GENERAL ASSEMBLY

DAT SCIENCE

CLASS 10: ENSEMBLE METHODS AND NEURAL NETWORKS

I. ENSEMBLE METHODS OVERVIEW
II. RANDOM FORESTS
III. BOOSTING TREES
IV. NEURAL NETWORKS

I. ENSEMBLE METHODS OVERVIEW

WHAT IS AN ENSEMBLE METHOD?

- Ensemble methods are a set of machine learning algorithms that combine the result of multiple ‘base’ estimators (such as linear regressions or decision trees) to improve their overall accuracy.

- Ensemble methods use what I call ‘modern’ machine learning: using the recent exponential increase in computing power to develop models using complex rules no human could create in a reasonable amount of time.
    - The good of this: higher accuracy than traditional methods
    - The bad: lower interpretability of the models themselves.

- Ensemble methods grew out of Leo Brieman’s seminal work *Statistical Modeling: The Two Cultures,* which heavily criticized the existing statistical culture as being too fixated on theory at the expense of predictive accuracy.
    - Link: http://projecteuclid.org/euclid.ss/1009213726

- Breiman's major insight was the following: that traditional models are too focused on trying to directly **fit** models to what statisticians believe is the natural interaction of data.

- The problem with linear regression (and its related processes) is that we try to shoehorn linearity (and extensions thereof) on natural processes and interactions that are no way neatly linear nor described well using algebraic methods.

- Instead, Breiman proposed that modeling should simply do all it can to **predict** the response feature, and not care about whether the model itself directly represents interactions in nature (as we'll likely never understand nor interpret them fully!).

WHAT IS AN ENSEMBLE METHOD?

- Ensemble methods can be divided into two main categories:
  - **Averaging methods** build numerous (sometimes 1,000+) estimators **independently** and average their prediction.
    - Examples: Random Forests, Bagging Methods
  - **Boosting Methods** builds numerous (usually less than 1,000, but sometimes can be larger) estimators **sequentially** with the goal of improving overall accuracy with each new added estimator.
    - Examples: Boosting Trees, AdaBoost
- The most common machine learning methods in use today are Random Forests, Boosting Trees, and Deep Neural Networks (to be discussed later in this lecture).

II. RANDOM FORESTS

- A Random Forests algorithm produces many independent decision trees, and at the time of classification or regression, outputs either the mode class or mean value produced by its component trees.

- Random forests are either the most popular or second-to-most popular machine learning method in use today.

- Recall the way a decision tree estimator works:
    1. Calculate the Gini purity of the data
    2. Select a feature candidate split
    3. Calculate the purity of the data after the split
    4. Repeat for all features
    5. Choose the feature with the greatest decrease in purity and add it to the tree
    6. Repeat for each split until some stop criteria is met

- Random Forests extend this technique by building its trees in a sample of the data with replacement.

- In addition, instead of performing the above process for all features, it is done for a random subset of features.

- Why do we grow the trees on a sample of the data? So that we can help them ignore outliers and better generalize over the entire dataset (ie create lower variance). As the trees in full cover the entire dataset, this method provides no additional loss in accuracy (bias).

- Once all the trees have been built out, the Random Forest decides a classification problem by taking the majority (or plurality) vote of all the trees, and a regression problem by taking the average value of all the trees.

- In classification problems, probabilities are also calculated as the average of all the trees.

- Random Forests are usually tuned via two parameters: (1) the number of trees in the forest; and (2) the number of features in the random subset of data.
    - Although the correct tuning parameters will differ for each model, the defaults set are usually (1) 500 trees; and (2) either (a) all the features (for regression) or the square root of the features (for classification).
    - Max_depth typically is best tuned at None (ie we should use all the features), and min_sample_split is best tuned at 1 (ie a leaf can include just 1 observation).
    - You should use cross-validated grid search to determine the best tuning parameter for your model, although in many cases, the differences from the default parameters may be minute.

HOW DO YOU CALCULATE FEATURE IMPORTANCES?

- Much like decision trees, feature importances are calculated by the hierarchy of each feature in the forest's component decision trees.

- The calculation of importance itself is the **average fraction** of the data the feature splits – so, for example, the highest feature in your hierarchy has a fraction of 1, while all others below it have either a similar or lower fraction.

- The advantages of Random Forests are:
  - Easily understandable (although not interpretable)
  - Robust to irrelevant data and do not require feature scaling
  - Random forests usually do not over-fit as you add additional trees.
  - Highly accurate compared to single trees and linear methods.

- The drawbacks of random forests are:
  - The overall model is not directly interpretable (although you can test by altering input data)
  - They have a high computing overhead, especially as you add more trees
  - If you tune your models maximally, they are sometimes less predictive than boosting trees or neural nets.
  - Class probabilities are usually not directly calibrated to out-of-sample-data, although their sorting is usually correct.

III. BOOSTING TREES

- Boosting Trees employ a process popularized by AdaBoost to grow regression and classification trees:

1. A single decision tree is built and predicted on the data.
2. A weight (**‘boost’**) is given for each observation based on either its correct or incorrect classification, or its absolute error from the predicted value.
3. A second decision tree is built with the updated weights, in effect skewing its composition to concentrate on the observations that are most dificult to predict.
4. Steps 2 and 3 are repeated until you reach a pre-determined number of trees or fail to improve accuracy above a certain metric.

- Boosting Trees differ from AdaBoost in that you can customize your loss function that generates the weights given to training observations.

- The standard loss function for classification is ‘deviance’, i.e. ability / inability to appropriately calculate the observation’s class.

- The standard loss function for regression is ‘least squares’, i.e. the squared error of the observation from its corresponding predicted value.

- Other loss functions include least absolute deviation, the root squared error of the observations, and quantiles, i.e. the ability of each tree to properly calibrate the data via prediction intervals.

- Stochastic Gradient Boosting extends boosting trees by employing the sampling methods used in Random Forests.

- By sub-sampling the data, you help prevent overfitting of the AdaBoost process.

- However, the number of iterations needed to find an optimal result is usually far larger than more traditional gradient boosting methods.

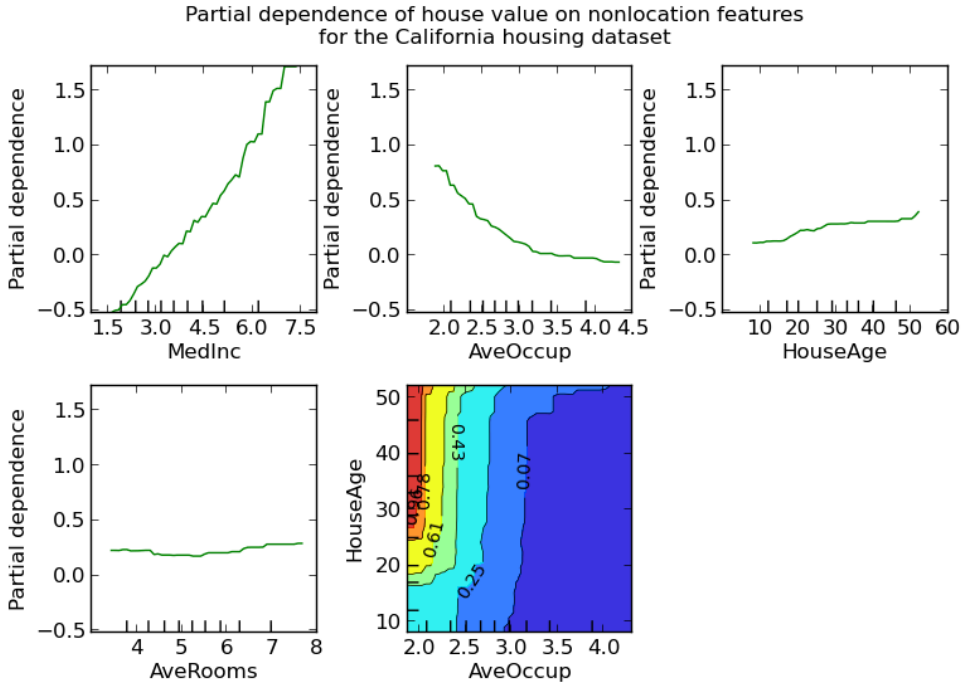
- Boosting Trees employ a large number of tuning parameters (including the parameters used in decision trees and random forests), but a few are very important:
  - **Learning rate** is the relative weight of the output produced by each tree iteration in the final vote (for classification) or averaging (for regression).
    - A smaller learning rate will usually provide higher in-sample accuracy at the expense of overfitting on out-of sample data. You can counter this by increasing the number of trees in the model, but this increases its computational overhead.
    - A typical learning rate is 0.05.

  - **Subsampling** is the percentage of the overall data each tree is trained on. A typical subsampling rate is 0.5.

FEATURE IMPORTANCES & PARTIAL DEPENDENCE

- Boosting feature importances work like that of random forests: it's the average importance of each tree.

- Partial dependence plots shows you the interaction between each independent feature and your response, and is helpful to determining the linearity of the interaction of the two features.



- Pros:
  - If tuned correctly, Boosting Trees typically outperform Random Forests on most classification and regression problems.
  - Interpretability of tree-based methods
  - Do not require feature scaling
  - Custom loss functions can be extremely helpful when maximizing predictive accuracy of a minority class.

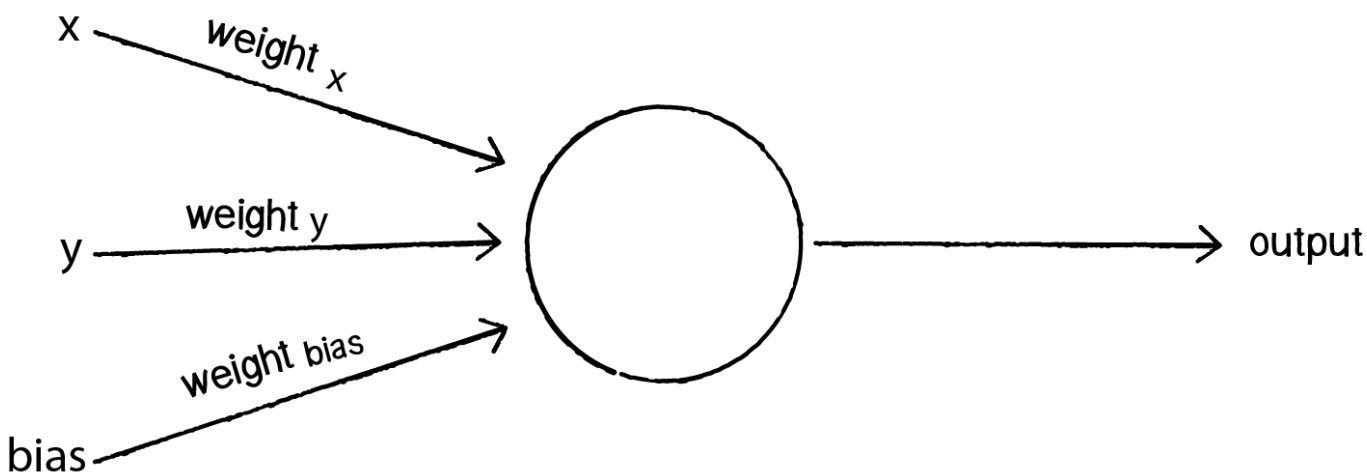
- Cons:
  - Easy to overfit on data
  - High computational overhead
    - The process is by nature sequential
    - Proper tuning parameters are harder to ‘guess’
  - Not well-suited for multi-class classification (use Random Forests instead)

- Let's return to your Hall of Fame dataset and do the following:
  - Fit a cross-validated Random Forest with 500 trees on your data.
  - Evaluate ROC AUC performance vs. the decision tree estimator you already built.
  - Use grid search to find the best parameters for n-estimators.
  - Plot out-of-sample accuracy by number of trees in the forest.
  - Fit a cross-validated Boosting Tree classifier on the data and compare accuracy to your other models.
  - Print out of sample accuracy. Compare it to random forest accuracy.
  - Tune the parameters for n_estimators, the learning rate, and subsampling percentage.
  - Plot ROC curve accuracy of your most predictive Boosting model vs the most predictive Random Forest model.
  - Create partial dependence plots for your Boosting model.

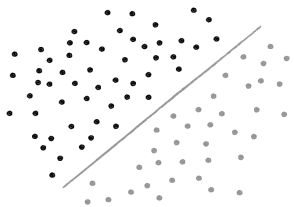
IV. NEURAL NETWORKS

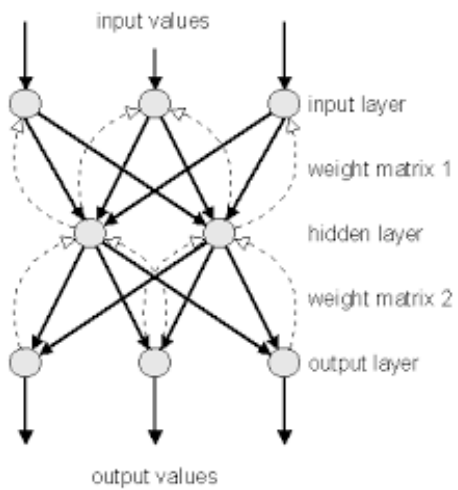
- Neural Networks attempt to duplicate the behavior of a human brain through the use of discrete units called **perceptrons**.

- Essentially, a perceptron optimizes weights of your input features plus a constant (called a bias) in an iterative process similar to a linear or logistic regression to maximize its predictive 'fit' (in-sample accuracy) on the data.



- Much like a linear regression, a perceptron can only solve **linearly separable** problems.
- However, if you hook the outputs of one perceptron to the inputs of another, you can begin to solve for nonlinear problems.
- Each ‘layer’ of neurons accepts the input of the previous layer, and produces an output.
- Weights of initial and subsequent layers are updated by **backpropagation,** where error is minimized by altering weights at each step of the network.

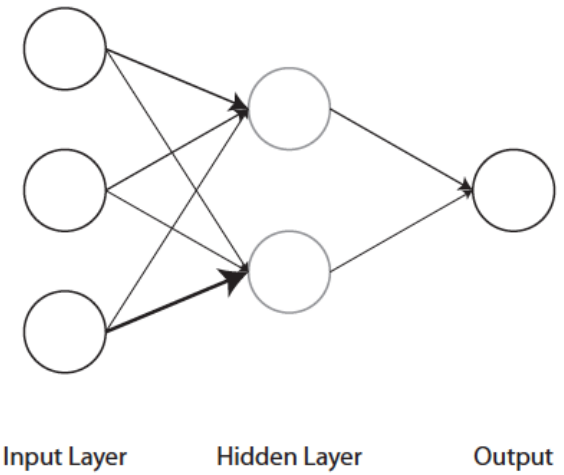




- Much like boosting trees and random forests, only a subset of data is exposed to the network at a time to avoid overfitting.

- At the end of the network is a single perceptron that takes all the last layer's results and then weights them to maximize accuracy.

- Ultimately, you can think of this as a **weighted vote** or **weighted average** depending on whether you are addressing a classification or regression problem.



- Scikit-learn implements neural networks via a two-step process: (1) the network itself; and (2) the final aggregating perceptron (or logistic regression or support vector machine).

- These can be combined using a pipeline in scikit-learn.

- The parameters used in neural networks are:
  - the number of ‘hidden’ units (usually set to ~500) or layers (usually set to 10)
  - The ‘learning rate’ or cutoff at which weight adjustments stop (ranges from 0 to 1; most start at 0.01 but this needs to be tuned via grid search).
  - The number of iterations over the training dataset to build the model (recommendation is 10).

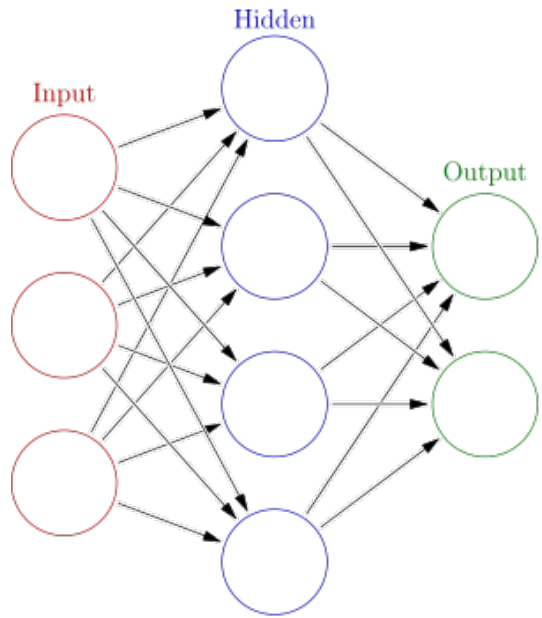
- Unfortunately, the implementation does not maximize predictive accuracy in the creation of the network as the only currently available network models use unsupervised methods.

- However, the use of the final perceptron or logistic regression does allow us to have predictive accuracy that sometimes rivals or beets boosting trees or random forests.

- Pylearn2 and Theanets have direct implementation, but use language and syntax outside the scope of this class.

- Pros:
  - Highest predictive accuracy currently for most non-linear problems.
  - Avoids the ‘step-like’ prediction thresholds of most tree-based methods.

- Cons:
  - Limited implementation in scikit-learn
  - Very high computational overhead
  - Easy to overfit / hard to tune
  - Slow prediction on large amounts of data
  - Irrelevant features can throw off accuracy



- Let's return to your Hall of Fame dataset and do the following:
  - Create a data pipeline that combines an unsupervised neural network with a logistic regression (really, you should be combining the neural net with a support vector machine, which we will learn in the next class).
  - Tune the parameters for learning rate, hidden units, and number of iterations.
  - Plot ROC curve accuracy of your model and compare to your Boosting, Random Forest, and Decision Tree models.

QUESTIONS?