Applied Machine Learning

Homework 6

Anand Kamble
Department of Scientific Computing
Florida State University

Introduction

In this homework, we implemented the LogitBoost algorithm using univariate (based on a single feature) piecewise constant regressors as weak learners, as described on slide 29 of the Boosting slides. We used 10 bins for the piecewise constant regressors. At each boosting iteration, we chose the weak learner that obtained the largest reduction in the loss function on the training set $D = \{(x_i, y_i), i = 1, \dots, N\}$, with $y_i \in \{0, 1\}$:

$$L = \sum_{i=1}^{N} \ln(1 + \exp[-\tilde{y}_i h(x_i)])$$

where $\tilde{y}_i = 2y_i - 1$ takes values ± 1 and $h(x) = h_1(x) + \cdots + h_k(x)$ is the boosted classifier. We applied this method to multiple datasets, and the code is designed to be easily adapted to the Gisette, Dexter, and Madelon datasets.

Environment Setup

The environment setup was as follows:

```
conda create -n "homework6" python=3.11 &&\
conda activate homework6 &&\
pip install numpy pandas matplotlib scikit-learn tqdm
```

The code for this homework is available on GitHub:
`https://github.com/anand-kamble/FSU-assignments/blob/main/Machine%20Learning/HW06/main.py`

Implementation of LogitBoost with Univariate Piecewise Constant Regressors

LogitBoost Class Implementation

The LogitBoost algorithm is implemented as a Python class named `LogitBoost`. Below is the code and its corresponding explanation.

Class Initialization The `LogitBoost` class initializes important parameters used throughout the boosting process.

```python
class LogitBoost:
    def __init__(self, num_iterations=100, num_bins=10):
        self.num_iterations = num_iterations
        self.num_bins = num_bins
        self.weak_learners = []
```

Listing 1: LogitBoost Class Initialization

Weak Learner Fitting The `_fit_weak_learner` method fits a univariate piecewise constant regressor to the weighted data.

```python
def _fit_weak_learner(self, X, z, w):
    best_feature = None
    best_bin_means = None
    best_loss = float('inf')

    for feature in range(X.shape[1]):
        bins = np.linspace(X[:, feature].min(), X[:, feature].max(), self.
            num_bins)
        bin_indices = np.digitize(X[:, feature], bins)
        bin_means = np.zeros(self.num_bins)

        for b in range(1, self.num_bins + 1):
            mask = bin_indices == b
            if np.sum(mask) > 0:
                bin_means[b - 1] = np.sum(w[mask] * z[mask]) / np.sum(w[mask])

        predictions = bin_means[bin_indices - 1]
        loss = np.sum(w * (z - predictions) ** 2)

        if loss < best_loss:
            best_loss = loss
            best_feature = feature
            best_bin_means = bin_means

    return best_feature, best_bin_means
```

Listing 2: Weak Learner Fitting

Training Process The `fit` method implements the core boosting loop.

```python
def fit(self, X, y):
    N = X.shape[0]
    h = np.zeros(N)  # Initial classifier
    self.loss_history = []

    for iteration in tqdm(range(self.num_iterations), desc="Fitting LogitBoost"
        ):
        z = y / (1 + np.exp(y * h))
        w = np.exp(y * h) / (1 + np.exp(y * h)) ** 2

        feature, bin_means = self._fit_weak_learner(X, z, w)
        bin_indices = np.digitize(X[:, feature], np.linspace(X[:, feature].min
            (), X[:, feature].max(), self.num_bins))
        h_new = bin_means[bin_indices - 1]
        h += h_new

        loss = np.sum(np.log(1 + np.exp(-y * h)))
        self.loss_history.append(loss)

        self.weak_learners.append((feature, bin_means))
```

Listing 3: Fit Method

Prediction Function The `predict` method computes the boosted classifier's output.

```
def predict(self, X):
    N = X.shape[0]
    h = np.zeros(N)

    for feature, bin_means in self.weak_learners:
        bin_indices = np.digitize(X[:, feature], np.linspace(X[:, feature].min
            (), X[:, feature].max(), self.num_bins))
        h += bin_means[bin_indices - 1]

    return np.sign(h), h
```

Listing 4: Prediction Function

Data Loading and Preprocessing

We loaded the Dexter dataset and normalized each feature to have zero mean and unit variance using the training set statistics:

```
X_train, y_train, X_test, y_test = load_dataset("dexter")

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Listing 5: Data Loading and Preprocessing

The `load_dataset` function is defined in the `dataset` module and is available on GitHub at `https://github.com/anand-kamble/FSU-assignments/blob/main/Machine%20Learning/dataset/__init__.py#L44`.

Data Export for TikZ Plotting

To generate high-quality plots directly within LaTeX using TikZ and PGFPlots, we exported the plot data from Python into `.dat` files. This allowed us to import the data into LaTeX and recreate the plots with consistent styling and fonts. For example:

```
np.savetxt('plot_data.dat', data_array, fmt='%.6f', header='X Y', comments='')
```

Listing 6: Exporting Plot Data

By saving the data in this way, we seamlessly integrated our plots into the LaTeX document using TikZ.

Parallelization with Multiprocessing

To speed up the training process for different values of k, we utilized Python's `multiprocessing` module to parallelize the computations. By creating a pool of worker processes, we executed the `train_and_evaluate` function concurrently for each k:

```
with mp.Pool() as pool:
    results = pool.map(partial(train_and_evaluate, X_train=X_train, y_train=
        y_train, X_test=X_test, y_test=y_test), ks)
```

Listing 7: Parallelization with Multiprocessing

This approach significantly reduced computation time by leveraging multiple CPU cores.

Results for Gisette Dataset

Training Loss vs. Iteration (k=500)

We trained the LogitBoost model on the Gisette dataset with $k = 500$ boosting iterations. The training loss over iterations is shown in Figure 1.

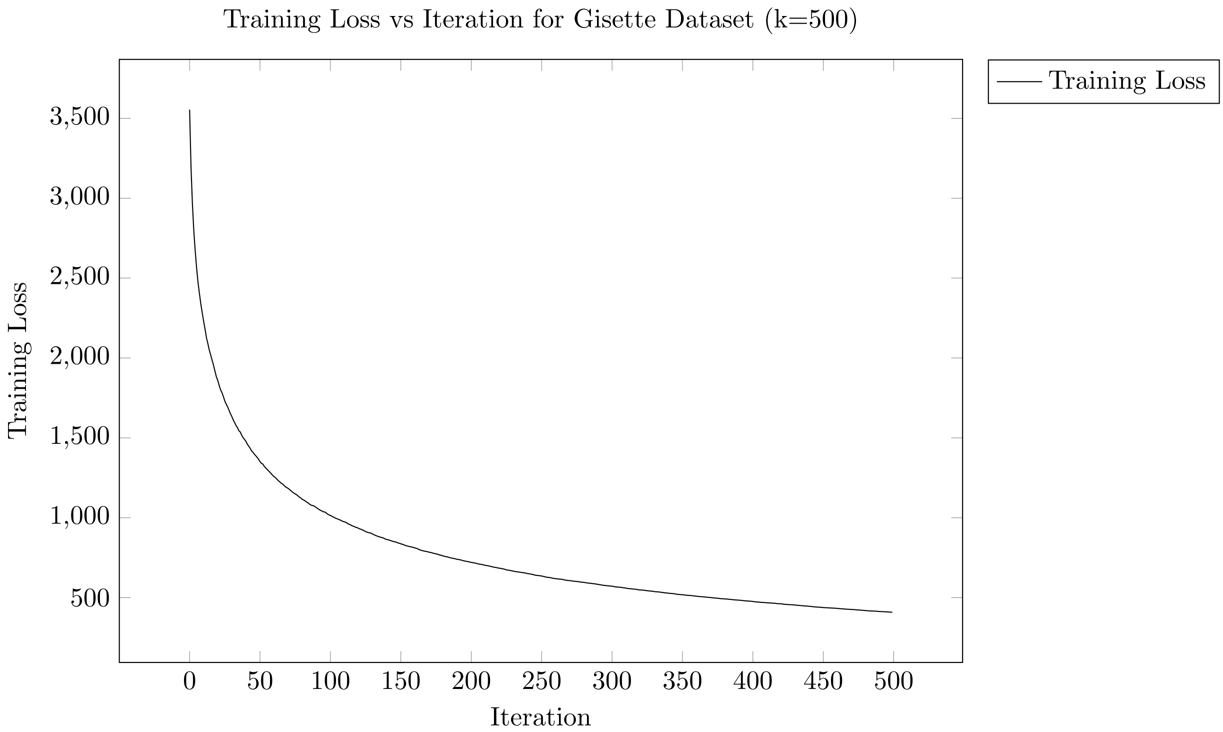

Figure 1: Training Loss vs. Iteration for Gisette Dataset with $k = 500$

Misclassification Errors

The misclassification errors on the training and test sets for different values of k are presented in Table 1.

Table 1: Misclassification Errors for Gisette Dataset

Number of Iterations (k)	**Training Error**	**Test Error**
10	0.132833	0.138000
30	0.080667	0.089000
100	0.034000	0.047000
300	0.007000	0.032000
500	0.001667	0.033000

Misclassification Error vs. Number of Iterations

Figure 2 illustrates the relationship between misclassification error and the number of boosting iterations k.

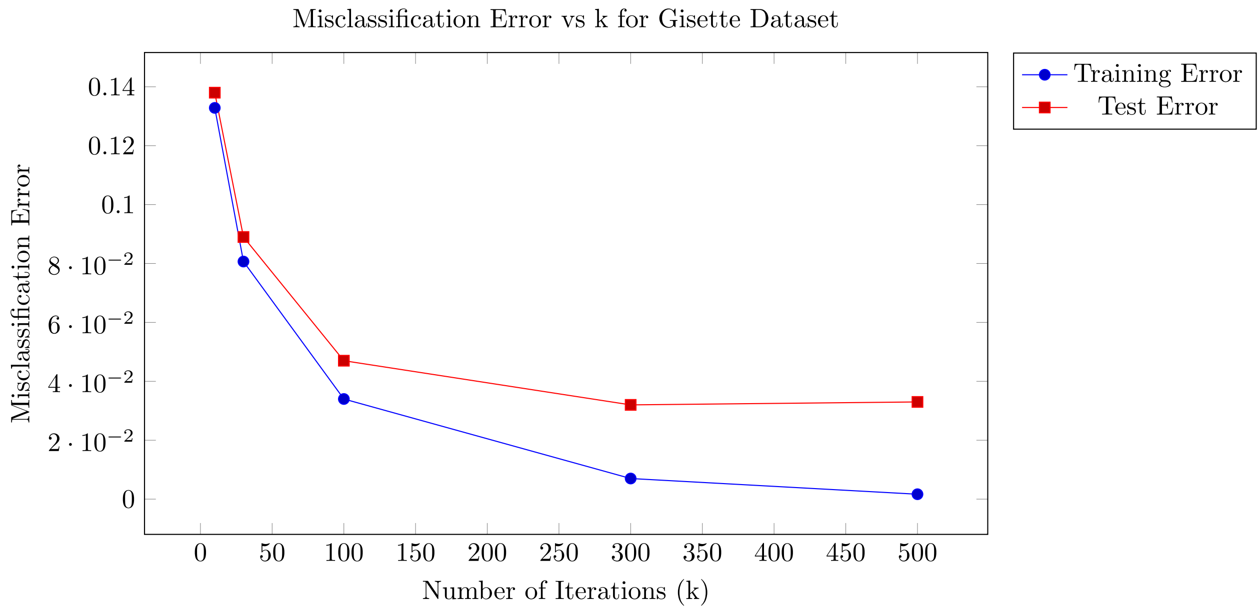

Figure 2: Misclassification Error vs. Number of Iterations for Gisette Dataset

ROC Curves (k=300)

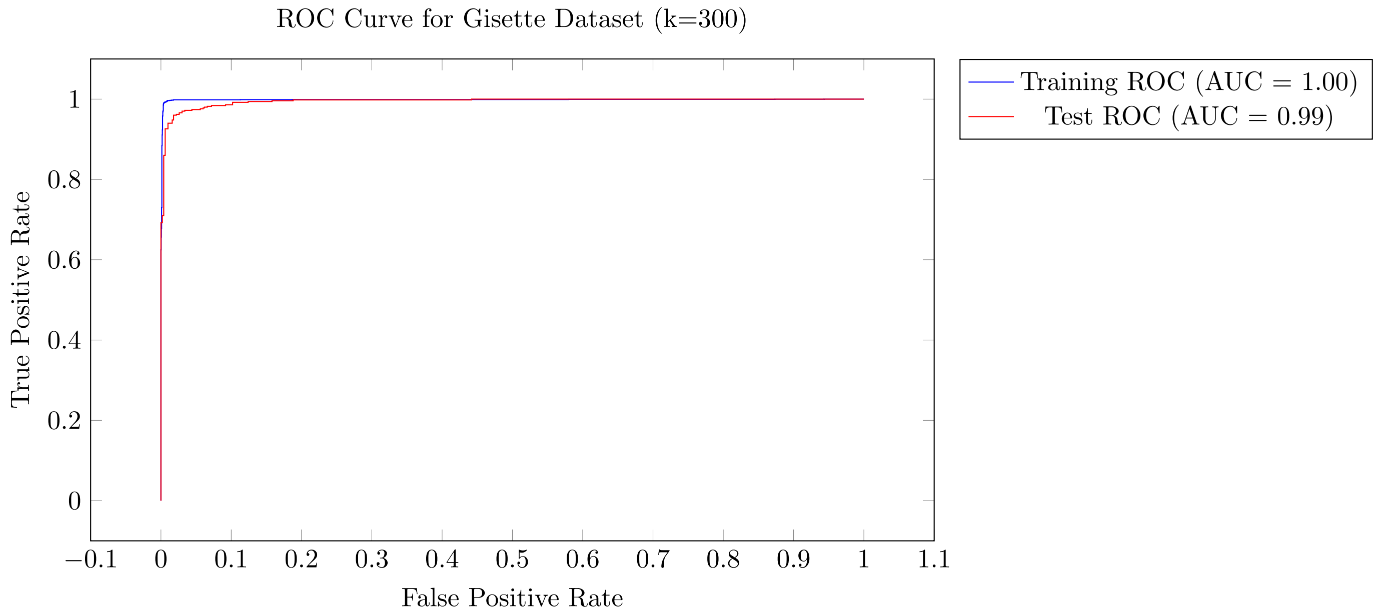

Figure 3: ROC Curves for Gisette Dataset with $k = 300$

Results for Dexter Dataset

Training Loss vs. Iteration (k=500)

We trained the LogitBoost model on the Dexter dataset with $k = 500$ boosting iterations. The training loss over iterations is shown in Figure 4.

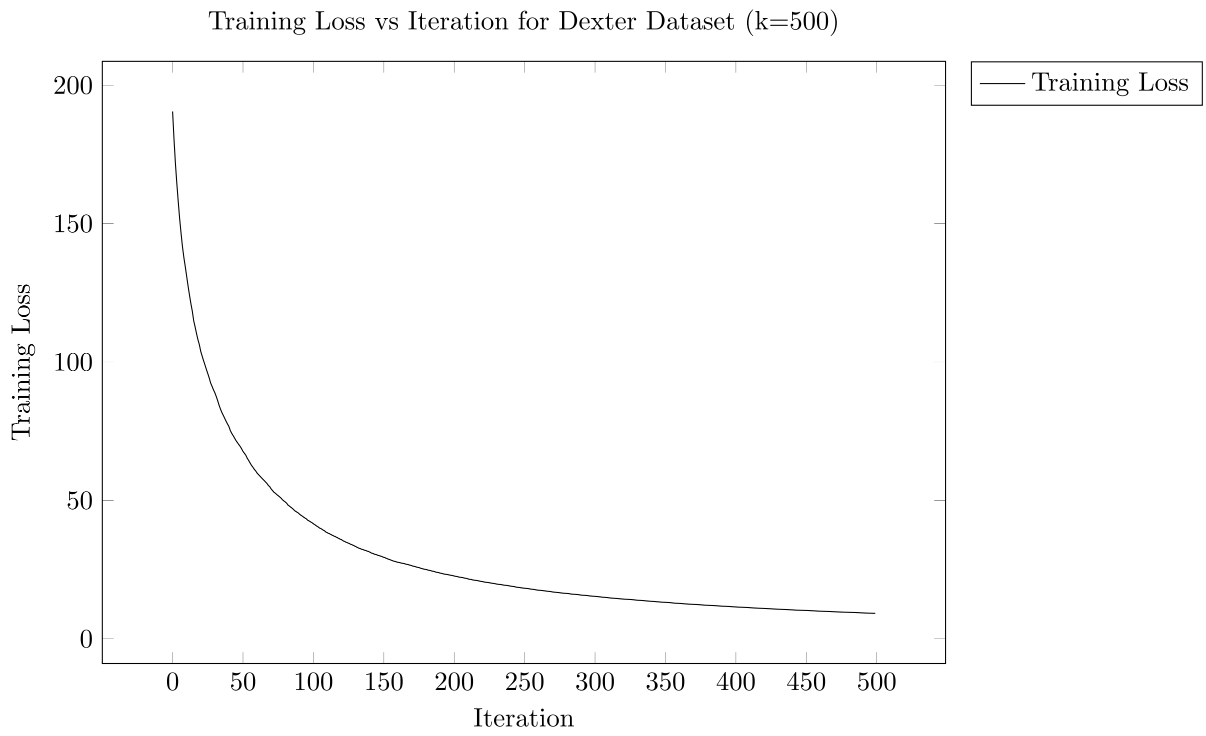

Figure 4: Training Loss vs. Iteration for Dexter Dataset with $k = 500$

Misclassification Errors

The misclassification errors on the training and test sets for different values of k are presented in Table 2.

Table 2: Misclassification Errors for Dexter Dataset

Number of Iterations (k)	**Training Error**	**Test Error**
10	0.116667	0.186667
30	0.023333	0.126667
100	0.000000	0.140000
300	0.000000	0.146667
500	0.000000	0.156667

Misclassification Error vs. Number of Iterations

Figure 5 illustrates the relationship between misclassification error and the number of boosting iterations k.

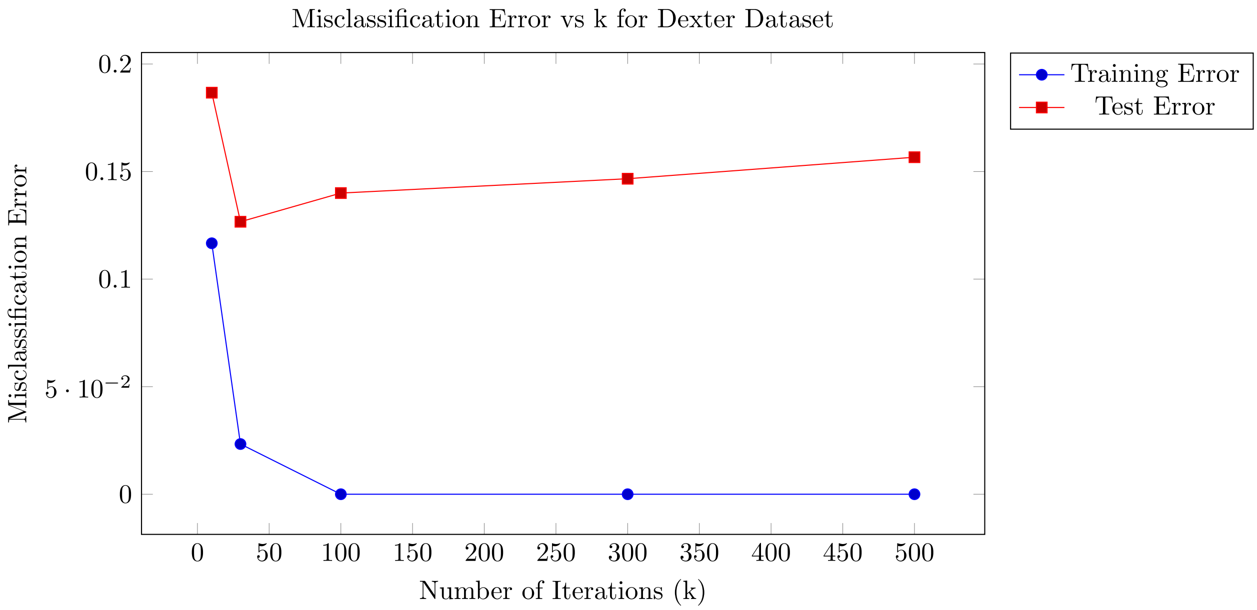

Figure 5: Misclassification Error vs. Number of Iterations for Dexter Dataset

ROC Curves (k=300)

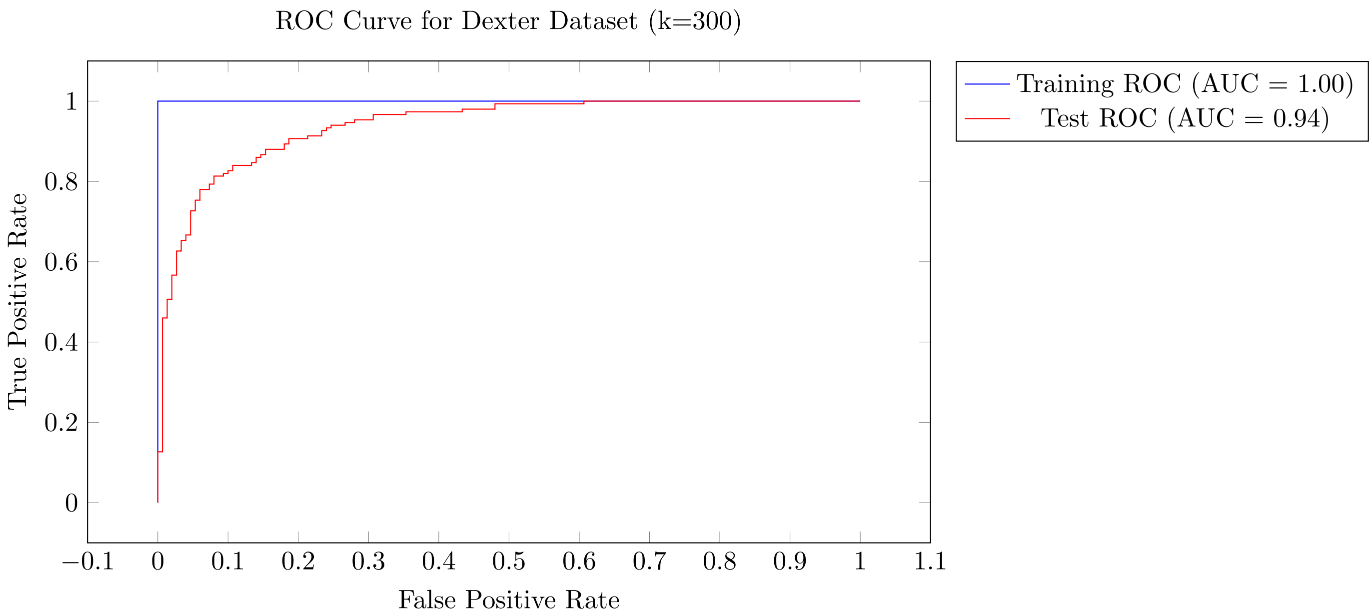

Figure 6: ROC Curves for Dexter Dataset with $k = 300$

Results for Madelon Dataset

Training Loss vs. Iteration (k=500)

We trained the LogitBoost model on the Madelon dataset with $k = 500$ boosting iterations. The training loss over iterations is shown in Figure 7.

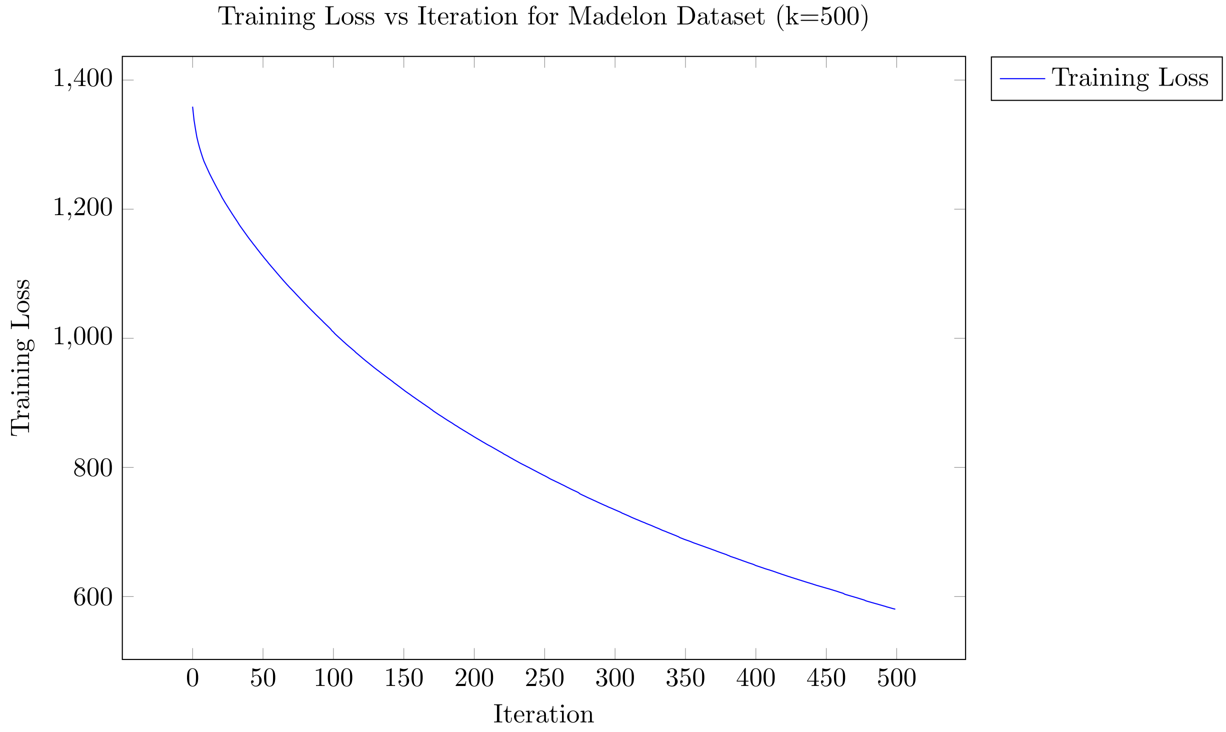

Figure 7: Training Loss vs. Iteration for Madelon Dataset with $k = 500$

Misclassification Errors

The misclassification errors on the training and test sets for different values of k are presented in Table 3.

Table 3: Misclassification Errors for Madelon Dataset

Number of Iterations (k)	**Training Error**	**Test Error**
10	0.339000	0.358333
30	0.283500	0.380000
100	0.162500	0.396667
300	0.035500	0.421667
500	0.009500	0.425000

Misclassification Error vs. Number of Iterations

Figure 8 illustrates the relationship between misclassification error and the number of boosting iterations k.

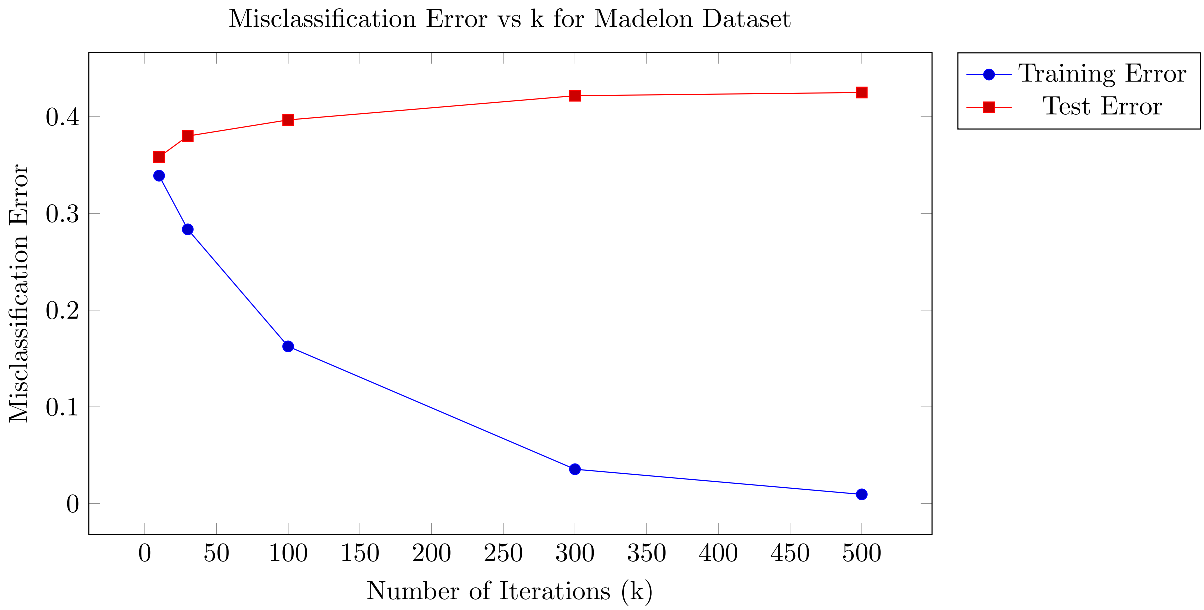

Figure 8: Misclassification Error vs. Number of Iterations for Madelon Dataset

ROC Curves (k=300)

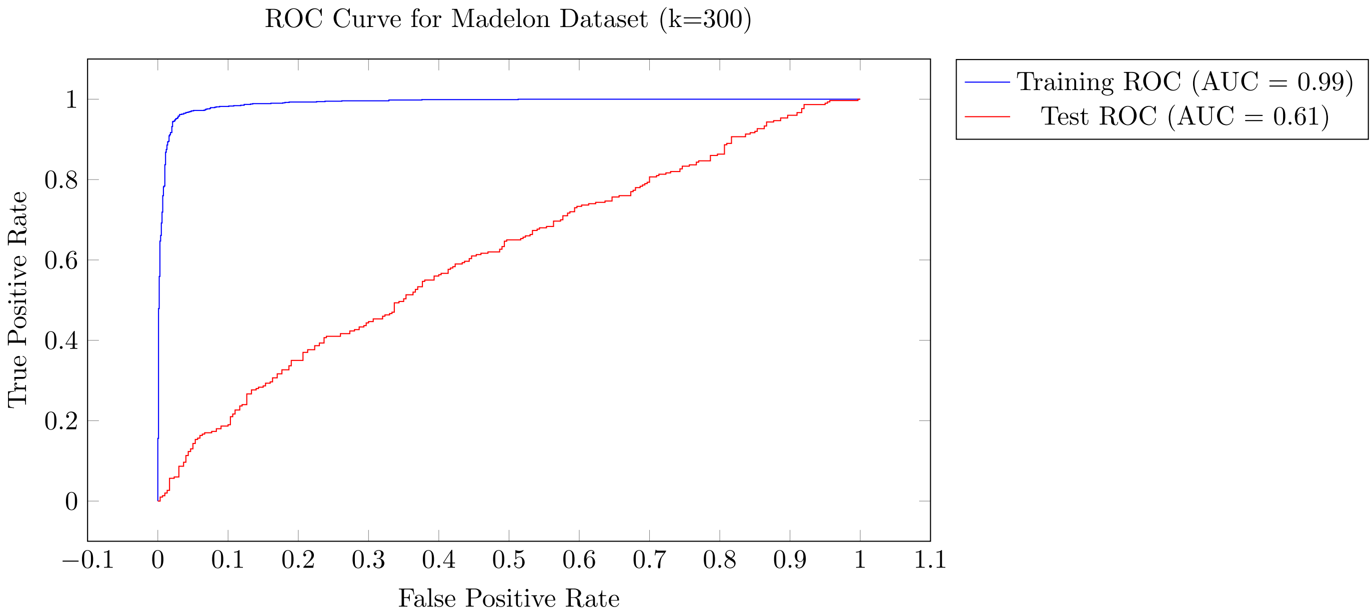

Figure 9: ROC Curves for Madelon Dataset with $k = 300$