Question 1

(a)

(1) For the following question: TP (True Positive), TN (True Negative), FP (False Positive) and FN (False Negative).

Precision measures the exactness of the model. It is the ratio of correctly predicted positive observations to the total predicted positives:

$$\text{Precision} = \frac{TP}{TP + FP}.$$

In the context of binary classification, precision is how reliable the model is when it predicts the positive class.

Recall measures the completeness of the model. It is the ratio of correctly predicted positive observations to all actual positives:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

Recall is the model's ability to find all relevant cases within the dataset. A high recall indicates that the model has a low FN rate.

(2) Accuracy can be misleading because it only measures the overall fraction of correct predictions without accounting for the distribution of classes or the specific types of errors made. Precision and recall are essential complementary metrics because they reveal the specific trade-offs of the model. For example, if 98% of a dataset belongs to the negative class, a model that simply predicts "negative" for every single instance will achieve 98% accuracy despite failing to identify any positive cases (0% recall).

(b)

[50]:
```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import MinMaxScaler

def encode_binary_columns(df):
    df_encoded = df.copy()

    obj_cols = df_encoded.select_dtypes(include=['object', 'category',␣
 ↪'string']).columns

    for col in obj_cols:
        unique_vals = df_encoded[col].dropna().unique()
        if len(unique_vals) <= 2:
            unique_vals = sorted(unique_vals)
            mapping = {val: i for i, val in enumerate(unique_vals)}
            df_encoded[col] = df_encoded[col].map(mapping)
    return df_encoded
```

```
df = pd.read_csv('Q1_bioprinting_data.csv')

# I followed the threshold mentioned in the paper to determine viability.
VIABILITY_THRESHOLD = 80
y = (df['Viability_at_time_of_observation_(%)'] >= VIABILITY_THRESHOLD).
 ↪astype(int)

cols_to_drop = [
    'Viability_at_time_of_observation_(%)',
    'Reference',
    'DOI',
    'Acceptable_Viability_(Yes/No)',
    'Acceptable_Pressure_(Yes/No)'
]
df = df.drop(columns=cols_to_drop, errors='ignore')

df['Syringe_Temperature_(°C)'] = df['Syringe_Temperature_(°C)'].fillna(22)
df['Substrate_Temperature_(°C)'] = df['Substrate_Temperature_(°C)'].fillna(22)

cols_only_null_zero = [
    col for col in df.columns
    if ((df[col].isna()) | (df[col] == 0)).all()
]
df = df.drop(columns=cols_only_null_zero)

df = df.loc[:, df.isna().mean() <= 0.5]

df = encode_binary_columns(df)

df_numeric = df.select_dtypes(include=['number'])
imputer = KNNImputer(n_neighbors=30)
df_imputed_data = imputer.fit_transform(df_numeric)
df_imputed = pd.DataFrame(df_imputed_data, columns=df_numeric.columns)

scaler = MinMaxScaler()
df_scaled_array = scaler.fit_transform(df_imputed)
X = pd.DataFrame(df_scaled_array, columns=df_numeric.columns)
```

(c)

[51]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=67)
```

```
dt_clf = DecisionTreeClassifier(random_state=67)

dt_clf.fit(X_train, y_train)

y_pred_dt = dt_clf.predict(X_test)

dt_acc = accuracy_score(y_test, y_pred_dt)
dt_prec = precision_score(y_test, y_pred_dt)
dt_rec = recall_score(y_test, y_pred_dt)

print("Decision Tree")
print(f"Accuracy: {dt_acc:.4f}")
print(f"Precision: {dt_prec:.4f}")
print(f"Recall: {dt_rec:.4f}")
```

```
Decision Tree
Accuracy: 0.7339
Precision: 0.7792
Recall: 0.7895
```

Accuracy (0.7339): the model correctly classifies approximately 73.4% of the bio-printing outcomes. This indicates a moderate ability to distinguish between acceptable and unacceptable viability based on the parameters.

Precision (0.7792): the model is fairly reliable when it predicts a positive outcome. This means that when the DT claims a bio-ink is "Acceptable," it is correct the majority of the time, keeping False Positives relatively low.

Recall (0.7895): the model achieves a recall of 79.0%, meaning it successfully identifies most of the actually viable bio-inks, though it still misses about 21% of them (False Negatives).

(d)

[52]:
```
from sklearn.svm import SVC

svm_clf = SVC(random_state=67)

svm_clf.fit(X_train, y_train)

y_pred_svm = svm_clf.predict(X_test)

svm_acc = accuracy_score(y_test, y_pred_svm)
svm_prec = precision_score(y_test, y_pred_svm)
svm_rec = recall_score(y_test, y_pred_svm)

print("SVM")
print(f"Accuracy: {svm_acc:.4f}")
print(f"Precision: {svm_prec:.4f}")
print(f"Recall: {svm_rec:.4f}")
```

```
SVM
Accuracy: 0.7258
Precision: 0.7625
Recall: 0.8026
```

Accuracy (0.7258): the model correctly classifies approximately 72.6% of the bio-printing outcomes. This indicates a moderate ability to distinguish between acceptable and unacceptable viability based on the parameters.

Precision (0.7625): the model is fairly reliable when it predicts a positive outcome. This means that when the SVM claims a bio-ink is "Acceptable," it is correct the majority of the time, keeping False Positives relatively low.

Recall (0.8026): the model achieves a recall of 80.0%, meaning it successfully identifies most of the actually viable bio-inks, though it still misses about 20% of them (False Negatives).

[53]:
```python
# Model Comparison
print("Model Comparison")
models = ["DT", "SVM"]
accs = [dt_acc, svm_acc]
precs = [dt_prec, svm_prec]
recs = [dt_rec, svm_rec]

comparison_df = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall"],
    "DT": [dt_acc, dt_prec, dt_rec],
    "SVM": [svm_acc, svm_prec, svm_rec]
})
print(comparison_df.round(4).to_string(index=False))
```

```
Model Comparison
   Metric     DT    SVM
 Accuracy 0.7339 0.7258
Precision 0.7792 0.7625
   Recall 0.7895 0.8026
```

The decision tree is slightly better at being precise: minimizing failed experiments predicted as successes, resulting in a marginally higher overall accuracy. The SVM has better recall, making it the better choice if the primary goal is to find as many viable bio-ink candidates as possible.

Question 3

(a) Unregularized Logistic Regression fails with linearly separable data because it tries to achieve perfect classification with 100% confidence ($P = 1$). Since the sigmoid function $\sigma(z)$ only reaches exactly 1 when the input z is infinity, the model keeps increasing the magnitude of the weights $\boldsymbol{\beta}$ forever to push the score higher. Without a regularization penalty to stop this growth, the weights explode towards infinity, and the algorithm never actually converges to a final solution.

(b) The log-loss (term 1) decreases as weights grow, the penalty (term 2) grows quadratically. Because $\lambda > 0$, the penalty eventually outgrows the decay of the log-loss. This forces the total loss $J(\boldsymbol{\beta})$ to approach $+\infty$ as the weights grow in any direction. Since J is continuous and a continuous function that goes to infinity in all directions must have a minimum, this guarantees the existence of a global minimum at a finite value of $\boldsymbol{\beta}$.

(c) A larger λ forces the model to be simpler (smaller weights). The bias increases which may prevent the model from capturing complex underlying patterns in the training data and cause underfitting. The variance decreases, the model becomes less sensitive to noise in the training set, leading to more stable predictions on new data and reduces overfitting.

(d) Lowering threshold from 0.5 to 0.3 will cause:

FNR to decrease: The model correctly identifies more true positives, so fewer positive cases are missed (higher recall). FPR to increase: The model is more likely to incorrectly label negative instances as positive.

This is beneficial when False Negatives are costly or when the goal is to identify as many positive instances as possible (high recall), even at the cost of more false alarms.

Question 4

(a) SVM tries to find a hyperplane defined by $w^T x + b = 0$ that maximizes the margin, which is inversely proportional to the norm of the weight vector (margain $= \frac{2}{||w||}$). It solves $\min \frac{1}{2}||w||^2$. If features are not scaled, the distance calculations are dominated by the feature with the larger range (x_2). To have a comparable effect on the decision boundary, the weight w_2 associated with the large feature would need to be very small, while w_1 would need to be large. Since the optimizer minimizes $||w||^2$, it favors solutions where the weights are small. So the optimization may focus almost only on x_2 to define the margin and ignore x_1, this might lead to inaccurate decision boundary.

(b) Soft-margin SVM optimizes the function

$$\min \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i.$$

In a highly imbalanced dataset, the majority of the contribution to this sum comes from the negative class. The model can minimize the objective function effectively by classifying all examples as negative (yielding $\xi_i = 0$ for 99% of the data). The penalty for misclassifying the 1% positive cases is smaller than the cost (reduction in margin or increase in negative misclassifications) required to correctly classify them. Thus, the model sacrifices the minority class to satisfy the majority.

(c) To address the feature scaling issue, we could use min-max scaling as we did for question 1. Min-Max scaling rescales every feature to a fixed range, typically $[0, 1]$. For this question, we can map both x_1 and x_2 to the same $[0, 1]$ interval so that the size of the values doesn't throw off the optimization of the weight vector w. By scaling the data first, we ensure both features contribute equally to the calculation of the decision boundary.