Decision Trees, Ensembles

ML4SE

Denis Litvinov

October 6, 2022

Table of Contents

Linear Classifiers

Logistic Regression

Loss: $L = -y_i \log p_i - (1 - y_i) \log(1 - p_i)$

Decision function: $\hat{p}(y = 1|x) = \frac{1}{1+e^{-w^T x}}$

SVM

Loss: $L = \frac{1}{2C}||w||_2^2 + \sum_{i=1}^{N} max(0, 1 - y_i w^T x_i)$

Decision function: $\hat{y}(x) = sign(w^T x + b)$

Decision function from dual task: $\hat{y}(x) = sign(\sum_{i=1}^{N} \alpha_i y_i < x_i, x > +b)$

What are the differences?

Kernels

introduce $\psi : X \to H$, so $f(x) = < \alpha, \psi(x) >$ where H is some Hilbert space.
How to choose ψ?

Kernels 2

$K(x, z)$ is kernel iff

- $K(x, z) = K(z, x)$
- for any finite $\{x_i\}_{i=1}^{N}$ the matrix is positive semi-define.

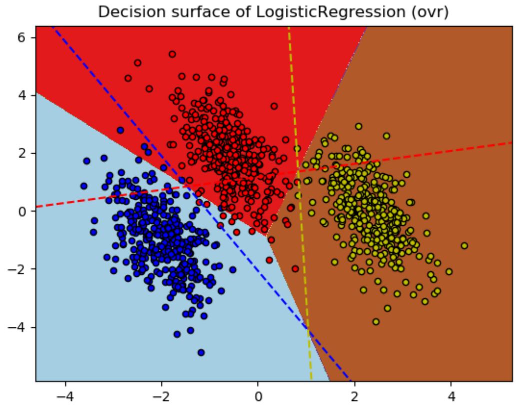
So we can write $K(x, z) =< \psi(x), \psi(z) >$
And $f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x)$

What are pros and cons of Kernel Trick ?

One vs Rest

Idea: build multiclass classifier from several binary classifiers
Train K binary classifiers.

$$\hat{y} = \arg\max_k h_k(x)$$



One vs Rest

Note:

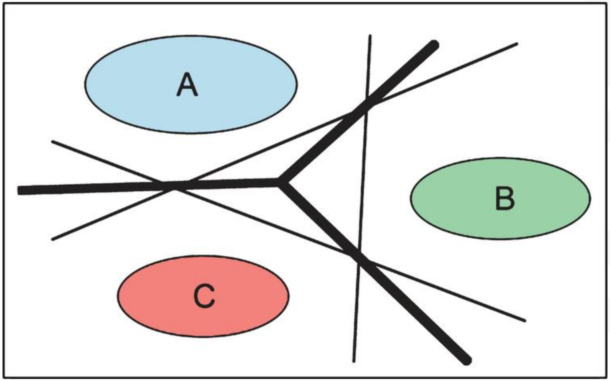
1. h_k is unbalanced even if initial problem was balanced
2. scale of the confidence values may differ between the binary classifiers b_k

One vs One

Idea: build multiclass classifier from several binary classifiers
Train $K(K-1)/2$ binary classifiers.

$$\hat{y} = \arg\max_k \sum_{i \neq k} h_{ik}(x)$$

Note: One vs one is less prone to imbalance in dataset



Multinomial

Cross-entropy loss:

$$Loss(y_i, p_i) = -\sum_{k=1}^{K} y_{ik} \log p_{ik}$$

where k is a number of classes

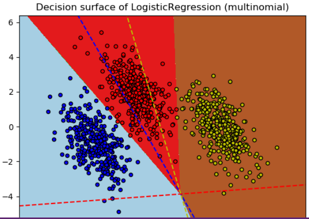
$y_{ik} = I[y_i = k]$

$p_{ik} = p(y_i = k|x_i)$

Probability if x_i has class k:

$$p(y_i = k|x_i) = softmax(W^T x_i)_k$$

where $W \in R^{DxK}$



Softmax

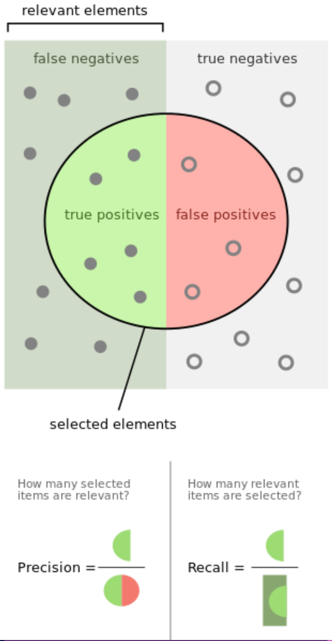
Requirements:

1. non-negative
2. sums to 1 (is a probability)
3. monotonic increasing

for $z \in R^K$

$$softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

$$\begin{bmatrix}1.2\\0.9\\0.4\end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix}0.46\\0.34\\0.20\end{bmatrix}$$


relevant elements
false negatives
true negatives
true positives
false positives
selected elements
How many selected items are relevant?
Precision =
How many relevant items are selected?
Recall =

Quality Metrics

There is no direct quality metric, it is assembled from metrics for binary classification problems.

$$Pr_{micro} = \frac{\sum_k TP_k}{\sum_k TP_k + \sum_k FP_k}$$

$$Re_{micro} = \frac{\sum_k TP_k}{\sum_k TP_k + \sum_k FN_k}$$

$$F1_{micro} = \frac{2 * Pr_{micro} * Re_{micro}}{Pr_{micro} + Re_{micro}}$$

weighted in proportion of class size:

$$Pr_w = \frac{1}{K} \sum_k \frac{|K|}{N} \frac{TP_k}{TP_k + FP_k}$$

$$Re_w = \frac{1}{K} \sum_k \frac{|K|}{N} \frac{TP_k}{TP_k + FN_k}$$

$$F1_w = \frac{2 * Pr_w * Re_w}{Pr_w + Re_w}$$

Quality Metrics

$$Pr_{macro} = \frac{1}{K}\sum_{k}\frac{TP_k}{TP_k + FP_k}$$

$$Re_{macro} = \frac{1}{K}\sum_{k}\frac{TP_k}{TP_k + FN_k}$$

$$F1_{macro} = \frac{2 * Pr_{macro} * Re_{macro}}{Pr_{macro} + Re_{macro}}$$

Which metric is insensitive to class imbalance

Quality Metrics

Macro averaging is insensitive to class imbalance.

Validation

Till now:

- $\{x_i, y_i\}_{i=1}^N$ is sampled from $P(x, y)$
- Choose appropriate Quality metric Q
- Choose loss function to mimic quality metric behaviour
- Split into to non-overlapping subsets (train and test)
- Model $h(x_i; w, \theta)$ is described by its trainable weights w and non-trainable hyperparams θ

Validation

Till now:

- choose some hyperparam value $\theta = \theta_0$ and train model

$$\sum_{i \in train} Loss(h(x_i; w, \theta_0), y_i) \to \min_w$$

- test model performance on test dataset

$$\hat{R}(\theta_0) = \sum_{i \in test} Loss(h(x_i; w_*, \theta_0), y_i)$$

$$\hat{Q}(\theta_0) = Q(h(X_{test}; w_*, \theta_0), y_{test})$$

- we expect that it is a good approximation

$$R(\theta_0) = E_{(x,y) \sim P(x,y)}[Loss(h(x_i; w, \theta_0), y_i)]$$

$$Q(\theta_0) = E_{D = \{(x,y) | (x,y) \sim P(x,y)\}}[Q(h(X_D; w_*, \theta_0), y_D)]$$

How to choose hyperparam θ?

Validation

Usually want to optimize hyperparams by testing several values of θ on the test set and choosing the best one.

But the performance on the test set $\hat{R}(\theta)$(empirical risk) is a random variable, which can depend on the particular train test split! Here validation comes into play.

We can say that $\hat{R}(\theta)$ is a point estimate of expected risk $R(\theta)$, which has its bias and variance.

Different validation schemes try to minimize bias or variance or both.

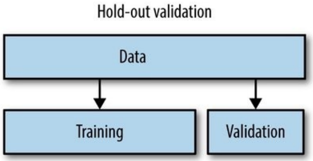
Hold Out

Given dataset of m objects, create m experiments:

1. create split train:val, usually in proportion 70:30, 80:20 or 90:10
2. fit model weights on train subset
3. evaluate performance on the val subset

Properties:

- High bias and low variance of estimate
- O(1) complexity
- Usually done when we have large dataset and or very heavy model



Cross Validation

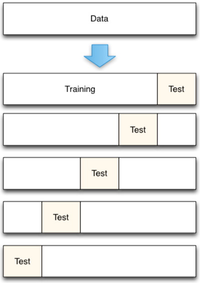
k = number of folds folds = non-intersecting subsets of the dataset

Make k experiments:

1. create split for $k-1$:1
2. train on $(k-1)$ folds and evaluate performance on the k-th fold
3. change split
4. Average scores over all experiments

Properties:

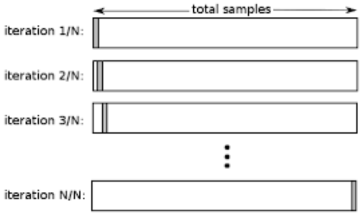
- Moderate bias and variance of estimate
- O(k) complexity



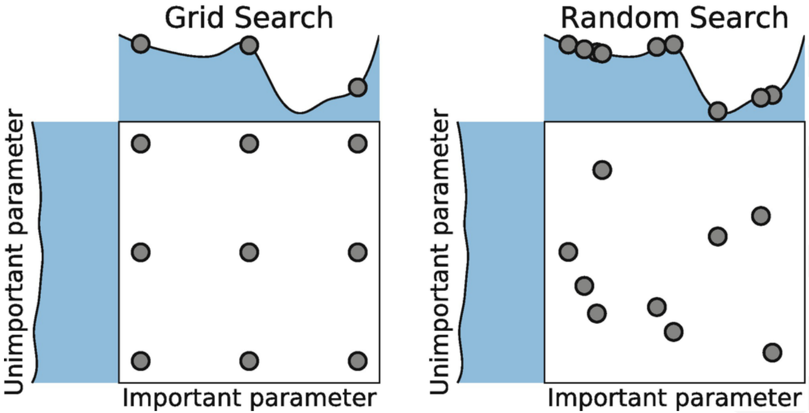
Leave One Out

Cross validation with $k = N$
Properties:

- Low bias and High Variance of estimate
- O(N) complexity
- Usually done when we have very small dataset
- There are performance metrics (e.g. AUC) that cannot be computed just on one sample.



Hyperparam Search

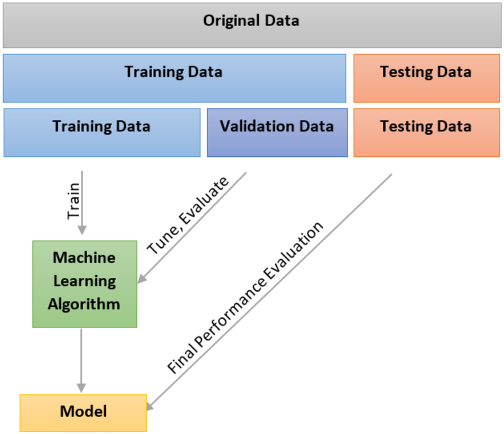


Common Pipeline

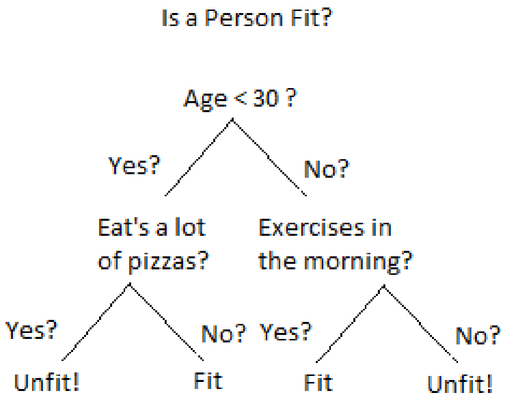
1. Split dataset for train, test parts
2. Choose validation scheme on training data
3. Train model on the train dataset without regularization, try to achieve zero training loss
4. Add regularization, tune hyperparams on validation
5. Evaluate final model performance on test dataset. Choose between different model families.

In practice we usually use chosen quality metric instead of loss function for choosing hyperparams and final testing.

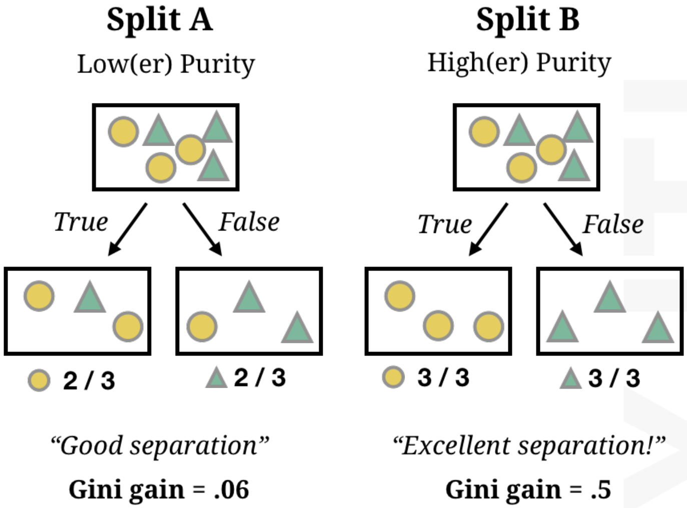
Common Pipeline



Decision Tree



Decision Tree



Definitions

Splitting criteria on vertex v: *j*-th component of feature vector *x* is less than the threshold *t*

$$\beta_v(x; j, t) = [x_j < t]$$

Greedy algorithm to build decision tree

```
Given a vertex $v$
    for every feature $f$:
        for every threshold $t$ on f:
            estimate chosen splitting criterion
Select $(t, f)$ that maximizes chosen criterion.
Make a split of incoming dataset into left $L$ and right $R$ subsets.
```

Impurity Criteria

Impurity criterion for dataset R: minimize loss function *Loss* with constant prediction c

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} Loss(y_i, c)$$

Impurity criterion for MSE regression

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c)^2$$

We know that for MSE task optimal value of c is

$$c_* = \frac{1}{|R|} \sum_{(x_j, y_j) \in R} y_j$$

Thus, impurity criterion is a variance of y

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c_*)^2 = Var[y]$$

Impurity Criteria

Impurity criterion for MAE regression

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} |y_i - c|$$

$$c_* = ?$$

Classification Criteria

Define class probability distribution over K classes

$$p_k = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k]$$

Most frequent class

$$k = \arg\max_k p_k$$

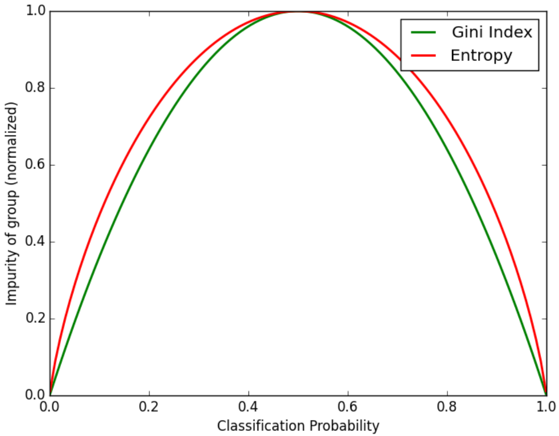
Classification error

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i \neq k_*] = 1 - p_*$$

Classification Criteria

Gini Criterion $H(R) = \sum_{k=1}^{K} p_k(1 - p_k)$

Entropy Criterion $H(R) = -\sum_{k=1}^{K} p_k \log p_k$



Objective

Objective function on every split: we want to increase the difference of the impurity criterion in the current vertex and its weighted sum in subtrees, that resulted after the split.

$$Q(R_m, f, t) = H(R_m) - \frac{|R_l|}{|R_m|} H(R_l) - \frac{|R_r|}{|R_m|} H(R_r) \to \max_{f,t}$$

where R_m initial dataset in the current vertex
R_l dataset in the left subtree after the split
R_r dataset in the right subtree after the split

Stopping Criteria

- max tree depth
- min number of objects in the leaf
- max number if leaves
- criterion gain $Q^{(k+1)} - Q^{(k)} < \epsilon$
- all objects in the leaf are of the same class

Stopping criterion controls tree complexity.

Complexity of training

n - number of samples
d - number of features
$O(\log d)$ - complexity of prediction
$O(nd \log n)$ - time training complexity

Decision Tree vs Linear Model

About linear models so far, **Pros**:

- Fast training
- Simple regularization

Cons:

- Hard to deal with non-linear function dependencies. Need of feature engineering and heuristics
- Scale sensitive
- Need of one-hot encoding of categorical features

Decision Tree vs Linear Model

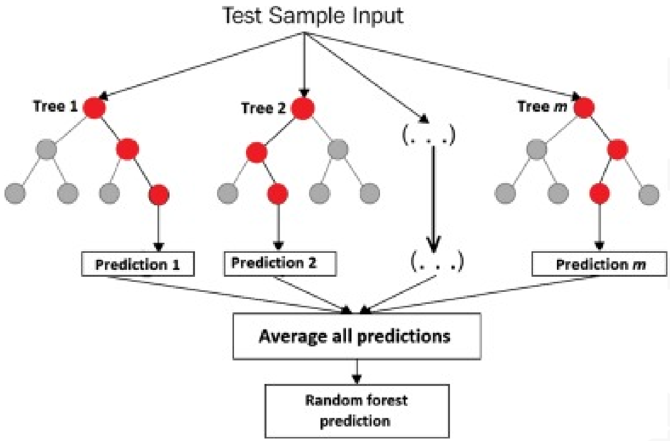
Decision trees, **Pros**:

- Can approximate non-linear functions of any kind
- Easy interpretable
- Scale insensitive
- Missing values
- Can process categorial features

Cons:

- Subject to severe overfitting
- Unstable - small variations in data produce different trees
- learning an optimal decision tree is NP-hard. All algorithms are greedy.
- Not efficient in high dimensions. Number of samples required to populate the tree doubles for each additional level the tree grows to.

Bagging


Test Sample Input
Tree 1
Tree 2
(. . .)
Tree m
Prediction 1
Prediction 2
(. . .)
Prediction m
Average all predictions
Random forest prediction

Bagging

1. Bootstrap = sampled subset with repetitions from initial dataset
2. Bagging = averaging over predictions of T base models trained on bootstraped datasets

$$F(x) = \frac{1}{T}\sum_{t=1}^{T} h_t(x)$$

where
$F(x)$ - bagging ensemble model
$h_t(x)$ - base model, i.e. decision tree - must be uncorrelated
T - number of base models

Bagging

$$\begin{aligned} MSE &= Var[\epsilon] + E[(f(x) - E[h(x)])^2] + Var[h(x)] \\ &= Var[\epsilon] + bias^2 + Var[h(x)] \end{aligned}$$

Why bagging works?

Random Forest

Bagging gives $\frac{1}{T}$ factor in variance reduction under the assumption that models are not correlated
Random Forest:

1. for each tree subsample features from initial dataset
2. Train N non-correlated decision trees in parallel
3. Average their predictions

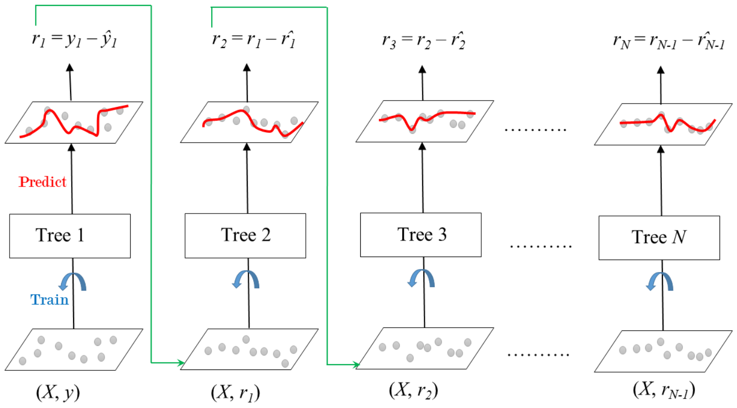
Random forest is a bagging of decision trees with subsampling over features.

Out-of-Bag Score

Every decision tree in the forest is trained on bootstrapped subset, so objects not included in train subset can be considered as control set = out-of-bag

$$OOB = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \frac{1}{\sum_{t=1}^{T}[x_i \notin X_t]}\sum_{t=1}^{T}[x_i \notin X_t]h_t(x_i))$$

Gradient Boosting



Boosting

Any boosting can be described by

$$F(x) = \frac{1}{T}\sum_{t=1}^{T} w_t h_t(x)$$

where $F(x)$ - boosting ensemble model
$h_t(x)$ - base model, i.e. decision tree
w_t - weight coefficient for t base model
T - number of base models

Unlike bagging, boosting is aimed to reduce bias in the predictions. Difference between Adaboost and Gradient Boosting in how the w_t is calculated.

AdaBoost

Iteratively build decision trees, trying to predict the errors of the last tree.
Given finite set $\{(x_i, y_i)\}_{i=1}^{N}$, where $y \in \{-1, 1\}$ and loss function

$$L(y, \hat{y}) = e^{-y\hat{y}}$$

```
1 init sample weights $w^1 = \frac{1}{N}$
2 For t in 1..T
2.1 $\epsilon_t = \sum_{h^t(x_i) \neq y_i} w_i^t$
2.2 $h^t(x) = \arg\min_h \epsilon_t$
2.3 $\alpha^t = \frac{1}{2} \ln \frac{1-\epsilon^t}{\epsilon^t}$
2.4 $F^t(x) = F^{(-1}(x) + \alpha^t h^t(x)$
2.5 $w_i^t = w_i^{t-1} \exp(-y_i \alpha^t h^t(x_i))$
2.6 normalize $w^t$, so $\sum_i w_i^t = 1$
```

Gradient Boosting

Given finite set $(x_i, y_i)_{i=1}^N$ and some differentiable loss function $L(y, \hat{y})$.

1. select $h^{(0)} = \arg\min_{c=const} L(y, c)$
2. for t in $1..T$
2.1 pseudo-residuals $r_i^t = -\nabla_y dL(y, F^{t-1}(x_i))$ in the point $y = F^{t-1}(x)$
On each iteration we wish to predict gradient of loss function over samples.
2.2 fit new h^t on the dataset $\{(x_i, r_i^t)\}_{i=1}^N$ with MSE loss
2.3 solve 1-D optimization task
$\alpha^t = \arg\min_\alpha \sum_i L(r_i, F^{t-1}(x_i) + \alpha h^t(x_i))$
2.4 $F^t(x) = F^{t-1}(x) + \alpha^t h^t(x)$

Note, that gradient has dimension equal to number of samples. =>
More data you have - more time to compute full gradient.

Boosting

Can we use linear models for base estimator in boosting or bagging?