p9120_hw3_q2q3

November 7, 2022

1 P9120 HW3 Q2 and Q3

In [5]:
```
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import zero_one_loss
from sklearn.model_selection import train_test_split

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, Activation
from sklearn.metrics import RocCurveDisplay

from numpy.random import seed
seed(2022)
tf.random.set_seed(2)
```

1.1 Q2

The features $X_1, ..., X_{10}$ are standard independent Gaussian, and the deterministic target Y is defined by

$$Y = \begin{cases} 1 \text{ if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5) \\ -1 \text{ otherwise.} \end{cases}$$

Here $\chi^2_{10}(0.5) = 9.34$ is the median of a chi-squared random variable with 10 degrees of freedom

(b) Redo the computations for the example of Figure 10.2. Plot the training error as well as test error, and discuss its behavior.

In [6]:
```
# test = 2000
X, y = datasets.make_hastie_10_2(n_samples=12000, random_state=2022)
```

Split the training and testing set, train the single stump

```
In [56]: X_train, X_test, y_train, y_test = train_test_split(
             X, y, test_size=2_000, shuffle=False
         )

         dt_stump = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
         dt_stump.fit(X_train, y_train)
         dt_stump_err_train = 1.0 - dt_stump.score(X_train, y_train)
         dt_stump_err = 1.0 - dt_stump.score(X_test, y_test)

         dt = DecisionTreeClassifier(max_depth=8, min_samples_leaf=2)
         dt.fit(X_train, y_train)
         dt_err_train = 1.0 - dt.score(X_train, y_train)
         dt_err = 1.0 - dt.score(X_test, y_test)
```

```
In [57]: print(dt.tree_.node_count)
```

```
249
```

The decision tree we got has 249 total number of nodes, similar to the text book example

1.1.1 Adaboost with real SAMME.R

We now set the hyperparameters for our AdaBoost classifiers, and define the real AdaBoost classifier and fit it to the training set.

```
In [47]: N_ESTIMATORS = 400
         LR = 1.0
```

```
In [48]: ada_real = AdaBoostClassifier(
             base_estimator=dt_stump,
             learning_rate=LR,
             n_estimators=N_ESTIMATORS,
             algorithm="SAMME.R",
         )
         ada_real.fit(X_train, y_train)
```

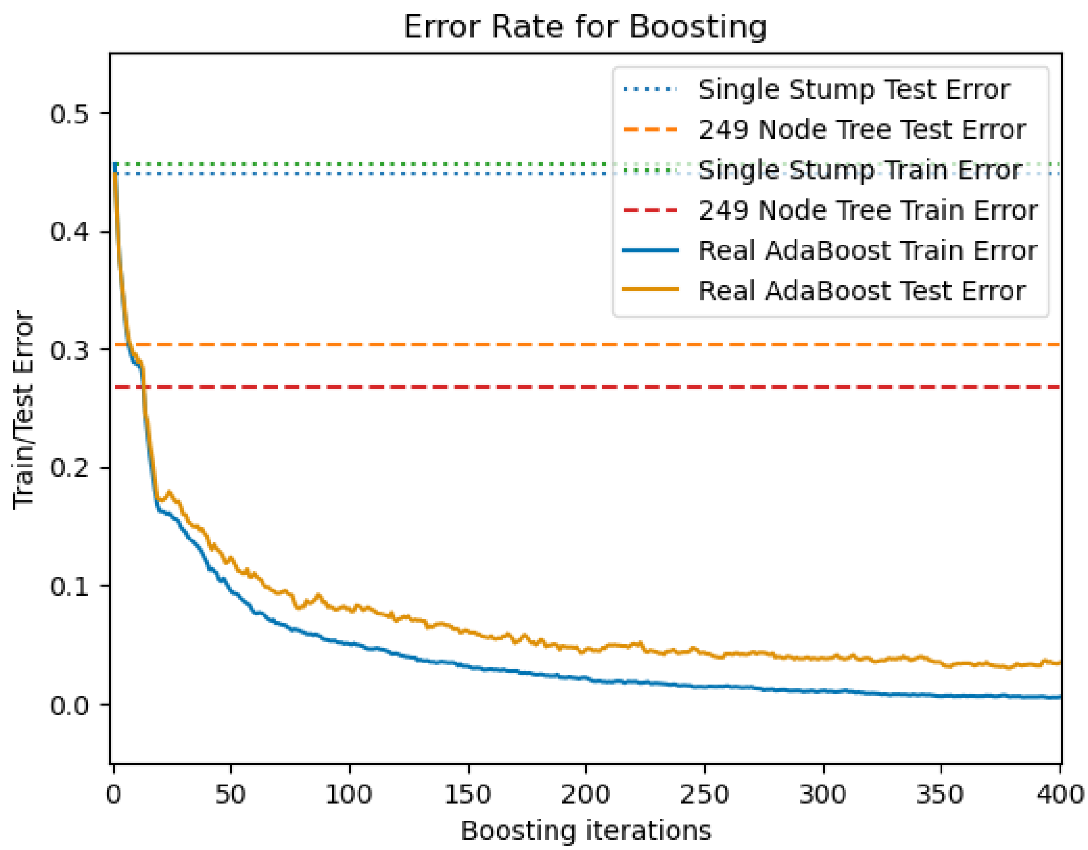
```
Out[48]: AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                            n_estimators=400)
```

1.1.2 Compute test errors

```
In [49]: ada_real_err_train = np.zeros((N_ESTIMATORS,))
         for i, y_pred in enumerate(ada_real.staged_predict(X_train)):
             ada_real_err_train[i] = zero_one_loss(y_pred, y_train)

         ada_real_err = np.zeros((N_ESTIMATORS,))
         for i, y_pred in enumerate(ada_real.staged_predict(X_test)):
             ada_real_err[i] = zero_one_loss(y_pred, y_test)
```

1.1.3 Plot the figure

In [86]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

colors = sns.color_palette("colorblind")

ax.plot([1, N_ESTIMATORS], [dt_stump_err] * 2, ":",
        label="Single Stump Test Error")
ax.plot([1, N_ESTIMATORS], [dt_err] * 2, "--",
        label="249 Node Tree Test Error")

ax.plot([1, N_ESTIMATORS], [dt_stump_err_train] * 2, ":",
        label="Single Stump Train Error")
ax.plot([1, N_ESTIMATORS], [dt_err_train] * 2, "--",
        label="249 Node Tree Train Error")


ax.plot(
    np.arange(N_ESTIMATORS) + 1,
    ada_real_err_train,
    label="Real AdaBoost Train Error",
    color=colors[0],
)

ax.plot(
    np.arange(N_ESTIMATORS) + 1,
    ada_real_err,
    label="Real AdaBoost Test Error",
    color=colors[1],
)

ax.set_ylim((-0.05, 0.55))
ax.set_xlim((-1, 401))
ax.set_xlabel("Boosting iterations")
ax.set_ylabel("Train/Test Error")
ax.set_title("Error Rate for Boosting")

leg = ax.legend(loc="upper right", fancybox=True)
leg.get_frame().set_alpha(0.7)

plt.show()
```




```
In [84]: print("The training error for Adaboost after 400 iterations is %.3f \n\
         The testing error for Adaboost after 400 iterations is %.3f \n\
         The training error for single stump is %.3f \n\
         The testing error for single stump is %.3f \n\
         The training error for 249 Node decision tree is %.3f \n\
         The testing error for 249 Node decision tree is %.3f" %
         (ada_real_err_train[-1], ada_real_err[-1],
          dt_stump_err_train, dt_stump_err,
          dt_err_train, dt_err))
```

```
The training error for Adaboost after 400 iterations is 0.005
The testing error for Adaboost after 400 iterations is 0.034
The training error for single stump is 0.456
The testing error for single stump is 0.448
The training error for 249 Node decision tree is 0.268
The testing error for 249 Node decision tree is 0.303
```

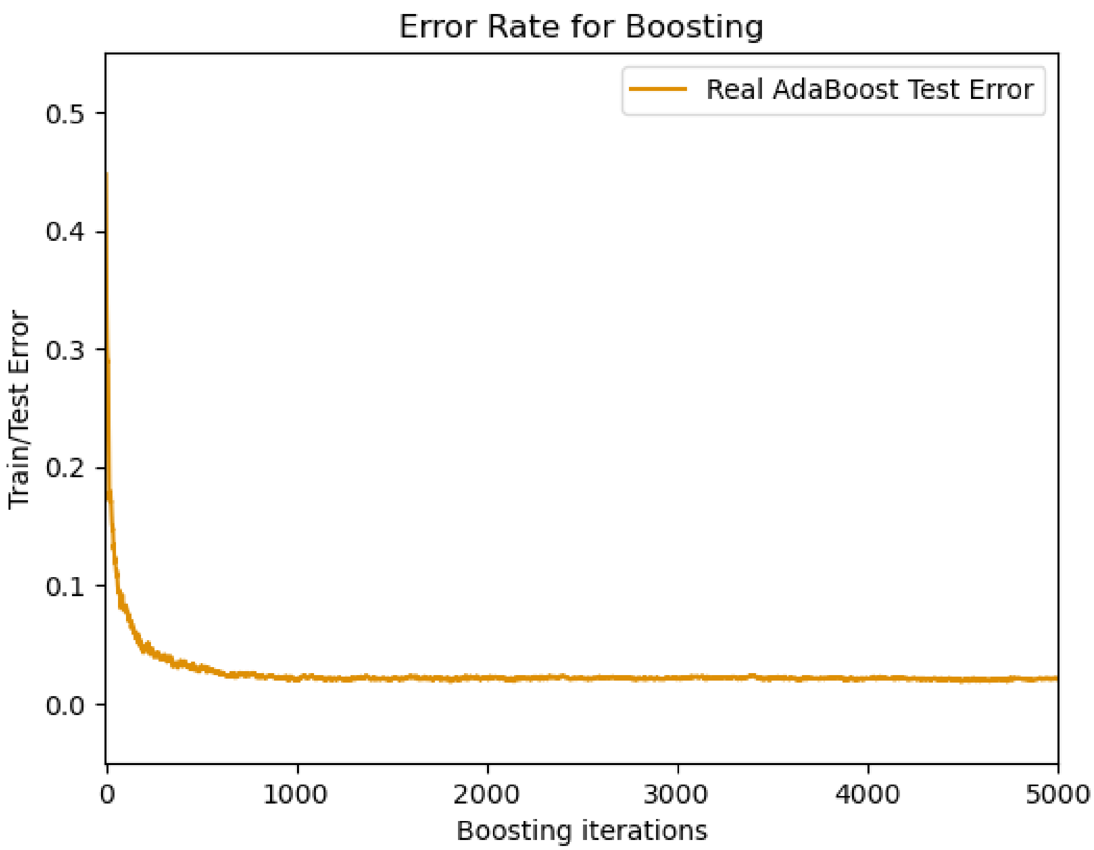
- Applying the single binary classification tree (single stump) gives a very poor test set error rate (~0.45), compared to 50% for a random classifier.

- However, as we increase the boosting iterations, the error rate steadily decreases, reaching 0.5% on training set after 400 iterations, and 3.4% on testing set after 400 iterations.
- Adaboost also quickly outperforms a single large classification tree (eror rate 30%)

(c) Investigate the number of iterations needed to make the test error finally start to rise.

In [118]:
```
N_ESTIMATORS = 10000

ada_real = AdaBoostClassifier(
    base_estimator=dt_stump,
    learning_rate=LR,
    n_estimators=N_ESTIMATORS,
    algorithm="SAMME.R",
)
ada_real.fit(X_train, y_train)

ada_real_err = np.zeros((N_ESTIMATORS,))
for i, y_pred in enumerate(ada_real.staged_predict(X_test)):
    ada_real_err[i] = zero_one_loss(y_pred, y_test)
```

In [119]:
```
fig = plt.figure()
ax = fig.add_subplot()
ax.plot(
    np.arange(N_ESTIMATORS) + 1,
    ada_real_err,
    label="Real AdaBoost Test Error",
    color=colors[1],
)

ax.set_ylim((-0.05, 0.55))
ax.set_xlim((-1, 10001))
ax.set_xlabel("Boosting iterations")
ax.set_ylabel("Train/Test Error")
ax.set_title("Error Rate for Boosting")

leg = ax.legend(loc="upper right", fancybox=True)
leg.get_frame().set_alpha(0.7)

plt.show()
```




```
In [120]: ada_real_err[-1]

Out[120]: 0.020499999999999963
```

The test error was steady at 0.02 and it does not increase

(d) Change the setup of this example as follows: define two classes, with the features in Class 1 being $X_1, X_2, ...X_{10}$, standard independent Gaussian variates. In Class 2, the features $X_1, X_2, ...X_{10}$, are also standard independent Gaussian, but conditioned on the event $\sum_j X_j^2 > 12$. Now the classes have significant overlap in feature space. Repeat the AdaBoost experiments as in Figure 10.2 and discuss the results.

```
In [142]: X_1 = np.empty([6000, 10])
          y_1 = np.ones(6000)
          i = 0
          while i < 5999:
              X_temp, _ = datasets.make_hastie_10_2(n_samples=1)
              if np.sum(X_temp ** 2) > 12:
                  X_1[i] = X_temp
                  i += 1
```

```
X_0, _ = datasets.make_hastie_10_2(n_samples=6000, random_state=1)
y_0 = np.negative(y_1)
```

In [147]:
```
X = np.concatenate((X_0, X_1))
y = np.concatenate((y_0, y_1))
```

Repeat the experiment:

In [156]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=2_000, shuffle=False
)

dt_stump = DecisionTreeClassifier(max_depth=1, min_samples_leaf=1)
dt_stump.fit(X_train, y_train)
dt_stump_err_train = 1.0 - dt_stump.score(X_train, y_train)
dt_stump_err = 1.0 - dt_stump.score(X_test, y_test)

dt = DecisionTreeClassifier(max_depth=8, min_samples_leaf=3)
dt.fit(X_train, y_train)
dt_err_train = 1.0 - dt.score(X_train, y_train)
dt_err = 1.0 - dt.score(X_test, y_test)
```

In [157]:
```
print(dt.tree_.node_count)
```

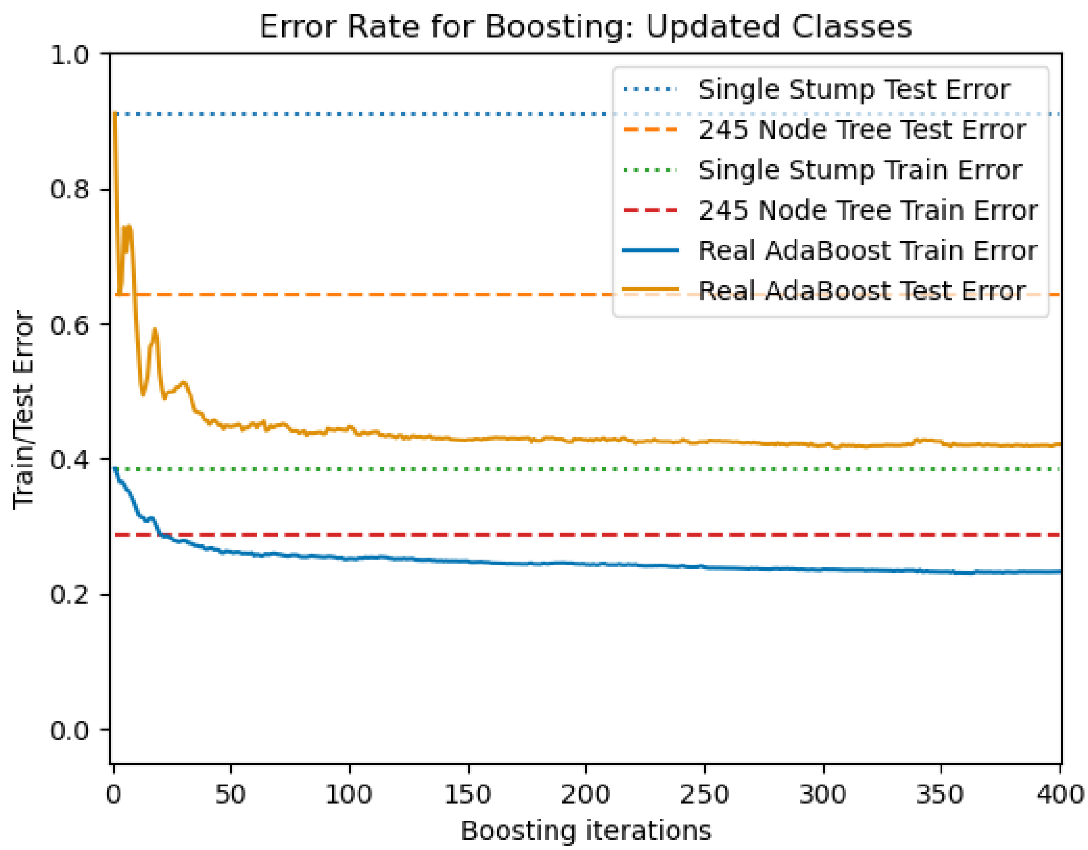
```
245
```

In [158]:
```
N_ESTIMATORS = 400
LR = 1.0
ada_real = AdaBoostClassifier(
    base_estimator=dt_stump,
    learning_rate=LR,
    n_estimators=N_ESTIMATORS,
    algorithm="SAMME.R",
)
ada_real.fit(X_train, y_train)
```

Out[158]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   n_estimators=400)
```

In [159]:
```
ada_real_err_train = np.zeros((N_ESTIMATORS,))
for i, y_pred in enumerate(ada_real.staged_predict(X_train)):
    ada_real_err_train[i] = zero_one_loss(y_pred, y_train)

ada_real_err = np.zeros((N_ESTIMATORS,))
for i, y_pred in enumerate(ada_real.staged_predict(X_test)):
    ada_real_err[i] = zero_one_loss(y_pred, y_test)
```

In [168]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

colors = sns.color_palette("colorblind")

ax.plot([1, N_ESTIMATORS], [dt_stump_err] * 2, ":",
        label="Single Stump Test Error")
ax.plot([1, N_ESTIMATORS], [dt_err] * 2, "--",
        label="245 Node Tree Test Error")

ax.plot([1, N_ESTIMATORS], [dt_stump_err_train] * 2, ":",
        label="Single Stump Train Error")
ax.plot([1, N_ESTIMATORS], [dt_err_train] * 2, "--",
        label="245 Node Tree Train Error")


ax.plot(
    np.arange(N_ESTIMATORS) + 1,
    ada_real_err_train,
    label="Real AdaBoost Train Error",
    color=colors[0],
)

ax.plot(
    np.arange(N_ESTIMATORS) + 1,
    ada_real_err,
    label="Real AdaBoost Test Error",
    color=colors[1],
)

ax.set_ylim((-0.05, 1))
ax.set_xlim((-1, 401))
ax.set_xlabel("Boosting iterations")
ax.set_ylabel("Train/Test Error")
ax.set_title("Error Rate for Boosting: Updated Classes")

leg = ax.legend(loc="upper right", fancybox=True)
leg.get_frame().set_alpha(0.7)

plt.show()
```




```
In [169]: print("The training error for Adaboost after 400 iterations is %.3f \n\
          The testing error for Adaboost after 400 iterations is %.3f \n\
          The training error for single stump is %.3f \n\
          The testing error for single stump is %.3f \n\
          The training error for 245 Node decision tree is %.3f \n\
          The testing error for 245 Node decision tree is %.3f" %
          (ada_real_err_train[-1], ada_real_err[-1],
           dt_stump_err_train, dt_stump_err,
           dt_err_train, dt_err))
```

```
The training error for Adaboost after 400 iterations is 0.232
The testing error for Adaboost after 400 iterations is 0.420
The training error for single stump is 0.385
The testing error for single stump is 0.910
The training error for 245 Node decision tree is 0.288
The testing error for 245 Node decision tree is 0.643
```

- Applying the single binary classification tree (single stump) gives a even poorer test set error rate (~0.910), compared to 50% for a random classifier. The single large classification tree also gives a poor test set error rate (0.643). This means they are not good at differentiating classes when they are too similiar

- As we increase the boosting iterations, the error rate steadily decreases, reaching 23% on training set after 400 iterations, and 42% on testing set after 400 iterations. The performance is much poorer than the previous example., but still better than the other two.
- The poor performance may due the fact that the AdaBoost is very sensitive to noise. It is fitting a classification model (an additive model) to an exponential loss function. The exponential loss function is sensitive to outliers/label noise since the penalty is exponentiated and wrong labeled data would suffer a large loss/penalty if it is deeply inside the other class.

1.2 Q3

The "spam" data" (https://web.stanford.edu/hastie/ElemStatLearn/data) has been divided into a training set and a test set. Fit a neural network to the training set, and calculate its classification error on the test set. Compare your results to the classification tree results presented in Section 9.2.5 of [ESL] on both the classification performance and interpretability of the final model.

```
In [170]: ## Read in data
          X_train = pd.read_csv("X_train.csv", header = 0, index_col = 0)
          X_train = X_train.reset_index(drop=True)
          y_train = pd.read_csv("y_train.csv", header = 0, index_col = 0)
          y_train = y_train.reset_index(drop=True)

          X_test = pd.read_csv("X_test.csv", header = 0, index_col = 0)
          X_test = X_test.reset_index(drop=True)
          y_test = pd.read_csv("y_test.csv", header = 0, index_col = 0)
          y_test = y_test.reset_index(drop=True)
```

1.3 Use Keras to build a sequential model of 3 hidden layers, 2 dropout layers

Taking 20% data out as our validation set

```
In [171]: # validation set
          X_train_, X_val = train_test_split(X_train, test_size=0.2, random_state = 42)
          y_train_, y_val = train_test_split(y_train, test_size=0.2, random_state = 42)
```

```
In [172]: # create model
          METRICS = [
                keras.metrics.BinaryAccuracy(name='accuracy'),
                keras.metrics.AUC(name='auc')
          ]

          model = Sequential([
              Dense(256, input_shape=(X_train.shape[1],), activation="relu"),
              BatchNormalization(),
              Dropout(0.4),
```

```
            Dense(128, activation='relu'),
            BatchNormalization(),
            Dropout(0.4),
            Dense(64, activation='relu'),
            BatchNormalization(),
            Dropout(0.4),
            Dense(1, activation='sigmoid')
        ]);

        # Compile model
        model.compile(optimizer = keras.optimizers.Adam(learning_rate=0.0001),
                      loss = keras.losses.BinaryCrossentropy(),
                      metrics=METRICS)
        model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               14848
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 57,857
Trainable params: 56,961
Non-trainable params: 896
_________________________________________________________________
```

```
2022-11-07 02:33:28.010129: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFl
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

```
In [173]: EPOCHS = 100
          BATCH_SIZE = 40

          early_stopping = tf.keras.callbacks.EarlyStopping(
              monitor='auc',
              verbose=2,
              patience=20,
              mode='max',
              restore_best_weights=True)

          history = model.fit(
              X_train_,
              y_train_,
              epochs=200,
              validation_split=0.20,
              callbacks=[early_stopping],
              batch_size=40,
              verbose=0,
              validation_data=(X_val, y_val)
          )
```

```
2022-11-07 02:33:28.226653: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] No
```

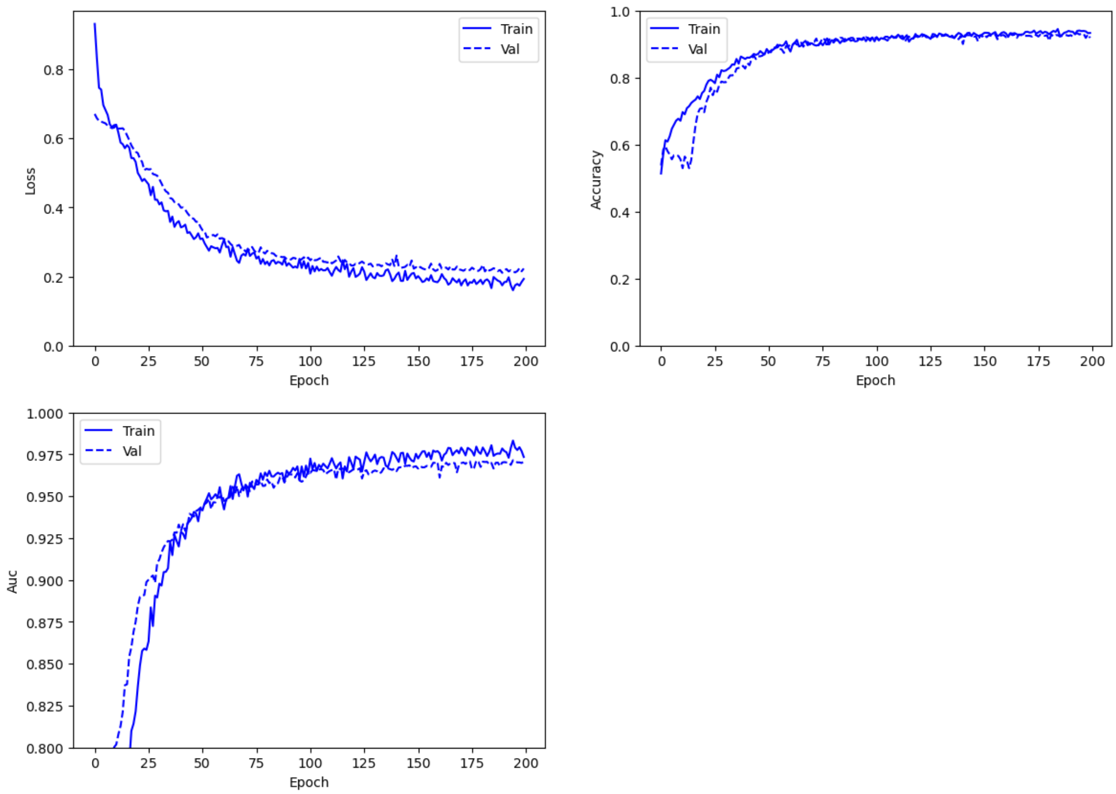
```
In [174]: results = model.evaluate(X_train, y_train, batch_size=BATCH_SIZE, verbose=0)
          print("Loss: {:0.4f}".format(results[0]))
```

```
Loss: 0.1754
```

1.3.1 History Checking

```
In [175]: def plot_metrics(history):
              metrics = ['loss', 'accuracy', 'auc']
              for n, metric in enumerate(metrics):
                  name = metric.replace("_"," ").capitalize()
                  plt.subplot(2,2,n+1)
                  plt.plot(history.epoch, history.history[metric],
                           color='b', label='Train')
                  plt.plot(history.epoch, history.history['val_'+metric],
                           color='b', linestyle="--", label='Val')
                  plt.xlabel('Epoch')
                  plt.ylabel(name)
                  if metric == 'loss':
                      plt.ylim([0, plt.ylim()[1]])
                  elif metric == 'auc':
                      plt.ylim([0.8,1])
                  else:
                      plt.ylim([0,1])
```

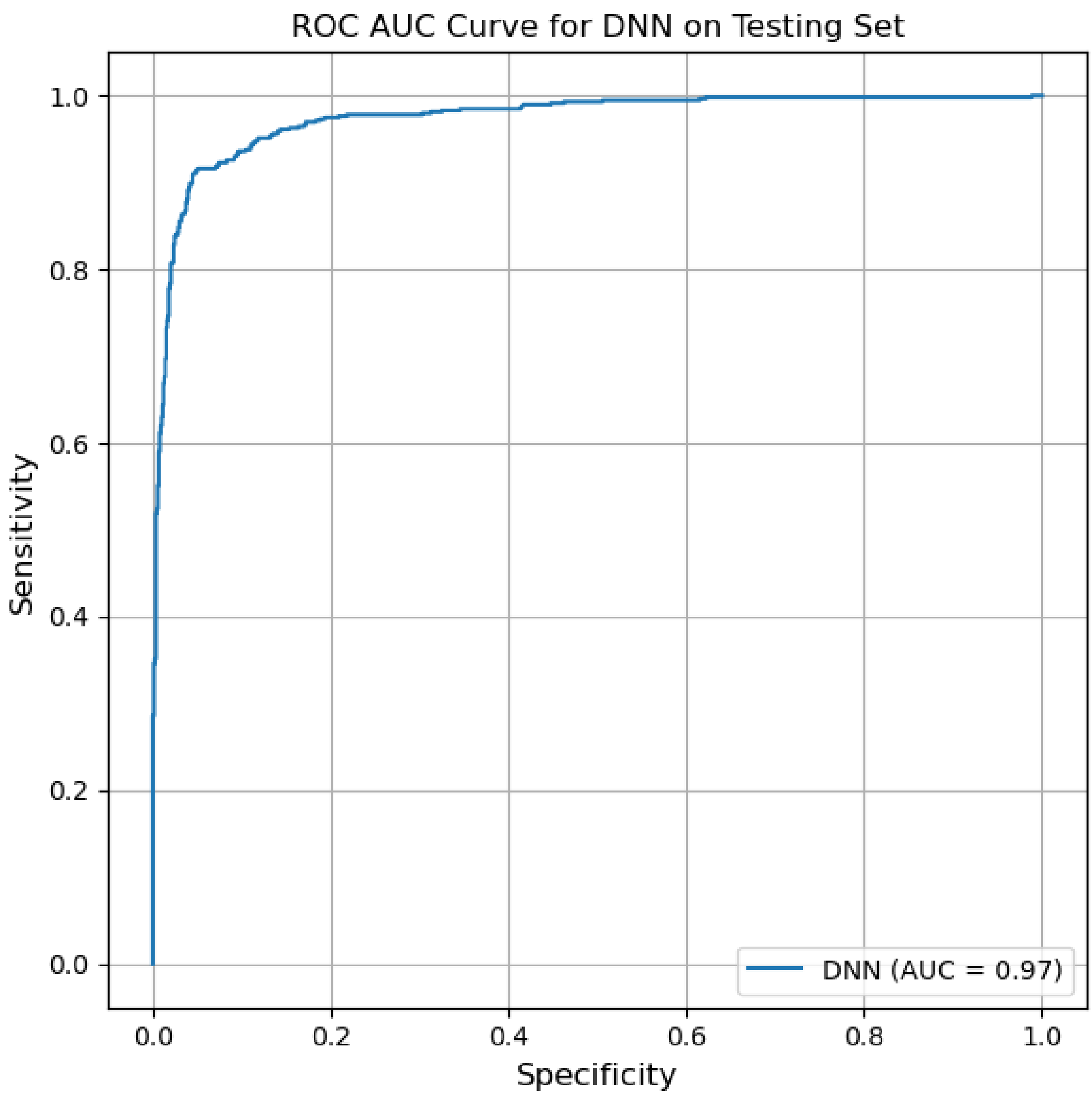
```
            plt.legend()

    plt.rcParams['figure.figsize'] = (14,10)
    plot_metrics(history)
```



As the number of Epoch grows, the training and validation error decreases and the ROC AUC increases. Our training is indeed effective.

Then we can use the model to predict the test set:

```
In [176]: test_pred = model.predict(X_test, batch_size=BATCH_SIZE)
          plt.rcParams['figure.figsize'] = (6,6)
          RocCurveDisplay.from_predictions(y_test, test_pred, name = 'DNN')
          plt.title("ROC AUC Curve for DNN on Testing Set")
          plt.xlabel("Specificity", fontsize = 12)
          plt.ylabel("Sensitivity", fontsize = 12)
          plt.xticks(fontsize=10)
          plt.yticks(fontsize=10)
          plt.grid()
          plt.tight_layout()
          plt.show()
```



Calculate classification error:

```
In [177]: # RSS
          test_error = np.mean((np.array(y_test)-test_pred)**2)
          ## sqrt(RSS/(n-2))
          test_error_sd = np.mean(
              np.sqrt((np.array(y_test)-test_pred)**2/(X_test.shape[0]-2)))
          print("The classification error on the test set is %.3f \
          and the standard deviation is %.3f" % (test_error, test_error_sd))
```

```
The classification error on the test set is 0.061 and the standard deviation is 0.003
```

- The classification error on the test set of the DNN model (0.061 ± 0.003) is better than that of

the classification tree's, and it also gives a better classification rule for any loss with an area of 0.97 (0.97 > 0.95).
- However, the as shown on Fig. 9.5, the pruned tree from the classification tree result can be easily interpreted. The DNN model is black-boxed, therefore it is lack of interpretability compared to the classification tree.