Machine Learning & Predictive Analytics

Class 4

Arnab Bose, Ph.D.

MSc Analytics

University of Chicago

Boosting

1. Trees are grown sequentially using information from previous trees.
2. Adaptive boosting (AdaBoost) –
    1. Each training data instance is initially weighted the same.
    2. A classifier is built to make predictions on the training data.
    3. The relative weight of the misclassified training data is increased.
    4. A second classifier is trained using the updated weights to make predictions (again) on the training data.
3. The technique has similarities with Gradient Descent except that instead of tweaking a single predictor's parameters to minimize a cost function, AdaBoost adds predictors to the ensemble – it is a coordinate descent algorithm. Coordinate descent is just like gradient descent, except that you can't move along the gradient, you have to choose just one coordinate at a time to move along.
4. Gradient Boosting – fits new model to the residual errors of the previous model.
5. Stochastic Gradient Boosting – use subsample hyperparameter to use a mini batch of sample to train each tree. This increases bias and lower variance and speeds up training.

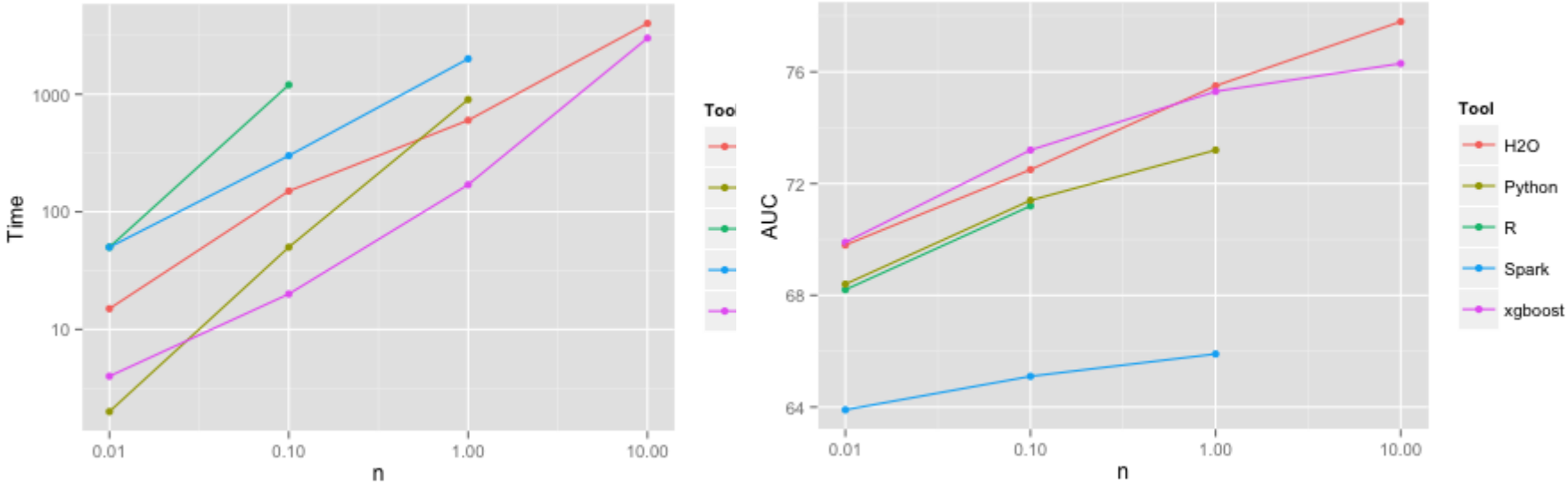
Gradient Boosting Classifier – Example

#	Age	Gender	Active	Chronic	Asymptomatic	Residual 1	Residual 2 (LR = 0.1)	Residual 2 (LR = 0.3)
1	23	M	Y	Y	0	- 0.7	-0.65	-0.60
2	71	M	N	Y	1	0.3	0.3	0.24
3	48	F	Y	N	1	0.3	0.35	0.40
4	35	F	N	Y	1	0.3	0.3	0.24
5	68	M	N	N	1	0.3	0.35	0.40
6	62	F	N	N	0	- 0.7	-0.65	-0.60

Details in PDF in Class 4 > Reading Materials

XGBoost

1. eXtreme Gradient Boosting supports
    1. Gradient boosting – aka gradient boosting machine with learning rate
    2. Stochastic gradient boosting – subsample of training instances to be used for each tree
    3. Regularized gradient boosting – scale the contribution of each tree by a factor $0 < \nu < 1$
2. Execution speed and model performance (python package: xgboost).
3. Can handle missing values.



http://datascience.la/benchmarking-random-forest-implementations/

LightGBM

1. Uses histogram-based algorithms, which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage.
    1. max_bin: max number of bins that feature values will be bucketed in
    2. min_data_in_bin: minimal number of data inside one bin
    3. bin_construct_sample_cnt: number of data that sampled to construct histogram bins.

2. Optimal Split for Categorical Features - Particularly for high-cardinality categorical features - sort the categories according to the training objective at each split.
3. Sparse Optimization - Need only O(2 * #non_zero_data) to construct histogram for sparse features.
4. Optimization in Network Communication and in Parallel Learning.
5. Can handle missing values.

CatBoost

1. Has a special methodology that assigns indices to categorical columns to enable one-hot encoding.
2. Great quality without parameter tuning- Reduce time spent on parameter tuning, because CatBoost provides great results with default parameters.
3. Categorical features support - Improve your training results with CatBoost that allows you to use non-numeric factors, instead of having to pre-process your data or spend time and effort turning it to numbers.
4. Improved accuracy - Reduce overfitting when constructing your models with a novel gradient-boosting scheme.
5. Fast prediction - Apply your trained model quickly and efficiently even to latency-critical tasks using CatBoost's model applier.
6. Can handle missing values.

XGB vs CatBoost vs Light GBM

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Boosting Parameters

1. The number of trees B (n_estimators). Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. Use cross-validation to select B.

2. The shrinkage parameter λ (learning_rate), a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.

3. The number of splits d in each tree (max_depth), which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

4. Note – since boosting takes into account previous trees, smaller trees are typically sufficient.

Stacking

1. Aggregate individual model predictions in an ensemble using a model.
2. Second layer model that aggregates first layer output is called blender or meta learner.
3. Each stacked layer requires a training set.
4. For all models and layers, an implementation option is to use Scikit-Learn `Pipeline`.

Kernel Methods & SVMs

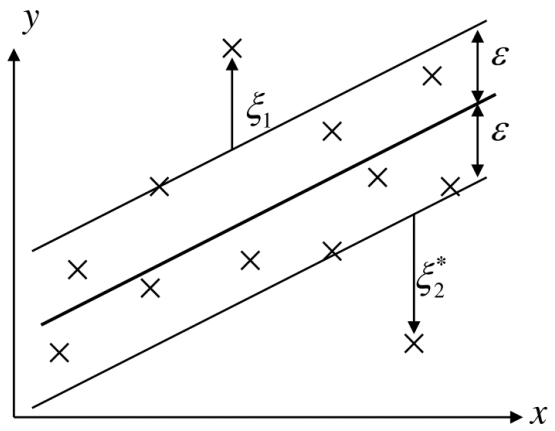
In Reading – SVM slides from An Introduction to Statistical Learning https://www.statlearning.com/

SVM for Regression

1. The objective is to fit as many instances as possible within the margin whose width is still controlled by hyperparameter.

2. Adding more training instances does not affect the model's prediction – hence the model is insensitive to the margin width.

3. Objective function $min\frac{1}{2}w^T w + C\sum_i^m \zeta^{(i)}\ \ subject\ to\ |y_i - w_i x_i| \leq \varepsilon + |\zeta^{(i)}|$



https://stats.stackexchange.com/questions/13194/support-vector-machines-and-regression

Kernel Methods

1. Memory-based methods – store the entire training set to make predictions for future data points.
2. Many linear parametric methods can be re-cast into equivalent "dual representation" in which the predictions are also based on linear combinations of kernel function evaluated at training data points

$$k(x_i, x_{i`}) = \phi(x_i)^T \phi(x_{i`})$$

3. Where $\phi(x)$ is a feature space mapping
4. Example – scalar inner product identity mapping for feature space

$$\phi(x) = x; \qquad k(x_i, x_{i`}) = {x_i}^T x_{i`}$$

5. Kernel trick / kernel substitution – input vector enters in the form of scalar product, then that can be replaced with some other choice of kernel.
6. Advantage is computational.

Kernel examples

Stationary kernels – function of the only the difference between arguments

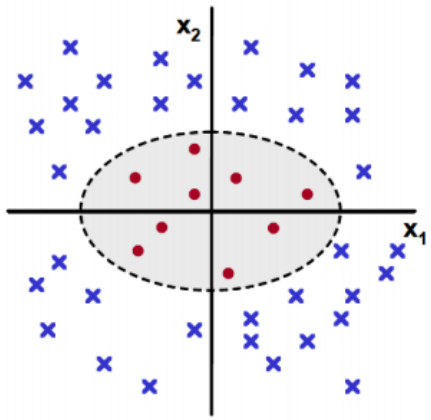
$$k(x_i, x_{i`}) = k(x_i - x_{i`})$$

Homogeneous kernels – radial basis functions (RBF) that depend only on the magnitude of the distance between arguments

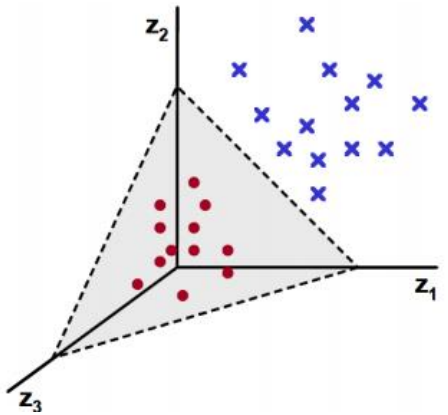
$$k(x_i, x_{i`}) = k(\|x_i - x_{i`}\|) = \exp(-\gamma \|x_i - x_{i`}\|^2)$$

Where $\|x_i - x_{i`}\|^2$ is the squared Euclidean distance

Linear in Kernel Transformed Space = Non-linear in Original Space



$$z_1 = x_1^2$$
$$z_2 = \sqrt{2}x_1\ x_2$$
$$z_3 = x_2^2$$



Heuristics when to use Linear Kernel

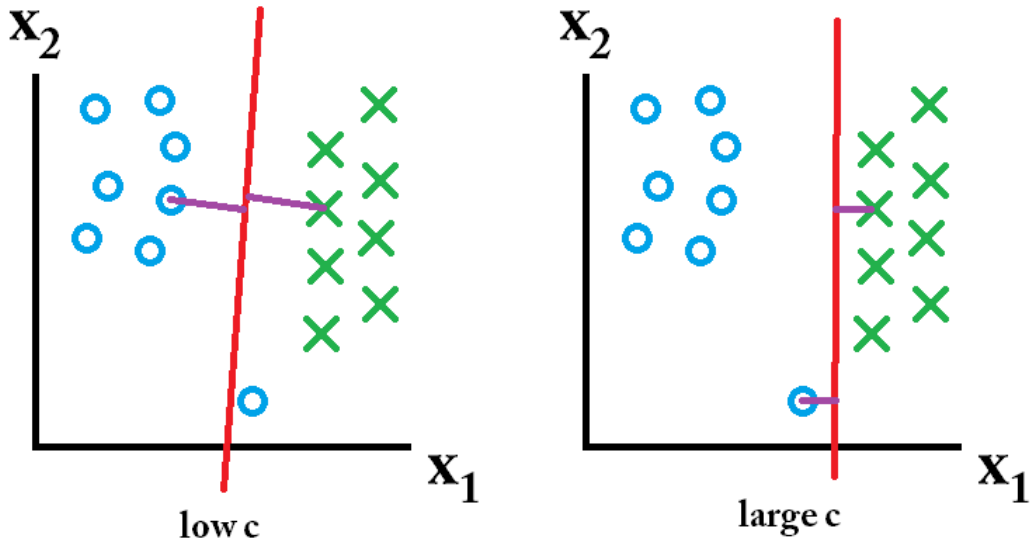
1. When number of features is large (few 10s of 1000) and number of training samples is small (few 1000) => linear kernel

2. When number of features is small (few 100s) and number of training samples medium (few 10s of 1000) do not use linear kernel => start with Gaussian

3. When number of features is small (few 100s) and number of training samples is large (few 100s of 1000 to millions) => linear kernel

SVM in scikit-learn

C is essentially a regularization parameter, which controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights.



https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

Textbook Chapters and Assignment

- Materials covered available in book:
  - DL: Chapter 5
  - HML: Chapters 5, 7
  - ISL: Chapters 8, 9

- Code: https://github.com/ageron/handson-ml2
- Different SVM Kernels: http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/