W4995 Applied Machine Learning

(Gradient) Boosting, Calibration

02/20/19

Andreas C. Müller

Gradient Boosting

Boosting (in General)

Gradient Boosting Algorithm

$$f_1(x) \approx y$$

$$f_2(x) \approx y - f_1(x)$$

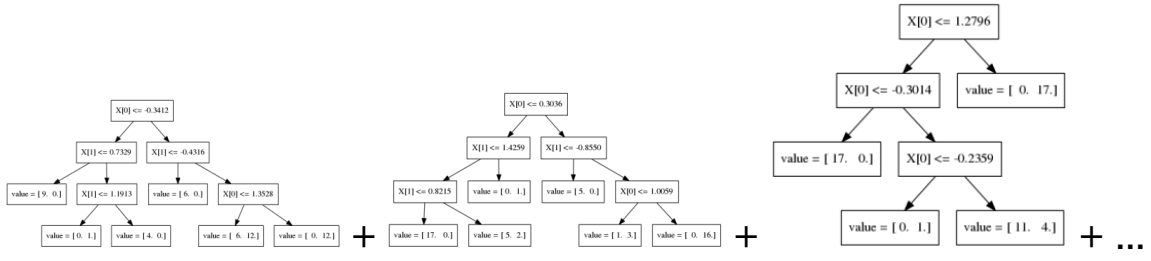
$$f_3(x) \approx y - f_1(x) - f_2(x)$$

Gradient Boosting Algorithm

$$f_1(x) \approx y$$

$$f_2(x) \approx y - f_1(x)$$

$$f_3(x) \approx y - f_1(x) - f_2(x)$$

$y \approx$ 

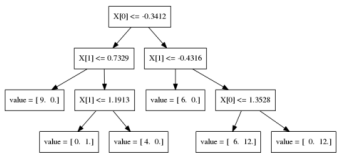
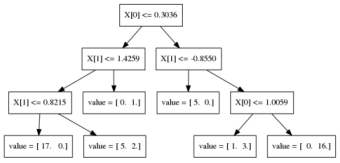
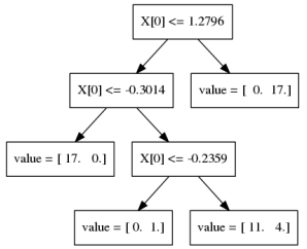
+ ...

Gradient Boosting Algorithm

$$f_1(x) \approx y$$

$$f_2(x) \approx y - \gamma f_1(x)$$

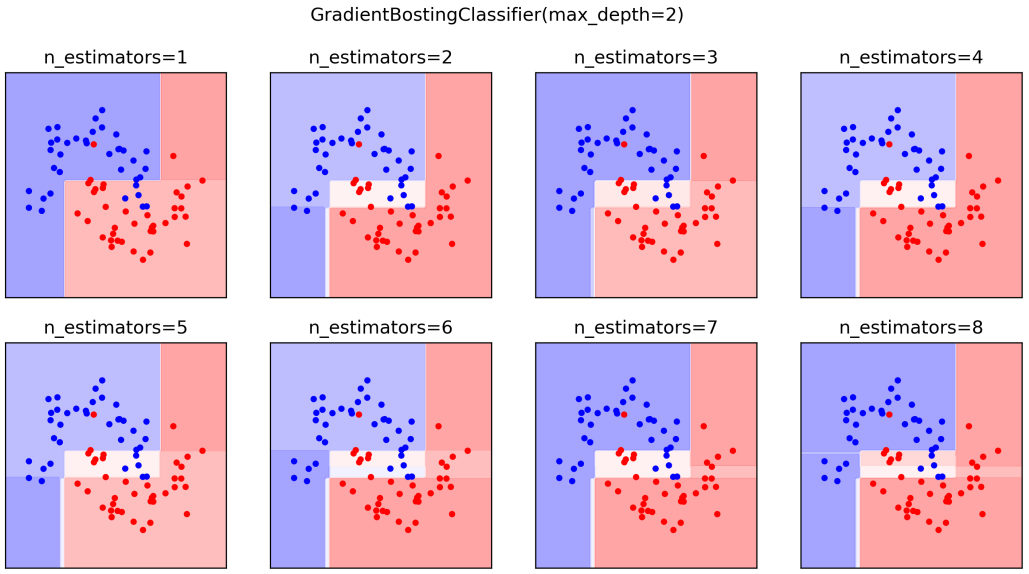
$$f_3(x) \approx y - \gamma f_1(x) - \gamma f_2(x)$$

$y \approx \gamma$  $+ \gamma$  $+ \gamma$  $+ \ldots$

Learning rate γ, $i.e.$ 0.1

GradientBoostingRegressor

GradientBoostingClassifier



Gradient Boosting Advantages

- Slower to train than RF (if using "old" GradientBoostingRegressor), but much faster to predict
- Very fast using XGBoost, LightGBM, pygbm, new scikit-learn implementation #12807
- Small model size
- Typically more accurate than Random Forests

Tuning Gradient Boosting

- Pick n_estimators, tune learning rate
- Can also tune max_features
- Typically strong pruning via max_depth

Improvements: Binning and "extreme" gradient boosting

A better split criterion (?)

- Optimize "weights" in leave (not just counting)
- Include second order (hessian) and regularizer

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

Binning

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted(I, by* $\mathbf{x}_{jk}$*)* **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Binning

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted(I, by* $\mathbf{x}_{jk}$*)* **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

Aggressive sub-sampling

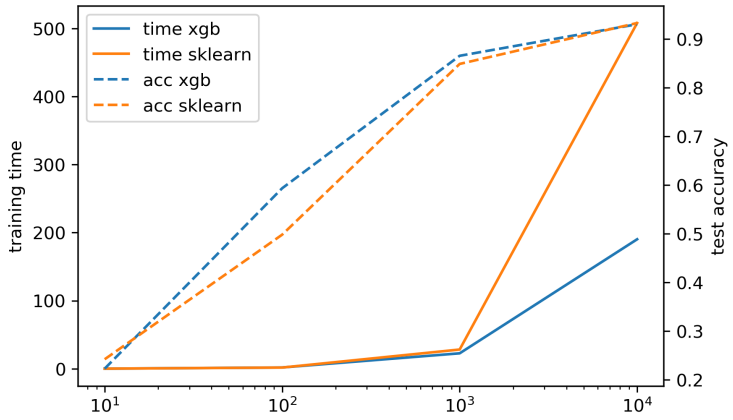
According to user feedback, using column sub-sampling prevents over-fitting even more so than the traditional row sub-sampling. (XGBoost: A Scalable Tree Boosting System, 2016)

XGBoost

```
conda install -c conda-forge xgboost
```

```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
xgb.score(X_test, y_test))
```

- supports missing values
- supports multi-core



- Exact splits ~5x faster on single core
- Appoximate splits, multi-core -> more speed!
- Also check lightGBM

Early stopping

- Adding trees can lead to overfitting
- Stop adding trees when validation accuracy stops increasing
- Optional in XGBoost and sklearn >= 0.20

More Tuning of Gradient Boosting

- Tune max_features
- Tune column subsampling, row subsampling
- Typically strong pruning via max_depth
- Regularization
- Pick learning rate and do early stopping?

Concluding tree-based models

When to use tree-based models

- Model non-linear relationships
- Doesn't care about scaling, no need for feature engineering
- Single tree: very interpretable (if small)
- Random forests very robust, good benchmark
- Gradient boosting often best performance with careful tuning

Calibration

Source

- Probabilities can be much more informative than labels:

- “The model predicted you don’t have cancer” vs “The model predicted you’re 40% likely to have cancer”

Calibration curve (Reliability diagram)

calibration_curve with sklearn

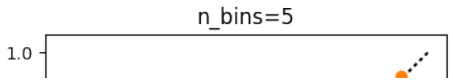
Using subsample of covertype dataset

```
from sklearn.linear_model import LogisticRegressionCV
print(X_train.shape)
print(np.bincount(y_train))
lr = LogisticRegressionCV().fit(X_train, y_train)
(52292, 54)
[19036 33256]
```

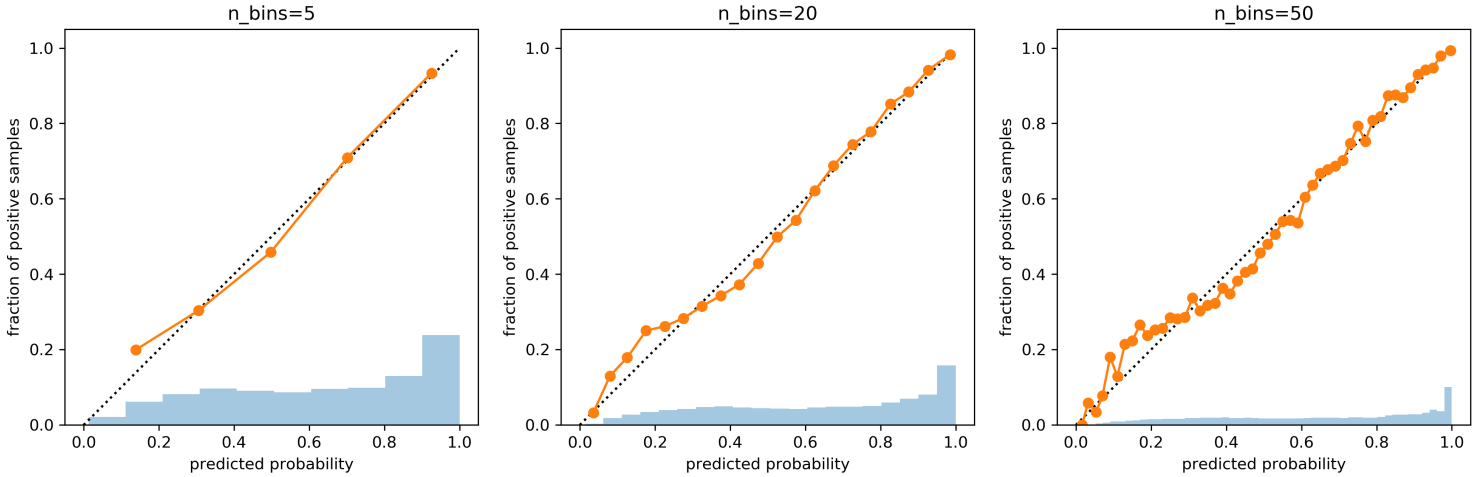
```
lr.C_
array([ 2.783])
```

```
print(lr.predict_proba(X_test)[:10])
print(y_test[:10])
[[ 0.681  0.319]
 [ 0.049  0.951]
 [ 0.706  0.294]
 [ 0.537  0.463]
 [ 0.819  0.181]
 [ 0.     1.   ]
 [ 0.794  0.206]
 [ 0.676  0.324]
 [ 0.727  0.273]
 [ 0.597  0.403]]
[0 1 0 1 1 1 0 0 0 1]
```

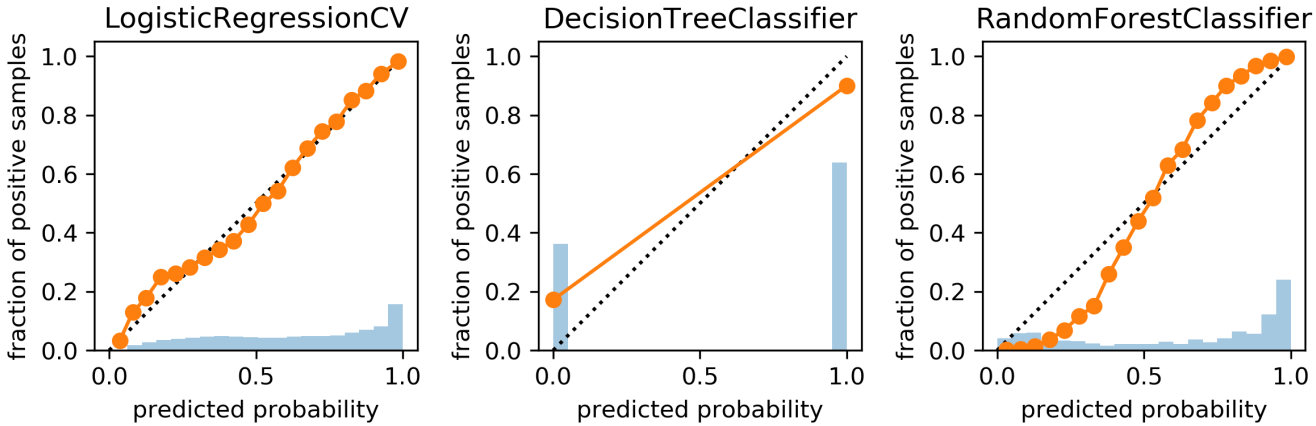
```
from sklearn.calibration import calibration_curve
probs = lr.predict_proba(X_test)[:, 1]
prob_true, prob_pred = calibration_curve(y_test, probs, n_bins=5)
print(prob_true)
print(prob_pred)
[ 0.2    0.303  0.458  0.709  0.934]
[ 0.138  0.306  0.498  0.701  0.926]
```



Influence of number of bins



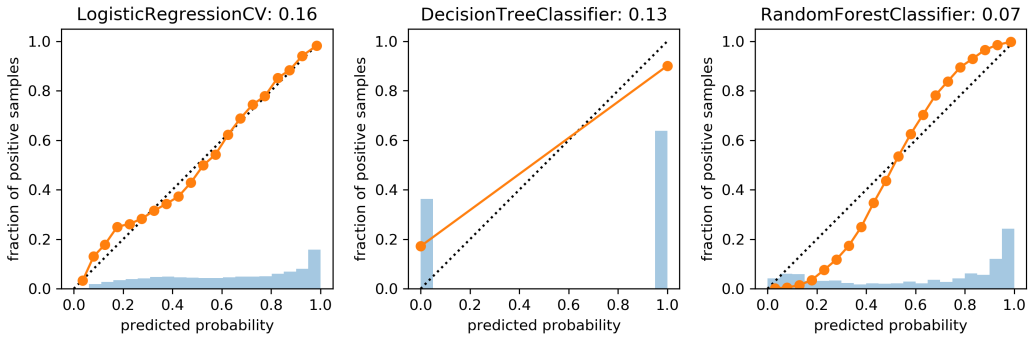
Comparing Models



Brier Score (for binary classification)

- "mean squared error of probability estimate"

$$BS = \frac{\sum_{i=1}^{n} (\hat{p}(y_i) - y_i)^2}{n}$$



Fixing it: Calibrating a classifier

- Build another model, mapping classifier probabilities to better probabilities!
- 1d model! (or more for multi-class)

$$f_{callib}(s(x)) \approx p(y)$$

- s(x) is score given by model, usually
- Can also work with models that don't even provide probabilities! Need model for f_{callib}, need to decide what data to train it on.
- Can train on training set → Overfit
- Can train using cross-validation → use data, slower

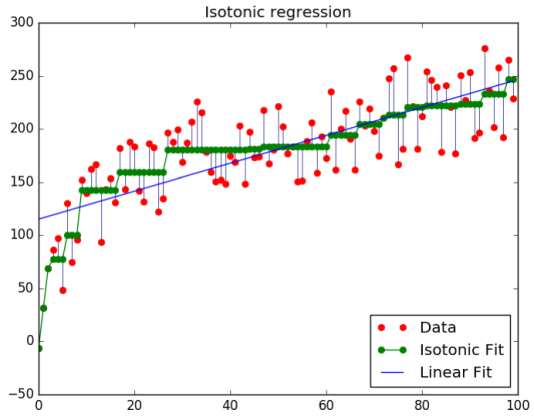
Platt Scaling

- Use a logistic sigmoid for f_{callib}
- Basically learning a 1d logistic regression
- (+ some tricks)
- Works well for SVMs

$$f_{platt} = \frac{1}{1 + exp(-ws(x) - b)}$$

Isotonic Regression

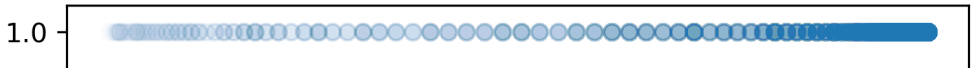
- Very flexible way to specify f_{callib}
- Learns arbitrary monotonically increasing step-functions in 1d.
- Groups data into constant parts, steps in between.
- Optimum monotone function on training data (wrt mse).



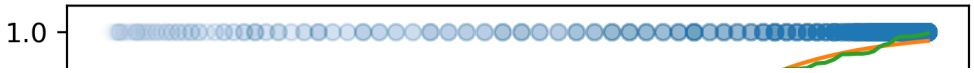
Building the model

- Using the training set is bad
- Either use hold-out set or cross-validation
- Cross-validation can be used to make unbiased probability predictions, use that as training set.

Fitting the calibration model

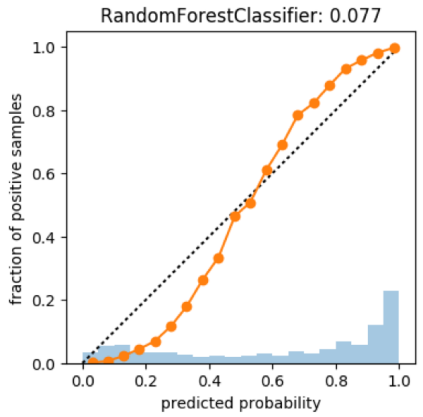


Fitting the calibration model



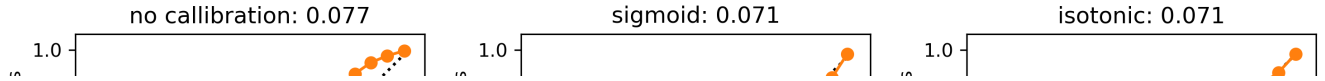
CalibratedClassifierCV

```
from sklearn.calibration import CalibratedClassifierCV
X_train_sub, X_val, y_train_sub, y_val = train_test_split(X_train, y_train,
                                                          stratify=y_train, random_state=0)
rf = RandomForestClassifier(n_estimators=100).fit(X_train_sub, y_train_sub)
scores = rf.predict_proba(X_test)[:, 1]
plot_calibration_curve(y_test, scores, n_bins=20)
```



Calibration on Random Forest

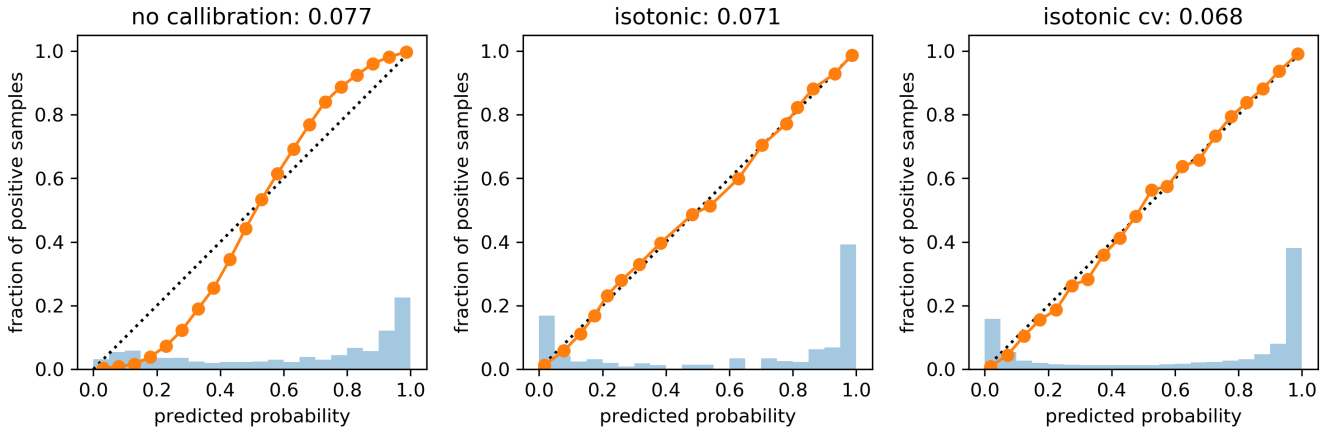
```
cal_rf = CalibratedClassifierCV(rf, cv="prefit", method='sigmoid')
cal_rf.fit(X_val, y_val)
scores_sigm = cal_rf.predict_proba(X_test)[:, 1]

cal_rf_iso = CalibratedClassifierCV(rf, cv="prefit", method='isotonic')
cal_rf_iso.fit(X_val, y_val)
scores_iso = cal_rf_iso.predict_proba(X_test)[:, 1]
```



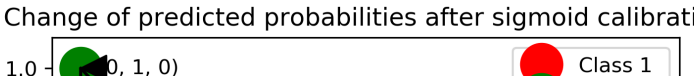
35 / 38

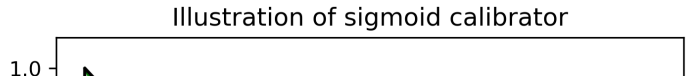
Cross-validated Calibration

```
cal_rf_iso_cv = CalibratedClassifierCV(rf, method='isotonic')
cal_rf_iso_cv.fit(X_train, y_train)
scores_iso_cv = cal_rf_iso_cv.predict_proba(X_test)[:, 1]
```



Multi-Class Calibration





Questions ?