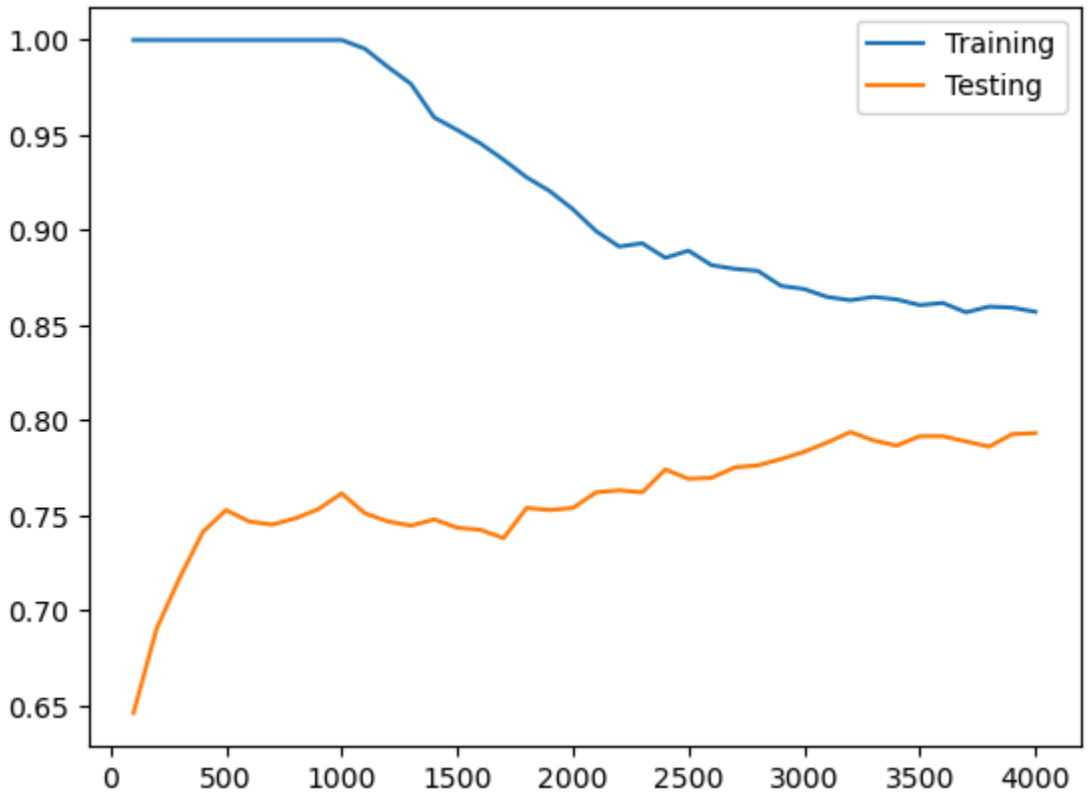


📣 **HAND-IN** 📣 : On Moodle, upload your plot (where M ranges up to 4000).

Task 4: Pushing the Limits (4 pts)

Now that you've trained a logistic regression classifier on this task, see how far you can "push the limit" and train as accurate a classifier as possible. This task is open-ended: you can try hyperparameter optimization, new forms of data augmentation, or other techniques.

Hyperparameter tuning: You may be able to increase the accuracy by optimizing the amount of regularization. In sklearn, the regularization strength of the logistic regression classifier can be tuned by setting the C parameter in the constructor of LogisticRegression. To optimize hyperparameters in a principled way, we need three data sets: training, validation, and testing. However, the data you are provided with contains just training and testing sets. Hence, you should sub-divide the training data into two subsets: "real" training (this is what you call fit() on), and validation (which you use to estimate how good each hyperparameter configuration is). As a suggestion, try using 80% of trainingFaces and trainingLabels for training, and the remaining 20% for validation.

Data augmentation: In addition to left-right flips, you can also try adding a small amount of (typically Gaussian) random noise to each pixel (using a small standard deviation of the Gaussian distribution, so that the face images still looks like a face). See np.random.randn. In order to avoid manually overfitting to the test data, you should also experiment with different forms of data augmentation on a separate validation set; then, once you find a good method of augmentation, test out the resulting classifier on the testing set.

📣 **HAND-IN** 📣 : On Moodle, upload your code above, and report the final testing accuracy.

Hyperparameter tuning

```
import sklearn.linear_model
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
```

```
# Split the augmented training data into training and validation sets
train_faces, val_faces, train_labels, val_labels = train_test_split(
    augmentedTrainingFaces, augmentedTrainingLabels, test_size=0.2, random_state=42)

print("Training set size: ", train_faces.shape[0])
print("Validation set size: ", val_faces.shape[0])

# divide the training set into real training and validation sets
real_train_faces, val_faces, real_train_labels, val_labels = train_test_split(
    train_faces, train_labels, test_size=0.2, random_state=42)

print("Real training set size: ", real_train_faces.shape[0])
print("Real validation set size: ", val_faces.shape[0])
```

```
Training set size:  3200
Validation set size:  800
Real training set size:  2560
Real validation set size:  640
```

```
# Rough Hyperparameter tuning
C_values = [0.01, 0.1, 1, 10, 100, 1000]  # Different values of C to try
best_C = None
best_val_accuracy = 0
for C in C_values:
    logisticRegressor = sklearn.linear_model.LogisticRegression(C=C, max_iter=500)
    logisticRegressor.fit(real_train_faces, real_train_labels)
    val_accuracy = logisticRegressor.score(val_faces, val_labels)
    print(f"C={C}, Validation accuracy: {val_accuracy}")
    if val_accuracy > best_val_accuracy:
        best_val_accuracy = val_accuracy
        best_C = C
print(f"Best C: {best_C}, Best validation accuracy: {best_val_accuracy}")
```

```
C=0.01, Validation accuracy: 0.809375
C=0.1, Validation accuracy: 0.7796875
C=1, Validation accuracy: 0.7578125
C=10, Validation accuracy: 0.7515625
C=100, Validation accuracy: 0.7515625
C=1000, Validation accuracy: 0.753125
Best C: 0.01, Best validation accuracy: 0.809375
```

```
# Fine Hyperparameter tuning
C_values = np.arange(0.001, 0.1, 0.001)

best_C = None
```

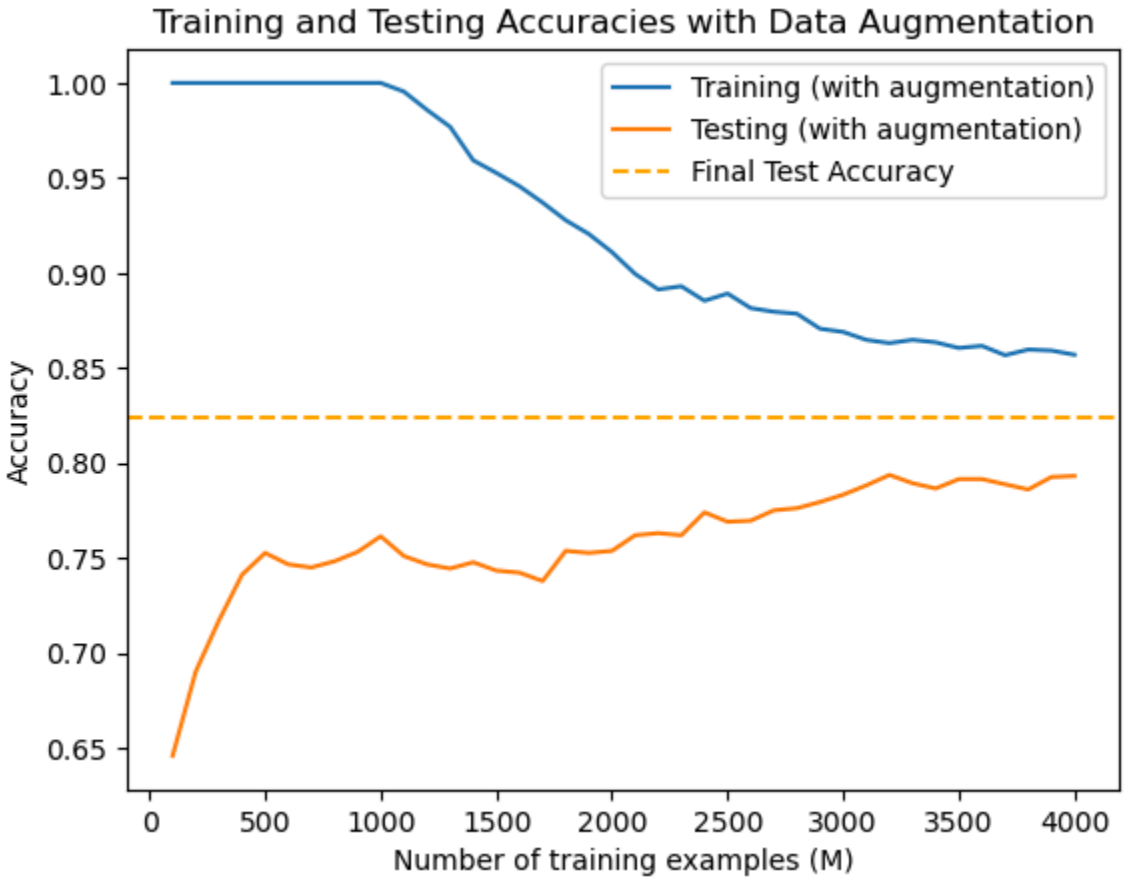
```
best_val_accuracy = 0
for C in C_values:
    logisticRegressor = sklearn.linear_model.LogisticRegression(C=C, max_iter=500)
    logisticRegressor.fit(real_train_faces, real_train_labels)
    val_accuracy = logisticRegressor.score(val_faces, val_labels)
    # print(f"C={C}, Validation accuracy: {val_accuracy}")
    if val_accuracy > best_val_accuracy:
        best_val_accuracy = val_accuracy
        best_C = C
print(f"Best C: {best_C}, Best validation accuracy: {best_val_accuracy}")
```

```
Best C: 0.008, Best validation accuracy: 0.809375
```

```
# Train final model with best C on the entire augmented training set
final_logisticRegressor = sklearn.linear_model.LogisticRegression(C=best_C, max_iter=5
final_logisticRegressor.fit(augmentedTrainingFaces, augmentedTrainingLabels)
final_test_accuracy = final_logisticRegressor.score(testingFaces, testingLabels)
print(f"Final testing accuracy: {final_test_accuracy}")
```

```
Final testing accuracy: 0.8238512035010941
```

```
# Replot the accuracies from Task 3 for comparison
plt.plot(Mvalues, trainingAccuracies, label="Training (with augmentation)")
plt.plot(Mvalues, testingAccuracies, label="Testing (with augmentation)")
plt.axhline(y=final_test_accuracy, color='orange', linestyle='--', label="Final Test A
plt.legend()
plt.xlabel("Number of training examples (M)")
plt.ylabel("Accuracy")
plt.title("Training and Testing Accuracies with Data Augmentation")
plt.show()
```


Training and Testing Accuracies with Data Augmentation
Training (with augmentation)
Testing (with augmentation)
Final Test Accuracy
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
500
1000
1500
2000
2500
3000
3500
4000
Number of training examples (M)