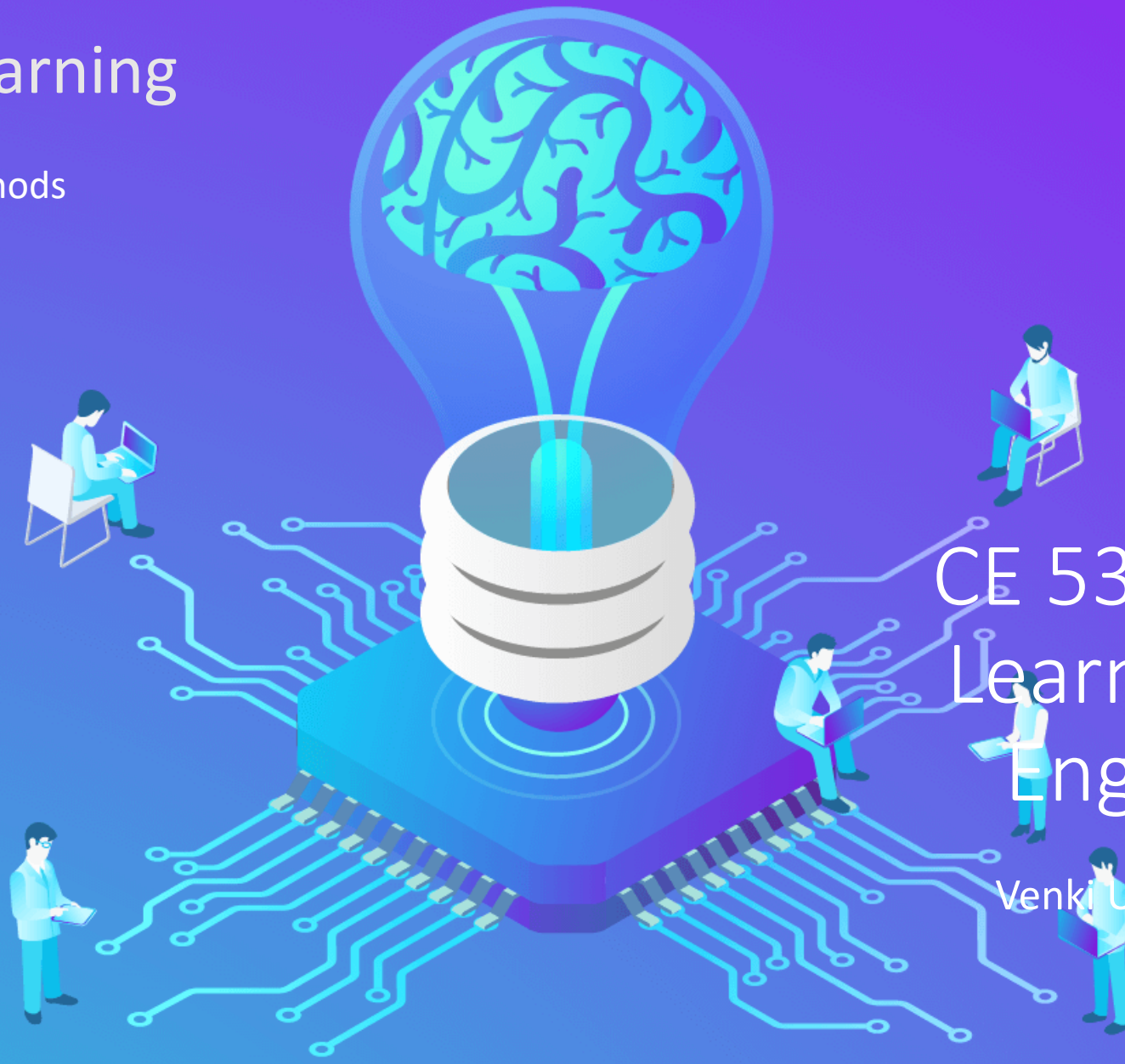
Machine Learning

Boosting Methods



CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap

- What is Machine Learning
- How is it useful for Civil Engineers
- Overview of Machine Learning Methods
- Linear Regression
  - Bivariate
  - Regression interpretation
  - Multivariate
- Logistic Regression
  - Maximum likelihood estimation
  - Regularization (introduction)
- Naïve Bayesian Classifier
  - What is it
  - What makes it naïve
  - Bayes theorem
  - Prior, likelihood and posterior
- K-Nearest Neighbor
  - How does the algorithm work
  - Why is it a lazy learner
  - How to do regression and classification
- Introduction to Decision Trees
  - Fundamentals
  - Information Gain, Entropy and Gini Index
  - ID3 algorithm
  - Classification and Regression Trees (CART)
  - Multi-Adaptive Regression Splines (MARS)

- Ensemble learners
  - Introduction
- Their benefits and drawbacks
- Simple (voting) ensemble learners
- Bagging and Pasting
- Generic bagging classifiers
- Random Forests (RF)
- Variable Importance using RF

Python – Introduction
Python – Functions
Python - Pandas
Python – np, scipy, statsmodels
Python – Scikit learn – linear, metrics
Python – Matplotlib, seaborn
Python – Mixed_Naive_Bayes
Python – scikit learn neighbors module
Python – scikit learn ensemble voting
Python – scikit learn bagging classifier
Python – scikit learn Random Forest

R – Classification and Regression Trees using rpart
R – Drawing trees using rpart.plot
R - Multiadaptive Regression Splines (MARS) using Earth Algorithm

Boosting Methods

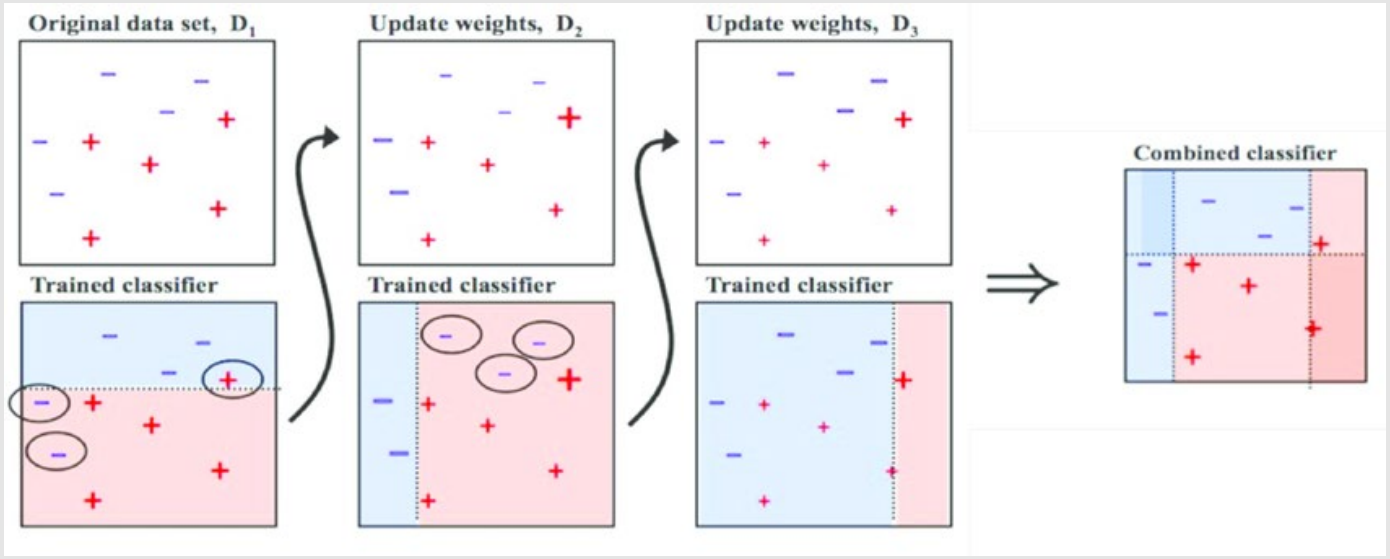
Boosting

- Boosting was originally referred to as Hypothesis Boosting
- Boosting is a technique where an ensemble of weak learners are used in sequence
  - A weak learner (e.g., a small tree) is fit to the data
  - The underfitted or residual part is fit to another model
  - The process is continued till predictions are satisfactory
- Unlike Bagging – Boosting is not parallel
  - Models are fit sequentially
- The output from a boosting model is the sum of predictions for all individual models (in case of regression)
- In case of classification – A weighted majority vote is taken

Boosting Algorithms

- Two commonly used boosting algorithms are:
  - AdaBoost
    - Adaptive Boosting
  - Gradient Boosting Trees (GBT)
- There are many other Boosting Algorithms for both classification and Regression
  - See - Ferreira, Artur J., and Mário AT Figueiredo. "Boosting algorithms: A review of methods, theory, and applications." In *Ensemble machine learning*, pp. 35-85. Springer, Boston, MA, 2012.

AdaBoost



- AdaBoost works on the philosophy that
  - Use a simple learner initially
    - Predict the Low hanging fruits
  - Pay a bit more attention to those records (training instances) that the predecessor underfitted
    - Focus more and more on harder to predict data
- AdaBoost Algorithm first trains a base classifier
  - Usually a decision tree
  - All training instances have equal weight
- Misclassified or Underfitted instances are given higher weight
  - Another classifier is fit to predict these values
- The process is continued

AdaBoost Algorithm

$$W^{(i)} = \frac{1}{N}$$

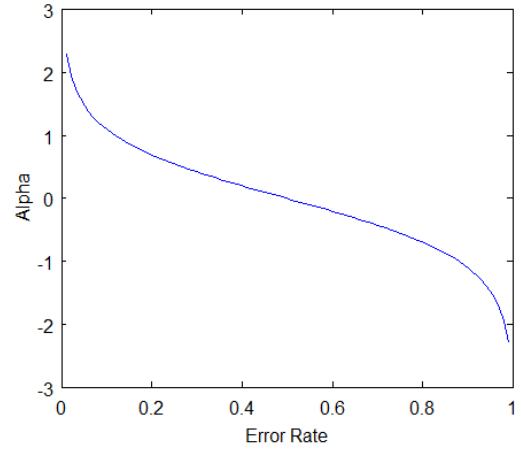
Initial Weight of Each Instance → Fit a classifier h(x)

- Each training instance is given an equal weight
  - Train a simple classifier
  - Just better than random sampling
    - A stump of a decision tree
- Calculate the weight of the classifier
  - Classification error rate

Misclassification Rate (# of Misclassification/Total # of classifications)

Weight of each classifier

$$\alpha_t = 0.5 ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$



Updated Weight of each Instance

Old Wt.

Scaling term

$$D_{t+1}(i) = \frac{D_t(i) exp[-\alpha_t y_i h_t(x_i))]}{Z_t}$$

New Weight

Sum of weights D (normalize to unity)

Fit a classifier h(x)

Final Prediction

$$H(x) = \sum_{t=1}^{T} (\alpha_t h_t(x))$$

AdaBoost and SAMME

- SAMME stands for Stagewise Additive Modeling using a Multiclass Exponential Loss function
  - SAMMA is a general form of AdaBoost
- SAMME is the same of AdaBoost for 2 class problems
- Scikit learn uses SAMME.R (where R stands for real)
  - Generally performs better than SAMME
  - Uses class probabilities instead of predictions
  - Default in scikit.learn library

Gradient Boosting

- Gradient Boosting also adds predictors sequentially to an ensemble
  - Instead of tweeking weights of training instances a new predictor is fit to the residual

$$F_{m+1}(X) = F_m(X) + h(X) = y$$

$$h(X) = y - F_m(X)$$

Algorithm 1 *Gradient boosting for binary classification using regression trees as base classifiers.*

1: Set the initial value, $f_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2: $t = 0$
repeat
3: $t = t + 1$
4: Compute the components of negative gradient $r_{it} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right], i = 1, ..., n$
5: Fit a regression tree to targets $r_{it}, i = 1, ..., n$ giving terminal regions $R_{jt}, j = 1, ..., J_t$.
6: Compute the updates to the current response,

$$\gamma_{jt} = \arg\min_\gamma \sum_{x_i \in R_{jt}} L(y_i, f_{t-1}(\mathbf{x}_i) + \gamma), j = 1, ..., J_t.$$

7: Update $f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \sum_{j=1}^{J_t} \gamma_{jt} I(\mathbf{x} \in R_{jt})$
until $t = T$
8: Output

$$\hat{f}(\mathbf{x}) = f_T(\mathbf{x})$$

Implementation in Python

- Scikit learn ensemble module has
  - AdaBoostClassifier
  - AdaBoostRegressor
  - GradientBoostingClassifier
  - GradientBoostingRegressor
- Can be used to easily fit Boosting tree algorithms
- These models have the following hyperparameters
  - ‘n_estimators’ — Number of trees
  - ‘learning_rate’ — Scales the contribution of each tree (small values ~ 0.1 require more trees but generalize better)

Hyperparameters for classification trees (max_depth) must also be specified

Unlike Bagging we use shallow trees so max_depth is small 1 or 2

Illustrative Example

- Predicting damage to culverts in Texas
- Create a Random Forest Classification Model to predict satisfactory and unsatisfactory states
- Use Gridsearch to identify optimal parameters for ‘max_depth’ and ‘n_estimators’

Satisfactory

Unsatisfactory

Code	Meaning	Description
9	Excellent	As new
8	Very Good	No problems noted.
7	Good	Some minor problems.
6	Satisfactory	Structural elements show some minor deterioration.
5	Fair	All primary structural elements are sound but may have minor section loss, cracking, spalling or scour.
4	Poor	Advanced section loss, deterioration, spalling or scour.
3	Serious	Loss of section, deterioration, spalling or scour has seriously affected primary structural components. Local failures are possible. Fatigue cracks in steel or shear cracks in concrete may be present.
2	Critical	Advanced deterioration of primary structural elements. Fatigue cracks in steel or shear cracks in concrete may be present or scour may have removed substructure support. Unless closely monitored it may be necessary to close the bridge until corrective action is taken.
1	Imminent Failure	Major deterioration or section loss present in critical structural components or obvious vertical or horizontal movement affecting structure stability. Bridge is closed to traffic but with corrective action may put back in light service.
0	Failed	Out of service, beyond corrective action.

Source: United States Department of Transportation. Recording and Coding Guide for the Structure Inventory and Appraisal of the Nation's Bridges. Washington, D.C., 1995, page 38.

Python Code

```
#Step 1: Load Libraries
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics
import seaborn as sns
from matplotlib import pyplot as plt
```

```
# Step 2: Change working directory
dir = 'D:\\Dropbox\\000CE5333Machine
Learning\\Week7EnsembleLEarning\\Code'
os.chdir(dir)

# Step 3: Change working directory
dir = 'D:\\Dropbox\\000CE5333Machine
Learning\\Week7EnsembleLEarning\\Code'
os.chdir(dir)
```

```
# Step 4: Read the dataset
a = pd.read_csv('TXculvertdata.csv') # read our dataset
features = ['SVCYR','ADT','Reconst','PTRUCK'] # INPUT DATA FEATURES
X = a[features] # DATAFRAME OF INPUT FEATURES
SVCYR2 = a['SVCYR']**2  # Add SVCYR square to the dataset
X['SVCYR2'] = SVCYR2  # CALCULATE THE SQUARE OF AGE
Y = a['Culvert_Damage'] # ADD IT TO THE INPUT FEATURE DATAFRAME

# Step 5: Split into training and testing data
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.30,
                                    random_state=10)
```

```
# Step 6: Perform Grid Search for Best Hyperparameters using 5-fold CV
# Assumes the base classifier to be a stump with max_depth 1
gsc = GridSearchCV(
      estimator=AdaBoostClassifier(),
      param_grid={'learning_rate': np.arange(0.1,1.0,0.1),
                  'n_estimators': (10, 50, 100, 200, 500),},cv=5,
                  n_jobs=-1)
grid_result = gsc.fit(X_train, y_train)
best_params = grid_result.best_params_
```

Python Code

```
# Step 7: Train the best AdaBoost Model
adab = AdaBoostClassifier(n_estimators=best_params['n_estimators'],
                          learning_rate=best_params['learning_rate'])
adab.fit(X_train,y_train)
```

```
# Step 8: Create a confusion Matrix
y_pred=adab.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix # y_test is going be rows (obs), y_pred (predicted) are cols
```

```
# Step 9: Evaluate usng accuracy, precision, recall
print("Accuracy:",metrics.accuracy_score(y_test, y_pred)) # overall accuracy
print("Precision:",metrics.precision_score(y_test, y_pred)) # predicting 0 (Sat)
print("Recall:",metrics.recall_score(y_test, y_pred)) # predicting 1 (unsat)
```

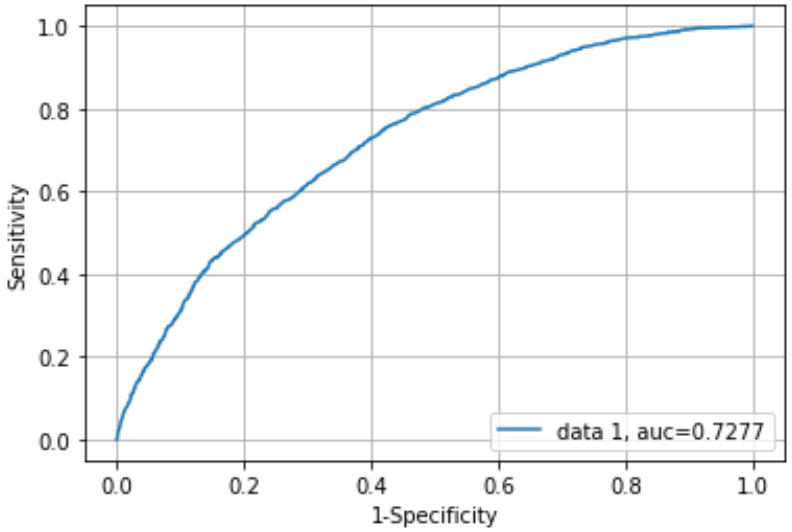
```
# Step 10: Plot ROC Curve
y_pred_proba = adab.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(round(auc,4)))
plt.legend(loc=4)
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.grid()
plt.show()
```

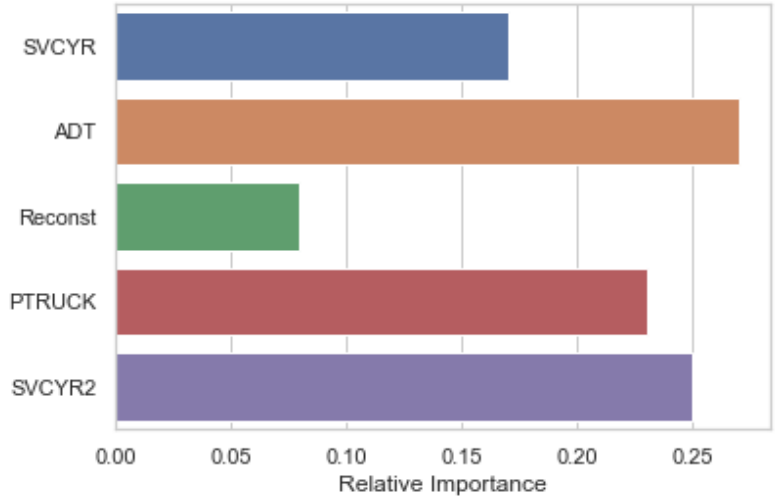
```
# Step 11: Determine Feature Importance
var = list(X.columns)
imp = adab.feature_importances_
sns.set(style="whitegrid")
ax = sns.barplot(x=imp, y=var) # Make a barplot
ax.set(xlabel="Relative Importance")
```

Results

Random Forest – Confusion Matrix

	Predicted	
Obs	0	1
0	2756	749
1	1209	1272





AdaBoost

Accuracy: 0.673
Precision: 0.629
Recall: 0.512

Comparison to Other Models

Model	Accuracy
Logistic	0.6657
Naïve Bayes	0.6557
KNN	0.6236
Ensemble	0.6676
Random Forest	0.676

Results generally comparable to RF except RF has a better Recall
The ADT and SVCYR2 are two most important variables but PTRUCK is also significant
Variable Importance a little different than RF Algorithm (RF algorithm mostly focused on SVCYR

Some Considerations

- Boosting trees take time to train
  - Not fully parallel
- Boosting trees can overfit the data
  - Reduce the number of estimators
  - Some estimators simply learn noise instead of the signal
- Boosting trees are sensitive to outliers
  - Errors are corrected in subsequent steps so outliers play a major role
- Extreme Gradient Boosting (XGBoost) is a newer and better (optimized) implementation of Gradient Boosting
  - Check out xgboost package

You should know

- What is the idea behind boosting
- Difference between boosting and bagging
- Different Boosting algorithms
  - AdaBoost
  - Gradient Boosting
- How to implement in Python
  - Left as an exercise for now