Khushbu Soni
Student_id:34113851

3 Logistic Regression versus Bayes Classifier

Q 5

5.1 Discriminative vs Generative Models

```
from scipy.stats import multivariate_normal
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, log_loss

import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer()
x, y = breast_cancer.data, breast_cancer.target

class BayesianClassifier:

    def __init__(self, shared_cov=True, cond_ind=True):
        self.shared_cov=shared_cov
        self.cond_ind=cond_ind

    def fit(self, x, y):
        self.classes_, class_counts = np.unique(y, return_counts=True)
        self.n_ , self.p_ = x.shape
        self.k_ = len(self.classes_)
        self.cond_means_ = np.zeros(shape=(self.k_, self.p_))
        self.cond_covs_ = np.zeros(shape=(self.k_, self.p_, self.p_))

        self.class_priors_ = class_counts/len(y)
        for c in range(self.k_):
            c_rows = y==c
            self.cond_means_[c, :] = x[c_rows].mean(axis=0)
            if self.cond_ind:
                np.fill_diagonal(self.cond_covs_[c, :, :], x[c_rows].var(axis=
            else:
                self.cond_covs_[c, :, :] = np.cov(x[c_rows].T, bias=True)

        if self.shared_cov:
            shared_cov = np.moveaxis(self.cond_covs_, 0, -1).dot(self.class_pr
```

```
            self.cond_covs_[:] = shared_cov

        return self

    def predict_proba(self, x):
        m, _ = x.shape
        cond_probs = np.zeros(shape=(m, self.k_))
        for c in range(self.k_):
            # find p(x | c_k)
            # singular covariance matrices could happen (e.g., through inaccur
            cond_probs[:, c] = multivariate_normal.pdf(x,
                                                       self.cond_means_[c],
                                                       self.cond_covs_[c],
                                                       allow_singular=True)
        # find marginal probabilities p(x) by summing all the conditionals wei
        marginal_probs = cond_probs.dot(self.class_priors_)

        # find probability vector (p(c1 | x), ..., p(ck | x)) via p(ci | x)=p(
        # however, p(x) might have been rounded to 0
        # thus, compute via case distinction
        probs = np.divide((cond_probs*self.class_priors_).T,
                          marginal_probs,
                          where=marginal_probs>0, out=np.zeros(shape=(self.k_,
        return probs

    def predict(self, x):
        return np.argmax(self.predict_proba(x), axis=1)

    def decision_function(self, x):
        probs = self.predict_proba(x)
        if self.k_ == 2:
            return np.log(probs[:, 1]/probs[:, 0])
        else:
            res = np.zeros(len(x), self.k_)
            for c in range(self.k_):
                res[:, c]=np.log(probs[:, c]/(1-probs[:, c]))
            return res

    def generate(self, n, c, random_state=None):
        return multivariate_normal.rvs(self.cond_means_[c], self.cond_covs_[c]
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer()
X, Y = breast_cancer.data, breast_cancer.target

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, y, train_size=0.8, ranc
```

```
# Initialize models
logistic_reg = LogisticRegression(max_iter=10000, random_state=0)
naive_bayes = BayesianClassifier(shared_cov=False, cond_ind=True)
bayesian_shared_cov = BayesianClassifier(shared_cov=True, cond_ind=False)
bayesian_full_cov = BayesianClassifier(shared_cov=False, cond_ind=False)

# Store models in a dictionary for easy comparison
model_dict = {
    'Logistic-Regression': logistic_reg,
    'Naive Bayes (Not shared cov)': naive_bayes,
    'Bayesian (Full Cov, Not Shared)': bayesian_full_cov,
    'Bayesian (Full Cov, Shared)': bayesian_shared_cov
}

# Train and evaluate models
for model_name, model_instance in model_dict.items():
    model_instance.fit(X_train, Y_train)
    y_train_pred = model_instance.predict(X_train)
    y_test_pred = model_instance.predict(X_test)
    train_acc = accuracy_score(Y_train, y_train_pred)
    test_acc = accuracy_score(Y_test, y_test_pred)

    print(f"{model_name}")
    print(f" - Train Accuracy: {train_acc:.5f}, \n - Test Accuracy: {test_acc:
```

```
Logistic-Regression
 - Train Accuracy: 0.96264,
 - Test Accuracy: 0.94737
Naive Bayes (Not shared cov)
 - Train Accuracy: 0.92967,
 - Test Accuracy: 0.93860
Bayesian (Full Cov, Not Shared)
 - Train Accuracy: 0.93407,
 - Test Accuracy: 0.89474
Bayesian (Full Cov, Shared)
 - Train Accuracy: 0.95824,
 - Test Accuracy: 0.96491
```

Bayesian (Full Covariance, Shared) has the highest Test Accuracy (96.49%), meaning it generalizes best to unseen data, and the Train Accuracy (95.82%) is also quite high, indicating that it fits the training data well without overfitting.

Logistic Regression also performs well, with a Test Accuracy of 94.74%, but it is slightly worse than the Bayesian (Full Covariance, Shared) model.

Naive Bayes (Not Shared Covariance) has slightly lower performance than both Logistic Regression and Bayesian (Full Covariance, Shared) in terms of both train and test accuracy.

Bayesian (Full Covariance, Not Shared) has the lowest test accuracy (89.47%), indicating that it may be overfitting to the training data.

The Bayesian Classifier with Full Covariance (Shared) performs the best in terms of both train and test accuracy, making it the top-performing model in this comparison.

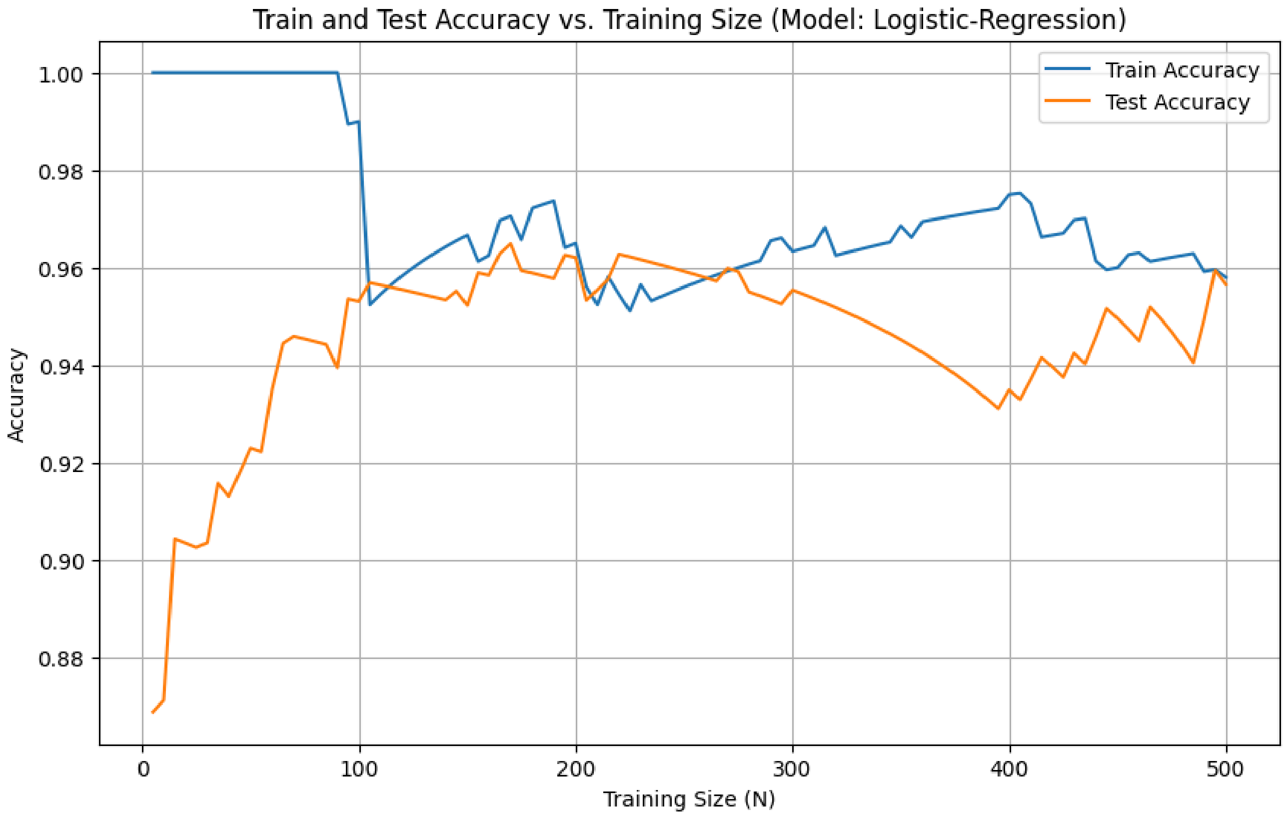
5.2 effect of training size on different models

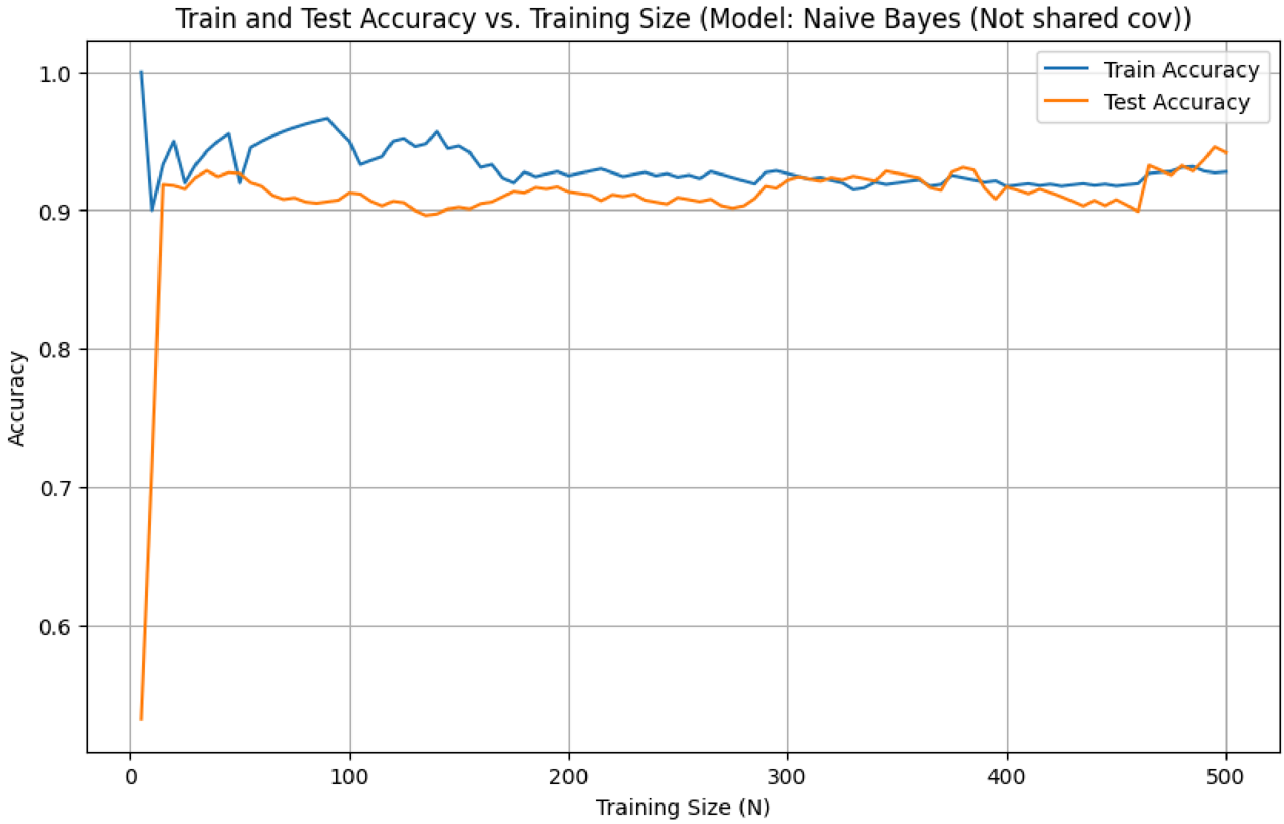
```
import pandas as pd


# Store results
results = {
    'N': [],
    'Model': [],
    'Train Accuracy': [],
    'Test Accuracy': []
}
total_samples = len(y)

# Experiment with increasing training sizes N = 5, 10, ..., up to 500
for N in range(5, 501, 5):
    if N >= total_samples:
        break  # Skip sizes larger than the total number of samples

    for _ in range(10):  # 10 training sets for each N
        # Randomly sample a training set of size N using stratified sampling
        test_size = total_samples - N  # Adjust test size dynamically
        X_train, X_test, Y_train, Y_test = train_test_split(X, y, train_size=N

        for model_name, model_instance in model_dict.items():
            # Fit the model
            model_instance.fit(X_train, Y_train)
            # Predict for training and testing data
            y_train_pred = model_instance.predict(X_train)
            y_test_pred = model_instance.predict(X_test)

            # Calculate accuracy
            train_acc = accuracy_score(Y_train, y_train_pred)
            test_acc = accuracy_score(Y_test, y_test_pred)

            # Store results
            results['N'].append(N)
            results['Model'].append(model_name)
            results['Train Accuracy'].append(train_acc)
            results['Test Accuracy'].append(test_acc)
```

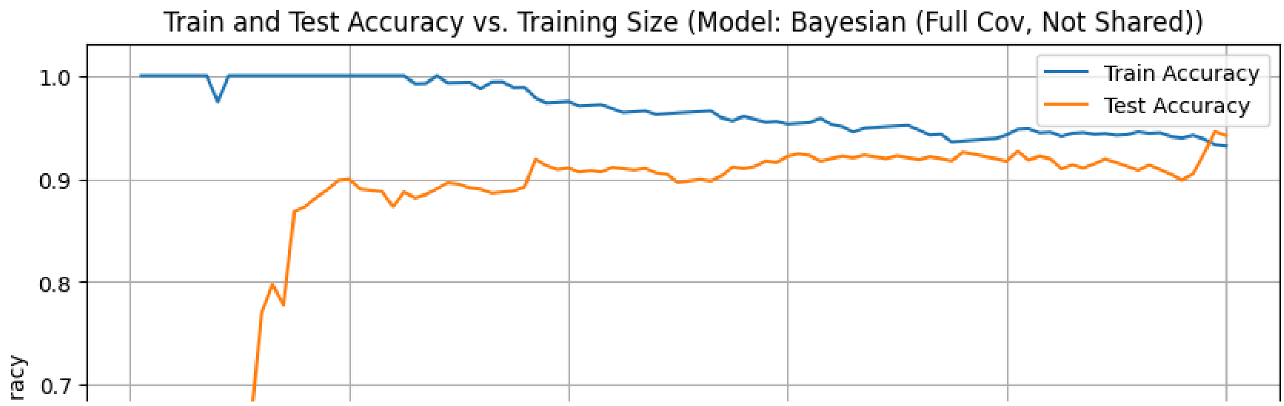
```
# Convert results to a DataFrame for easy analysis
results_df = pd.DataFrame(results)
```

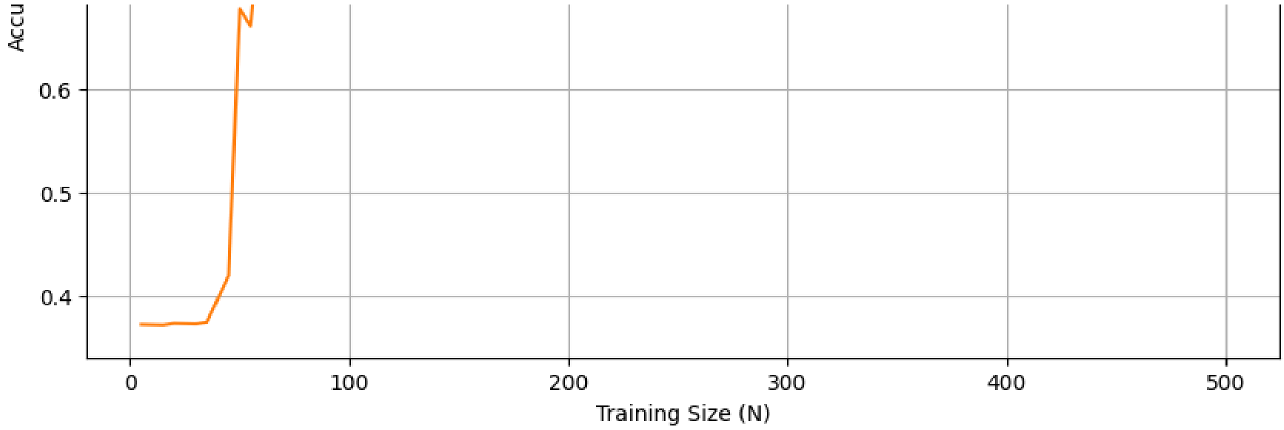
5.3 plots that compare the mean train and test performances of models.

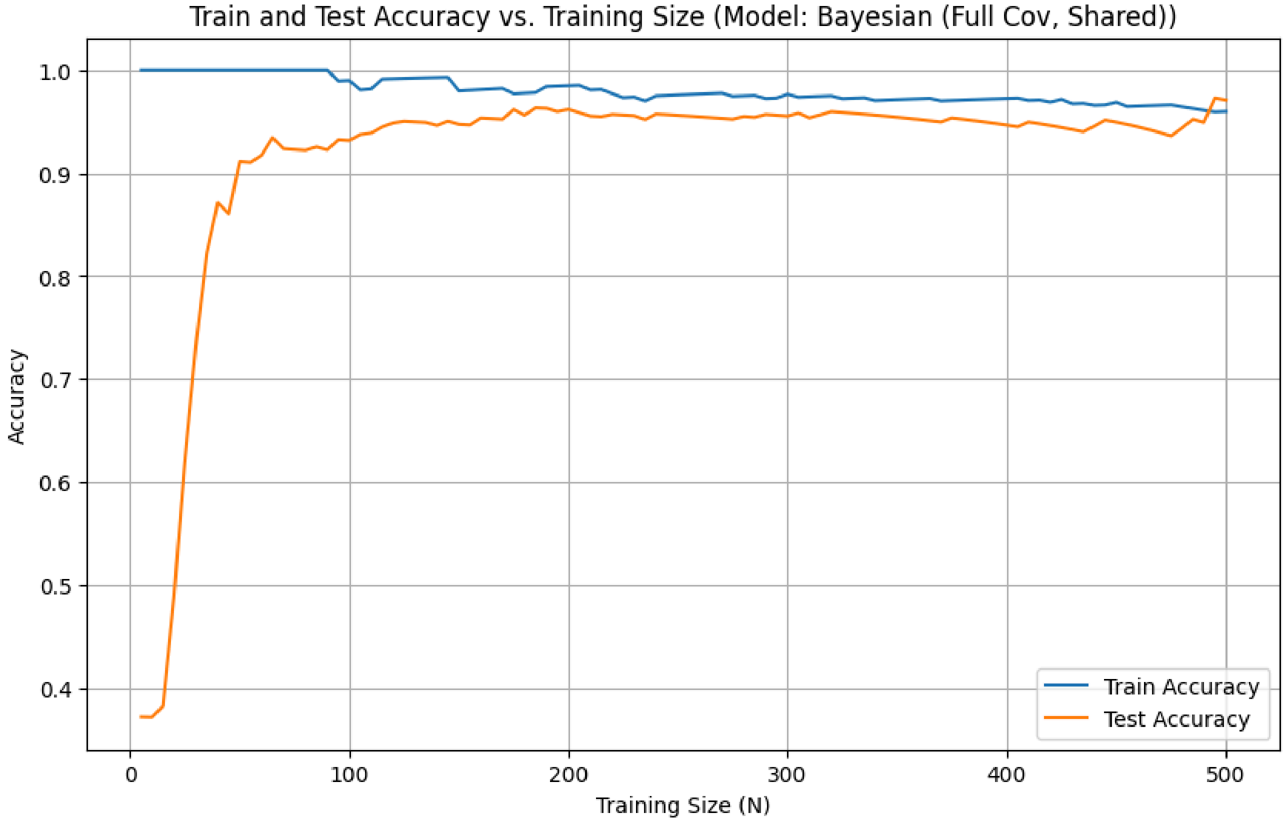
```
import matplotlib.pyplot as plt

# Plot training and test accuracy for each model
for model_name in model_dict.keys():
    model_results = results_df[results_df['Model'] == model_name]

    # Group by training size and take the mean accuracy over the 10 samples, c
    model_results_grouped = model_results.groupby('N')[['Train Accuracy', 'Tes

    plt.figure(figsize=(10, 6))
    plt.plot(model_results_grouped.index, model_results_grouped['Train Accurac
    plt.plot(model_results_grouped.index, model_results_grouped['Test Accuracy
    plt.title(f"Train and Test Accuracy vs. Training Size (Model: {model_name}
    plt.xlabel("Training Size (N)")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)
    plt.show()
```


Train and Test Accuracy vs. Training Size (Model: Logistic-Regression)
Train Accuracy
Test Accuracy
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
100
200
300
400
500
Training Size (N)


Train and Test Accuracy vs. Training Size (Model: Naive Bayes (Not shared cov))
Train Accuracy
Test Accuracy
Accuracy
1.0
0.9
0.8
0.7
0.6
0
100
200
300
400
500
Training Size (N)


Train and Test Accuracy vs. Training Size (Model: Bayesian (Full Cov, Not Shared))
Train Accuracy
Test Accuracy
1.0
0.9
0.8
0.7


Accu
0.6
0.5
0.4
0
100
200
300
400
500
Training Size (N)


Train and Test Accuracy vs. Training Size (Model: Bayesian (Full Cov, Shared))
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
100
200
300
400
500
Training Size (N)
Train Accuracy
Test Accuracy