Boosting

Gradient Boosting

Selecting and Tuning a Model

Frank Burkholder

Objectives

After this lecture you will be able to:

- List Gradient Boosting hyperparameters
- Be able to find the best hyperparameters for a model
  - Contrast exhaustive grid search and randomized search
- List a couple of **useful** non-sklearn based boosting algorithms
- Two jupyter notebook demos
  - `model-selection.ipynb`
    - general model selection procedure
  - `boosting-drury.ipynb`
    - has a useful example of staged-predict you need for your assignment

Gradient boosting in sklearn - and hyperparameters

class `sklearn.ensemble.` **`GradientBoostingRegressor`** (*loss='ls', learning_rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None, max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*) ¶ [source]

class `sklearn.ensemble.` **`GradientBoostingClassifier`** (*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*) ¶

Gradient boosted trees have all the same hyperparameters as decision trees, but with a few more:

- `learning_rate`
- `subsample`
- `max_features`

Selection of hyperparameters

GridsearchCV looks exhaustively through the parameters you give it to find the set that does the best on your scoring metric.

By default k=3, see documentation.

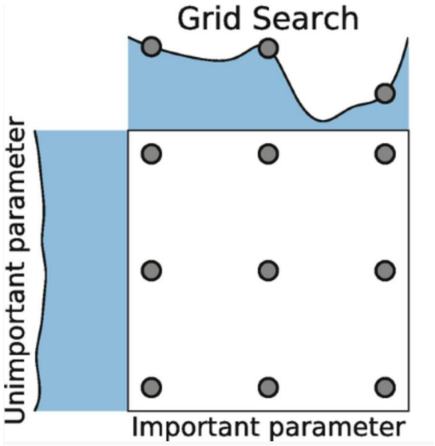
```
from sklearn.model_selection import GridSearchCV

random_forest_grid = {'max_depth': [3, None],
                      'max_features': ['sqrt', 'log2', None],
                      'min_samples_split': [1, 2, 4],
                      'min_samples_leaf': [1, 2, 4],
                      'bootstrap': [True, False],
                      'n_estimators': [20, 40, 60, 80, 100, 120],
                      'random_state': [42]}

rf_gridsearch = GridSearchCV(RandomForestClassifier(),
                             random_forest_grid,
                             n_jobs=-1,verbose=True,
                             scoring='f1_weighted')
rf_gridsearch.fit(X_train, y_train)
print "best parameters:", rf_gridsearch.best_params_
```

```
best parameters: {'bootstrap': True, 'min_samples_leaf': 1, '
     n_estimators': 100, 'min_samples_split': 1, 'random_state': 42, '
     max_features': 'sqrt', 'max_depth': None}
```

Types of Grid Search

- Exhaustive Grid Search (GridSearchCV)
  Looks through every combination of hyperparameters.

```
param_grid = {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}
```



= loss

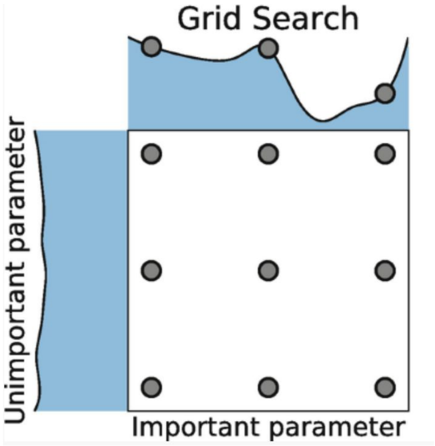
source: https://link.springer.com/chapter/10.1007/978-3-030-05318-5_1

Types of Grid Search

- Exhaustive Grid Search (`GridSearchCV`)
  Looks through every combination of hyperparameters.

```
param_grid = {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}
```

How many models would this Grid Search cause to be trained?



= loss

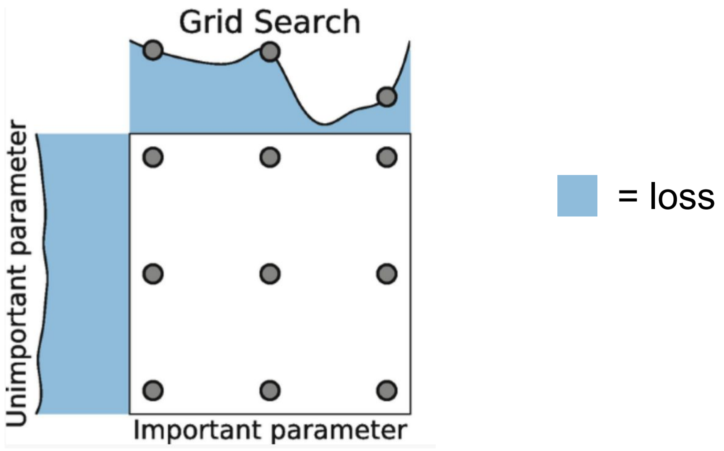
source: https://link.springer.com/chapter/10.1007/978-3-030-05318-5_1

Types of Grid Search

- Exhaustive Grid Search (GridSearchCV)
  Looks through every combination of hyperparameters.

```
param_grid = {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}
```

4 * 2 * 1 = 8, but by default 3 folds in cv, so 8 * 3 = 24

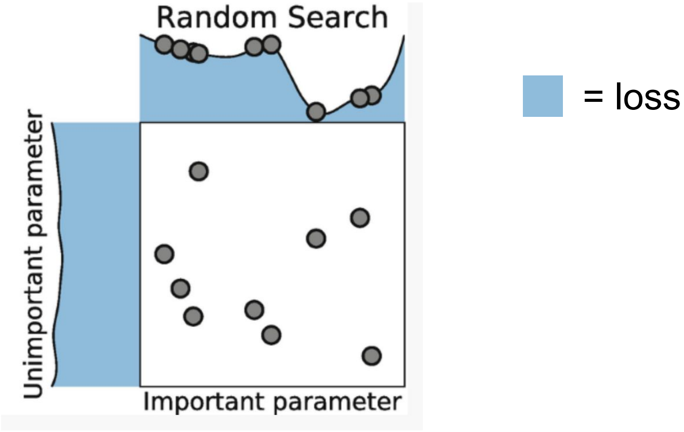


source: https://link.springer.com/chapter/10.1007/978-3-030-05318-5_1

Types of Grid Search

- Randomized Parameter Optimization (RandomizedSearchCV) Implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values. You tell it how many combinations to try.

```
{'C': scipy.stats.expon(scale=100), 'gamma': scipy.stats.expon(scale=.1),
  'kernel': ['rbf'], 'class_weight':['balanced', None]}
```



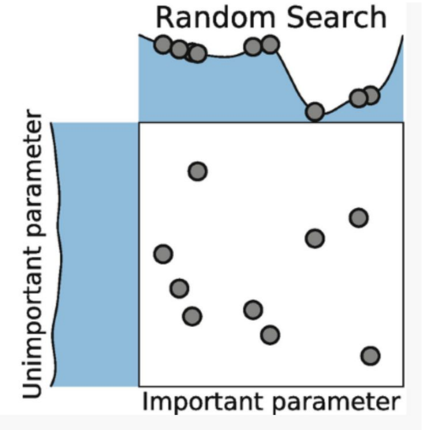
source: https://link.springer.com/chapter/10.1007/978-3-030-05318-5_1

Types of Grid Search

- Randomized Parameter Optimization (RandomizedSearchCV) Implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values. You tell it how many combinations to try.

```
{'C': scipy.stats.expon(scale=100), 'gamma': scipy.stats.expon(scale=.1),
  'kernel': ['rbf'], 'class_weight':['balanced', None]}
```

We will compare GridSearchCV to RandomizedSearchCV in `model_selection.ipynb`



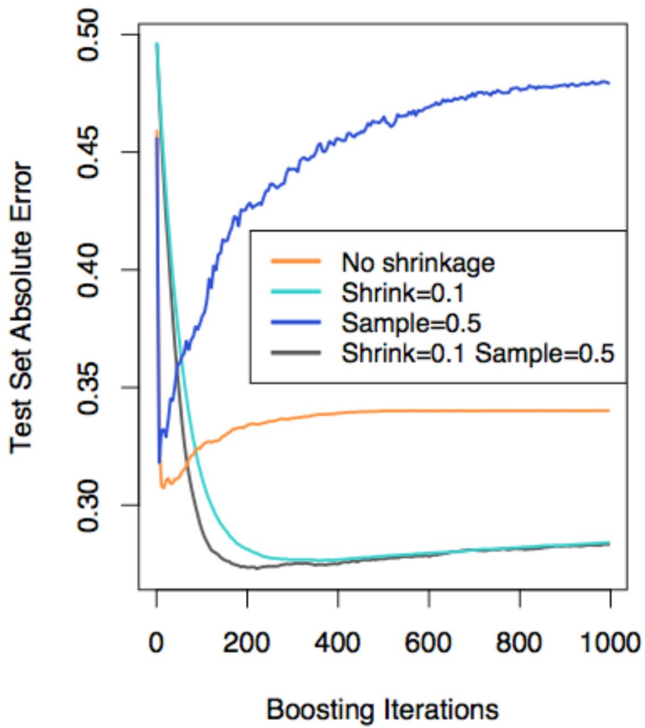
= loss

source: https://link.springer.com/chapter/10.1007/978-3-030-05318-5_1

Boosting tips

Overfitting can be a problem with boosting (debatable). To prevent this:

- Keep the base estimator simple (limit its `max_depth` to 2-8).
- Limit M, the maximum number of iterations. Use staged_predict (a method on a Gradient Boosting Regressor object) to monitor the train and test errors as they evolve with each iteration.
- Use shrinkage
- Use Stochastic Gradient Boosting using `subsample` and `max_features`
- Use large values for `min_samples_leaf` (also limits the depth of the tree)



Boosting - algorithms of note

XGBoost - A top performing algorithm on Kaggle.

note: installation on Mac involved - wait until you have time to debug

CatBoost - For only categorical features

Objectives

After this lecture you will be able to:

- List Gradient Boosting hyperparameters
- Be able to find the best hyperparameters for a model
    - Contrast exhaustive grid search and randomized search
- List a couple of **useful** non-sklearn based boosting algorithms
- Two jupyter notebook demos
    - `model-selection.ipynb`
        - general model selection procedure
    - `boosting-drury.ipynb`
        - has a useful example of staged-predict you need for your assignment

Jupyter notebook demos

`model_selection.ipynb`

(general model selection procedure)

`boosting_drury.ipynb`

(staged_predict for assignment)