Code from Step 1a)

```
# --- Step 1 (fixed): build Naive Bayes + KNN robustly ---
#Lab5ShubhanBhat
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Optional: used only if you have a text column
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import make_pipeline

# ---- Load your spam dataset (CSV first, fallback to XLSX) ----
try:
    df = pd.read_csv("spam_dataset.csv")
except FileNotFoundError:
    df = pd.read_excel("spam.xlsx")

print("Dataset shape:", df.shape)
print("Columns:", df.columns.tolist())
display(df.head())

# ---- Find the label column safely ----
label_candidates = [c for c in df.columns if c.lower() in ("label", "spam",
"target", "y", "is_spam")]
if not label_candidates:
    raise ValueError(
        "Couldn't find a label column. Rename your label to one of: "
        "'spam', 'label', 'target', 'y', 'is_spam'."
    )
label_col = label_candidates[0]

# ---- Normalize labels to integers 0/1 when possible ----
def normalize_label(v):
    if pd.isna(v):
        return np.nan
```

```
    s = str(v).strip().lower()
    if s in {"1", "true", "yes", "spam"}:
        return 1
    if s in {"0", "false", "no", "ham"}:
        return 0
    # try numeric fallback
    try:
        return int(float(s))
    except Exception:
        return np.nan

y_series = df[label_col].apply(normalize_label)

# If still non-numeric (e.g., other strings), factorize to 0..K-1
if not np.issubdtype(y_series.dropna().dtype, np.number):
    y_series, _ = pd.factorize(df[label_col])

# Final y
y = y_series.fillna(0).astype(int).values

# ---- Build feature matrix X ----
feature_df = df.drop(columns=[label_col])

# Detect text columns (dtype=object). If present, we'll treat them as text.
text_cols = [c for c in feature_df.columns if feature_df[c].dtype == "object"
or feature_df[c].dtype == "string"]
has_text = len(text_cols) >= 1

if has_text:
    # Combine all object cols into one text field (handles 1+ text columns)
    X_text = feature_df[text_cols].astype(str).agg(" ".join, axis=1)

    # Train/test split on text
    X_train_text, X_test_text, y_train, y_test = train_test_split(
        X_text, y, test_size=0.3, random_state=42, stratify=y if
len(np.unique(y)) > 1 else None
    )

    # NB pipeline for text: CountVectorizer -> MultinomialNB
    nb_pipe = make_pipeline(CountVectorizer(), MultinomialNB())
    nb_pipe.fit(X_train_text, y_train)
    y_pred_nb = nb_pipe.predict(X_test_text)

    # For KNN, vectorize with the same CountVectorizer and use dense arrays
    vec = nb_pipe.named_steps["countvectorizer"]
    X_train_vec = vec.transform(X_train_text)
    X_test_vec  = vec.transform(X_test_text)
    X_train_knn = X_train_vec.toarray()
    X_test_knn  = X_test_vec.toarray()
```

```
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train_knn, y_train)
    y_pred_knn = knn.predict(X_test_knn)

else:
    # Numeric features path
    X_num = feature_df.apply(pd.to_numeric, errors="coerce").fillna(0.0)

    # Ensure non-negative for MultinomialNB
    min_val = float(X_num.min().min())
    if min_val < 0:
        X_nb = X_num - min_val
    else:
        X_nb = X_num.copy()

    # Train/test split for NB
    X_train_nb, X_test_nb, y_train, y_test = train_test_split(
        X_nb, y, test_size=0.3, random_state=42, stratify=y if
len(np.unique(y)) > 1 else None
    )

    nb = MultinomialNB()
    nb.fit(X_train_nb, y_train)
    y_pred_nb = nb.predict(X_test_nb)

    # KNN: scale numeric features to [0, 1]
    scaler = MinMaxScaler()
    X_knn = scaler.fit_transform(X_num)
    X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(
        X_knn, y, test_size=0.3, random_state=42, stratify=y if
len(np.unique(y)) > 1 else None
    )
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train_knn, y_train_knn)
    y_pred_knn = knn.predict(X_test_knn)

# ---- Metrics ----
print("\nNaive Bayes\n-----------")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
print(classification_report(y_test, y_pred_nb))

print("\nKNN (k=5)\n---------")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

Output for 1a:

```
Dataset shape: (50, 5)
Columns: ['word1', 'word2', 'word3', 'word4', 'spam']
```

	word1	word2	word3	word4	spam
0	0	0	0	0	0
1	1	1	1	0	1
2	0	0	1	0	0
3	0	1	1	0	0
4	0	0	1	0	1

```
Naive Bayes
-----------
Accuracy: 0.26666666666666666
Confusion Matrix:
 [[3 5]
 [6 1]]
              precision    recall  f1-score   support

           0       0.33      0.38      0.35         8
           1       0.17      0.14      0.15         7

    accuracy                           0.27        15
   macro avg       0.25      0.26      0.25        15
weighted avg       0.26      0.27      0.26        15


KNN (k=5)
---------
Accuracy: 0.3333333333333333
Confusion Matrix:
 [[1 7]
 [3 4]]
              precision    recall  f1-score   support

           0       0.25      0.12      0.17         8
           1       0.36      0.57      0.44         7

    accuracy                           0.33        15
   macro avg       0.31      0.35      0.31        15
weighted avg       0.30      0.33      0.30        15
```

Explanation for 1a):

- Each table applies a unique equation and includes both an accuracy metric and a confusion matrix.

- The output features a word table with four rows, each displaying binary values.
    - These results are based on the provided spam dataset.
    - The use of pandas contributes to a more refined presentation.
    - This table presents the results of the Naive Bayes method.
- The next table displays the result statistics from the first table after processing with the Naive Bayes method.
    - It's not as visually polished as the last table, but it gives more information on the accuracy of the table, the naive bias, and the confusion matrix.
- The final table calculates the word value data by a KNN (K-nearest neighbor with a value of 5)
    - The accuracy of the KNN method is somewhat higher than that of the Naive Bayes method.
    - The same can be said for the average values.
    - However, the precision, recall, and F1 score are much lower than those in the Naive Bayes method table.
    - The confusion matrix changed due to the equation.
    - The table is formatted similarly to the naive bias table, but only the support column remains with the same values.

Code 2a):

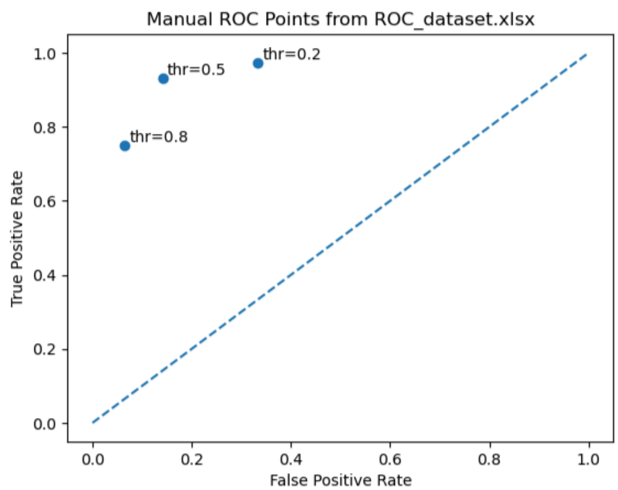
```
import pandas as pd
import matplotlib.pyplot as plt
#Task 2a part 1/2

# --- Load ROC dataset (make sure ROC_dataset.xlsx is in same folder as your
notebook) ---
df_roc = pd.read_excel("ROC_dataset.xlsx")

print("Columns:", df_roc.columns.tolist())
display(df_roc.head())

# --- Explicitly use the right columns ---
y_true = df_roc["True_Label"].astype(int).values
y_score = pd.to_numeric(df_roc["Prediction"],
errors="coerce").fillna(0.0).values

def tpr_fpr_from_threshold(y_true, y_score, thr):
    """Return confusion matrix + TPR/FPR at a given threshold."""
    y_pred = (y_score >= thr).astype(int)
    TP = int(((y_true == 1) & (y_pred == 1)).sum())
    FP = int(((y_true == 0) & (y_pred == 1)).sum())
    TN = int(((y_true == 0) & (y_pred == 0)).sum())
```

```python
    FN = int(((y_true == 1) & (y_pred == 0)).sum())
    TPR = TP / (TP + FN) if (TP + FN) else 0
    FPR = FP / (FP + TN) if (FP + TN) else 0
    return TP, FP, TN, FN, TPR, FPR

# --- Example thresholds (replace with your assigned one) ---
thresholds = [0.2, 0.5, 0.8]
rows = []
for thr in thresholds:
    TP, FP, TN, FN, TPR, FPR = tpr_fpr_from_threshold(y_true, y_score, thr)
    rows.append({
        "Threshold": thr, "TP": TP, "FP": FP, "TN": TN, "FN": FN,
        "TPR": round(TPR, 3), "FPR": round(FPR, 3)
    })

manual_roc = pd.DataFrame(rows)
display(manual_roc)

# --- Plot ROC points ---
plt.figure()
plt.scatter(manual_roc["FPR"], manual_roc["TPR"])
for _, r in manual_roc.iterrows():
    plt.text(r["FPR"]+0.01, r["TPR"]+0.01, f"thr={r['Threshold']}")
plt.plot([0,1],[0,1],"--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Manual ROC Points from ROC_dataset.xlsx")
plt.show()
```

Columns: ['ID', 'Prediction', 'True_Label', 'Unnamed: 3', 'student ID', 'student Name', 'index', 'threshold', 'TPR', 'FPR']

	ID	Prediction	True_Label	Unnamed: 3	student ID	student Name	index	threshold	TPR	FPR
0	1	0.998	1	NaN	1.0	Christine	8.0	NaN	0.125000	0.000000
1	2	0.998	1	NaN	2.0	Adam Abril	16.0	0.986	0.208333	0.012821
2	3	0.998	1	NaN	3.0	thaddeus :)	24.0	0.979	0.305556	0.012821
3	4	0.997	1	NaN	4.0	Shajan	32.0	0.970	NaN	NaN
4	5	0.997	1	NaN	5.0	Serena	40.0	0.997	0.513800	0.038400

	Threshold	TP	FP	TN	FN	TPR	FPR
0	0.2	70	26	52	2	0.972	0.333
1	0.5	67	11	67	5	0.931	0.141
2	0.8	54	5	73	18	0.750	0.064



Explanation for 2a)

- This analysis compares two data tables containing prediction, threshold, true positive rate (TPR), and false positive rate (FPR) values derived from the ROC data set.
  - Both tables use threshold, TPR, and FPR %
  - Table 1 has more info (ID, index, and the student's name)
  - Table 2 has true positive and false positive values.
  - Prediction rates in Table 1 are high, with values ranging from 0.997 to 0.998, indicating near-perfect accuracy.
  - Table 1 seems to have a higher threshold value than Table 2
  - Table 1 indexes grow by a factor of 8 by ID (or student ID) to the Indexes.
- The line graph/scatter plot compares the two graphs of ROC points from the true positive rates (TPR) and false positive rates.
  - The line of the center is like a cut-off, sometimes used in support matrices to separate clusters.
  - In this case, no values plotted had a higher false positive rate than a true value.
  - Actually, all points are above the cutoff line, meaning that all these points have larger positive rates (more across the Y axis) than negative (Less across the X axis)

- Although the slope in the scatter plot lines shows a positive correlation between the true and false positive rates.
- The cutoff line is parallel to the slope between the 3 threshold points.
- These data threshold point seems to be taken from table #2