KAGGLE PREDICTION COMPETITION

ENSEMBLE MODEL: XGBOOST + GAM

THE ROADMAP TO MY FINAL MODEL

1. Linear Model with all parameters
   - Noted all the significant parameters
2. LASSO with CV (fold=5) for selection of Lambda. (Lambda = 0.0825404185268019)
   - Results were not as good as I hoped, so I decided to try Ridge.
3. Ridge with CV (fold=5) for selection of Lambda. (Lambda = 300)
4. Weighted Ensemble (0.5 weight each) of Ridge + Bagged LASSO with 200 trees.
   - Results were slightly better than using Ridge or LASSO individually
   - Test RMSE at this point was about 4.09 (first submission)

THE ROADMAP TO MY FINAL MODEL

5. XGBoost model

XGBOOST: INTRO

- Boosting: Sequentially fit residuals
- Gradient Descent: iteratively minimize a function by moving in the direction of the steepest descent.
- XGBoost (Extreme Gradient Boosting): an ensemble learning method.
  - Ensemble learning offers a systematic solution to combine the predictive power of multiple models. The resultant is a single model which gives the aggregated output from several models.
  - At each boosting iteration, the objective function is minimized via gradient descent.

XGBOOST

- Boosting vs Gradient Boosting:
  - **Boosting** fits models to the residuals.
  - **Gradient Boosting** fits models to the gradient of the **objective function.**
- **Objective function** at model m:

$$Obj = \sum_{i=1}^{n} Loss\left(y_i, \hat{y}_i^{(m-1)} + f^m(x)\right) + \underbrace{\sum_{m=1}^{M} \Omega\left(f_m\right)}_{\text{Regularization on model complexity}}$$

XGBOOST: REGULARIZATION TERM

- For tree t, with leaves j = 1,2,…, T, and weights w:

$$\Omega(f_t) = \gamma T + \underbrace{\alpha \sum_{j=1}^{T} w_j}_{\text{L1 Regularization}} + \underbrace{\frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2}_{\text{L2 Regularization}}$$

XGBOOST: HYPERPARAMETERS

- eta: Learning rate (amount the feature weights are shrunk at each iteration)
- nrounds: number of trees
- γ: cost of leaf
- max depth: maximum depth of a tree
- α: L1 regularization penalty on weights
- λ: L2 regularization penalty on weights
- subsample: proportion of training data to sample at each tree (with replacement).

XGBOOST: HYPERPARAMETERS CV WITH 7 FOLDS

- $\boldsymbol{eta}$: 0.01 to 0.05 step 0.01= 5 values
- nrounds: 100 to 5000 step 100 = 50 values
- γ: 1 (Default)
- max depth: 1 to 10 step 1 = 10 values
- $\boldsymbol{\alpha}$: 0 to 5 step 0.5 = 10 values
- $\boldsymbol{\lambda}$: 0 to 5 step 0.5 = 10 values
- subsample: 0.2 to 1.0 step 0.1 = 9 values

XGBOOST: HYPERPARAMETERS CV WITH 7 FOLDS

- Using a Grid Search of the hyperparameter space this means:
- 5*50*1*10*10*10*9 = 2,250,000 models
- But with 7-fold CV => 2,250,000 * 7

  = **15,750,000 models**

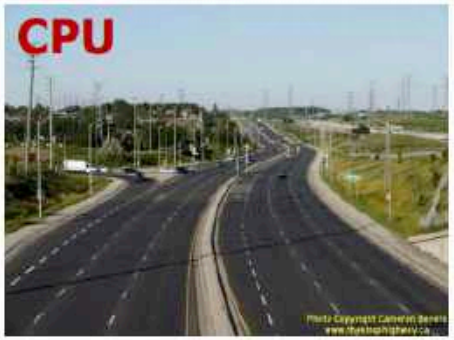
How to fit 15.75 million models in a reasonable amount of time?

CREDIT: Slide taken from CSC367
Lecture 9 by Bogdan Simion

Why GPUs?

- Large collection of SIMD multiprocessors
  - **Massive thread parallelism** – 100s of processors, **high memory bandwidth**
- Good for data-parallel computations
  - Must have an inherently high level of parallelism in our application

CPU



GPU

THE BEAUTY OF XGBOOST: GPU SUPPORT (CUDA)

- The XGBOOST algorithm is a very scalable and memory efficient algorithm.
- Multi-node Multi-GPU Training: XGBoost supports fully distributed GPU training through CUDA.
  - CUDA is a parallel computing platform and application programming interface model created by Nvidia for use on its GPUs.
- Hence, I took advantage of the Nvidia GPUs in the octolab machines in Deerfield Hall and used the cluster of 8 GPUs (GTX 1080: 2560 CUDA cores @1.6GHz) to parallelize my Hyperparameter tuning!
- It took **ONLY ~30 mins** to return the best hyperparameters from 15.7 million models.
  - 15.7 million/30mins* 60000 = **8.72 milliseconds a model**!!

XGBOOST: HYPERPARAMETERS RESULTS

- eta: 0.05
- nrounds: 250
- γ: 1 (Default)
- max depth: 2
- α: 1.5
- λ: 2
- subsample: 0.7

At this point using only my XGBoost model with optimal hyperparameters got a Test RMSE of ~**4.06**

THE ROADMAP TO MY FINAL MODEL

5. XGBoost: Test RMSE 4.06
6. New Weighted Ensemble of Ridge + Bagged LASSO + XGBoost
   - Same Ridge + Bagged LASSO as before
   - Weights: 0.33*LASSO + 0.33*Ridge + 0.33*XGB
   - Test RMSE: ~4.05 (Slight improvement)
7. My next few models/submissions were just me playing with the weights (trial and error):
   - Best weights: 0.25* Bagged LASSO + 0.15*Ridge + 0.6 * XGB
   - **Test RMSE: 4.034**
8. Added early stopping after 25 iterations and feature selection to my XGBoost model
   - Early stopping monitors the performance (RMSE) of the model on a validation dataset after each iteration of training and stops training if the model's performance does not get better after n=25 iterations. **Avoids Overfitting!**

FEATURE SELECTION USING XGBOOST

- A benefit I found of using gradient boosting is that after the boosted trees are constructed, it can calculate the importance scores for each attribute.
- Trained 109 new XGBoost models with the same hyperparameters using first all 109 features, then 108, then 107, and so on... removing the least important feature each time.
- Finally I simply selected the model with the lowest RMSE on my validation dataset, and trained a new model using only those features.
- Selected a model with 42 features.

THE ROADMAP TO MY FINAL MODEL

5. XGBoost: Test RMSE 4.06
6. New Equally Weighted Ensemble of Ridge + Bagged LASSO + XGBoost
   - Same Ridge + Bagged LASSO as before
   - Weights: 0.33*LASSO + 0.33*Ridge + 0.33*XGB
   - Test RMSE: ~4.05 (Slight improvement)
7. My next few models/submissions were just me playing with the weights (trial and error):
   - Best weights: 0.25* Bagged LASSO + 0.15*Ridge + 0.6 * XGB
   - **Test RMSE: 4.034**
8. Added early stopping after 25 iterations and feature selection to my XGBoost model
   - Early stopping monitors the performance (RMSE) of the model on a validation dataset after each iteration of training and stops training if the model's performance does not get better after n=25 iterations. **Avoids Overfitting!**
   - **New Test Ensemble RMSE: 3.99273**

THE ROADMAP TO MY FINAL MODEL

9. I made a GAM using all the 42 variables selected in XGBoost model.
   - I used the package's summary function to see which variables where significant and removed all the variables that were not significant (around 10).
   - Manually added spline functions on variables depending on whether it reduced my validation error or not. (2 variables: X101, X102)
   - Again, checked the summary and removed variables that were not significant (4 variables).
   - Then I added interactions to variables X4,X5 and X70,X80. Adding any more started increasing my validation error, despite reducing my training error.
   - New Test RMSE of Ensemble: 3.978
     - GAM model: model = LinearGAM(
       s(99) + s(100)
       + l(3) + l(6) + l(8) + l(11) + l(7) + l(9) + l(12) + l(10)
       + l(14) + l(29) + l(15) + l(71) + l(17) + l(21) + l(107)
       + l(16) + l(68) + l(78) + l(61) + l(55) + l(31) + l(13)
       + l(37) + l(4) + l(5) + l(2) + te(4, 5) + te(68, 78))

THE ROADMAP TO MY FINAL MODEL

10. I realized that my GAM function was highly overfitting the training data, because despite it bringing my ensemble's validation error lower, my test error only reduced by 0.01469. So, I added regularization to my model to reduce its flexibility.
    - I used the grid-search function on the GAM package to add an ideal lambda (penalty) to each of my functions.
    - Good improvement in test error
    - New Ensemble (of all 4 models) Test RMSE: 3.94404
11. Finally, in order to find the most optimum weights for each model in my ensemble I wrote a script that varied the weight of each of the 4 models, I found that I got the best RMSE's when the LASSO and Ridge model's weights were lowest. So I removed those models and only kept XGBoost + GAM!
    - Final Test RMSE of Ensemble (with only XGBoost + GAM): 3.940