1 Classification Intro

1.1 Basic Loss Functions

Let $\hat{y} \doteq \text{sign}(f(x_i))$ and $y_i \in \{-1, 1\}$, then the fraction of times $\hat{y}_i \neq y_i$ is

$$\begin{aligned}\mathcal{F} &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{[y_i \neq \text{sign}(f_i)]} \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{[y_i f(x_i) < 0]} \\ &\leq \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y_i f(x_i))\end{aligned}$$

where $\mathcal{L}$ is a loss function that acts as an upper bound on $\mathbb{I}$. $\mathcal{L}$ is usually continuous or otherwise easier to minimize than $\mathbb{I}$. By minimizing $\mathcal{L}$, we also minimize $\mathbb{I}$.

The "margin" $y_i f(x_i)$ can be thought of as how good a classification is. The more positive the margin is, the better the prediction is for x_i.

1.2 Ockham's Razor and Regularization

The principle of Ockham's Razor can be used to create better models. By penalizing the complexity of a model, we can create models that generalize better. Thus in practice we try to minimize

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y_i f(x_i)) + \text{Complexity}(f)$$

The "Complexity" function can be a number of things, such as a regularization term like $\|\lambda\|$ or $\|\lambda\|^2$, where λ represents the parameters of the model.

1.3 Imbalanced Data

If one class has only a few points, we can get what appears to be a good classifier by just predicting the same thing every time. This is obviously not good, so we can introduce a weighting term to make sure that both classes are of equal importance to the classifier.

$$\frac{1}{n}\left[C\sum_{y_i=1} \mathcal{L}(y_i f(x_i)) + \sum_{y_j=-1} \mathcal{L}(y_j f(x_j))\right] + \text{Regularization}(f)$$

2 Cross Validation

2.1 Evaluation

We need a dataset, algorith, and an evaluation measure. The process is

1. Divide data into equally-sized folds

2. Train on all but 1 fold and compute the evaluation measure on the last fold
3. Repeat with each fold as the test fold
4. Report μ and σ of eval measure over each trial

To compare algorithms that have ungergone this process, do matched-pairs t-tests to see if one algorithm's results are statistically better than another's.

2.2 Parameter tuning

The process for parameter tuning is

1. Set aside a test fold
2. Reserve a validation set from the training set
3. Training the algorithm on the rest of the training set for each possible parameter value and evaluate it on the validation set
4. Rotate the validation fold and repeat
5. Report μ of each evaluation measure for each parameter value over the validation folds, then choose the best parameter value
6. Train on the full training set (training + validation) with the best parameter value, then evaluate it on the test set

2.3 Nested Cross Validation

Nested cross validation uses cross validation for evaluation in an outer loop and cross validation for parameter tuning in an inner loop. It's essentially considering parameter tuning to be part of the algorithm.

This is very expensive computationally, so people try to take shortcuts like using fewer folds, fewer possible parameter values, etc.

3 Data Science Process

The data science process can be divided into several general stages

1. Acquisition
2. Cleaning
3. Integration/Aggregation/Transformation
4. Analysis/Modeling
5. Interpretation

Data science is an iterative process, so you have to go backwards to previous steps a lot (e.g. your analysis shows that your data is biased, so you have to go back and acquire more data and complete the subsequent steps again).

4 Information Theory

4.1 Entropy

The **information** of an event is the number of bits needed to encode the probability of that event. If p is the probability of an event, its information is $I(p) = -\log_2(p)$.

The expected information of a discrete distribution P is

$$\begin{aligned}\mathbb{E}_{p\sim P}[I(p)] &= \mathbb{E}_{p\sim P}\left[-\log_2(p)\right] \\ &= \sum_{i=1}^{n} p_i I(p_i) \\ &= -\sum_{i=1}^{n} p_i \log_2 p_i \\ &=: \mathcal{H}(P)\end{aligned}$$

and is called the **entropy** of the distribution P.

4.2 Intuition behind the definition of information

We want our definition of information to satisfy

1. $I(p) \geq 0, I(1) = 0$
2. $I(p_1 \cdot p_2) = I(p_1) + I(p_2)$ if p_1 and p_2 are independent
3. $I(p)$ is continuous

This implies that $I(p^n) = nI(p)$, which can be generalized further to $I(p^{n/m}) = \frac{n}{m}I(p)$. This is true for any fraction, so we want our function I to satisfy $I(p^r) = rI(p)$ for any $r \in \mathbb{R}$. The functions that do this are $I(p) = -\log_b(p)$. We chose $b = 2$ because of the connection with bits.

5 Decision Trees

Decision trees are useful because they are interpretable, model discrete outcomes nicely, and are powerful, non-linear, and capable of being as complex as you need.

Greedy tree induction:

1. Start at top of the tree
2. Grow it by splitting on features one by one. To split, look at how "impure" the node is (some features give more information than others)
3. Assign leaf nodes the majority vote in the leaf
4. At the end, go back and prune leaves to reduce overfitting

5.1 Information Gain

One potential splitting criterion is information gain, which is defined as the entropy reduction after branching on a given feature. Let S be a tree and A be a feature that we are splitting on. Let $B(A)$ be the branches created when splitting on A, and let P_i and N_i be the number of positives and negatives in branch i (if no subscript is given, then it is for the entire tree S). Then

$$\text{Gain}(S, A) = \mathcal{H}\left(\left[\frac{P}{P+N}, \frac{N}{P+N}\right]\right) - \sum_{i=1}^{|B(A)|} \frac{P_i + N_i}{P+N} \cdot \mathcal{H}\left(\left[\frac{P_i}{P_i+N_i}, \frac{N_i}{P_i+N_i}\right]\right)$$

To choose a feature to split on, choose A such that S,A is maximized. Then recurse and do this until the tree has been fully formed.

This procedure tries to split such that the data in each leaf node is as far from the uniform distribution as possible. This is good for accuracy, but can also lead to overfitting since we're encouraging many branches.

5.2 Information Gain Ratio

The information gain ratio is defined

$$\frac{\text{Gain}(S, A)}{\text{SplitInfo}(S, A)}$$

Let $|S_j|$ be the amount of data in branch j (if no subscript is given, then it is for the entire tree S), then the splitting info is defined

$$\text{SplitInfo}(S, A) = -\sum_{j=1}^{|B(A)|} \frac{|S_j|}{|S|} \log\left(\frac{|S_j|}{|S|}\right)$$

If $|S_j|/|S|$ is large, then the summation will be large, and this means the negative of the summation will be small. This encourages each branch to have more points, which reults in fewer branches overall.

5.3 Entropy Replacements for Binary Splits

5.3.1 Gini Index

This is used by CART, and is defined $2p(1-p)$. This is the variance of the Bernoulli distribution for p.

5.3.2 Misclassification Error

This can be defined $1 - \max(p, 1-p)$. Since classification is based on majority vote (i.e. $p \leq 0.5 \implies$ no and $p > 0.5 \implies$ yes), this gives the correct value regardless of the correct label.

insert figure here

5.3.3 Pruning

C4.5 has 3 options:

1. Leave the tree as is

2. Collapse a subtree into a leaf
3. Replace a subtree with one of its own subtrees (most common branch for that split)

insert figure here

It chooses which option to take by computing an upper bound on the probability of error for each option. It uses a standard upper confidence bound on probabilities from the binomial distribution with $\alpha = 0.25$ (why? It's just a heuristic so this is just what was settled on after a while). what is α?

Take a weighted average of the err bound over each branch (for each option), then pick the option that has the lowest one.

5.4 CART

Differences from C4.5:

- Only binary splits
- Gini index for splitting
- Uses minimum cost complexity for pruning

5.4.1 Classification

Each subtree is assigned a cost, and the subtree with the lowest cost is chosen. Let $L(S)$ be the set of leaves of tree S, then the cost is defined

$$c(S) = \sum_{\ell \in L(S)} \sum_{x_i \in \ell} \mathbb{I}_{[y_i \neq \ell\text{'s class}]} + \lambda |L(S)|$$

This cost funciton balances accuracy and sparsity. Note that each new leaf is worth the same as λ misclassified points.

5.4.2 Regression

Assign $f(x)$ to be constant in each leafe. Choose the value of $f(x)$ to minimize the squared loss

$$\begin{aligned}\mathcal{L}^{\text{train}}(f) &= \sum_i (y_i - f(x_i))^2 \\ &= \sum_{\ell \in L(S)} \sum_{x_i \in \ell} (y_i - f(x_i))^2\end{aligned}$$

Since f is constant in ℓ, this becomes

$$\begin{aligned}&= \sum_{\ell \in L(S)} \sum_{x_i \in \ell} (y_i - f_i)^2 \\ &=: \sum_{\ell \in L(S)} \mathcal{L}_j^{\text{train}}(f_\ell)\end{aligned}$$

Now choose f_ℓ to minimize $\mathcal{L}_j^{\text{train}}(f_\ell)$. We can solve this explicitly by simply setting the derivative to 0.

$$\begin{aligned}
0 &= \frac{d}{d\tilde{f}} \sum_{x_i \in \ell} (y_i - \tilde{f})^2 \Big|_{\tilde{f}=f_\ell} \\
0 &= -2 \sum_{x_i \in \ell} (y_i - \tilde{f}) \Big|_{\tilde{f}=f_\ell} \\
0 &= \left(\sum_{x_i} y_i \right) - |S_\ell| \tilde{f} \quad \Big|_{\tilde{f}=f_\ell} \\
\tilde{f} &= \frac{1}{|S_\ell|} \sum_{x_i \in \ell} y_i \\
&=: \overline{y}_{S_\ell}
\end{aligned}$$

The value $\overline{y}_{S_\ell}$ is the average of the labels of leaf ℓ's examples.

6 Boosting

Boosting takes a weak learning algorithm and turns it into a strong learning algorithm. It reweights the data and uses the weak learning algorithm to create a weak classifier for the weighted data. It then makes an update based on the weighted average of all the weak classifiers.

6.1 Generic Boosting Algorithm

For $t = 1$ to T:

- Construct discrete probability distribution $\mathbf{d}_t$ over indices $\{1, \ldots, n\}$
- Run the weak algorithm A on $\mathbf{d}_t$ to produce weak classifier $h_t : \mathcal{X} \to \{-1, 1\}$
- Calculate misclassification error $\varepsilon_t = \mathbb{P}(h_t(x_i) \neq y_i \mid i \sim \mathbf{d}_t) =: \frac{1}{2} - \gamma_t$, where $\gamma_t > \gamma_A$ by the weak learning assumption.
- Output H, a combination of the h_t's.

6.2 AdaBoost

Assign observation i the weight $\mathbf{d}_{1,i} = \frac{1}{n}$ (equal weights).

For $t = 1$ to T:

- Train the weak algorithm using the data weighted by $\mathbf{d}_{t,i}$ to produce weak classifier h_t.
- Choose coefficient
$$\alpha_t = \frac{1}{n} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$
- Update weights
$$\mathbf{d}_{t+1,i} = \frac{\mathbf{d}_{t,i} e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

- Output final classifier $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Note that if $y_i = h_t(x_i)$, then the weight update is $\mathbf{d}_{t,i} e^{-\alpha_t}/Z_t$, so the weight gets smaller. If $y_i \neq h_t(x_i)$ instead, then the weight update is $\mathbf{d}_{t,i} e^{\alpha_t}/Z_t$, so the weight gets larger. This is essentially making incorrectly-classified points more important and correctly-classified points less important.

6.3 Coordinate Descent

Given a function $\mathcal{L}(\mathbf{b})$, we can minimize the function by moving along one coordinate of $\mathbf{b}$ at a time.

$$\min_\alpha \mathcal{L}\left([b_1, \ldots, b_j + \alpha, \ldots, b_p]\right) = \min_\alpha \mathcal{L}\left(\mathbf{b} + \alpha \mathbf{e}_j\right)$$

This can be a useful alternative to gradient descent when

- the gradient can't be calculated
- the feasible region is constrained (e.g. SVM)
- you want to control the optimization

6.4 AdaBoost as Coordinate Descent

AdaBoost can be viewed as coordinate descent on the exponential loss. Weak classifier j is "coordinate" j, and moving in direction j by α means adding α to the coefficient of weak classifier j.

To derive AdaBoost, we start by bounding the misclassification error by the exponential loss

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{[y_i f(x_i) \leq 0]} \leq \frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(x_i)}$$

Now choose f to be a linear combination of weak classifiers

$$f(x) = \sum_{j=1}^{p} \lambda_j h_j(x)$$

where $h_j = (x_i) = x_{ij}$ is the j-th feature of point x_i. Then the exponential objective becomes

$$\begin{aligned} \mathcal{L}(\lambda) &= \frac{1}{n}\sum_i e^{-y_i \sum_j \lambda_j h_j(x_i)} \\ &= \frac{1}{n}\sum_i e^{-(M\lambda)_i} \end{aligned}$$

where M is the "matrix of margins" $[M_{ij}] = y_i h_j(x_i)$, the margin of i for the j-th weak classifier. Now we're set up to do coordinate descent on λ.

6.4.1 Step 1: Find discent direction from point λ_t

$$\begin{aligned} j_t &= \arg\max_j \left[-\frac{d\mathcal{L}(\lambda_t + \alpha e_j)}{d\alpha}\Big|_{\alpha=0} \right] \\ &= -\frac{d}{d\alpha} \end{aligned}$$

finish this.

the derivation gets rid of the "arg max" because that's annoying to write out.

6.5 Summary

The objective function is

$$\mathcal{L}(\lambda) = \frac{1}{n} \sum_i e^{-(M\lambda)_i}$$

At time t, the descent direction is

$$j_t = \arg\max_j (d_t^T M)_j,$$

the weight update coefficient is

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right),$$

and the new data distribution is

$$\mathbf{d}_{t+1,i} = \frac{e^{-(M\lambda_{t+1})_i}}{Z_{t+1}}.$$

showing this is equivalent to original formulation, then probabilistic interpretation, convergence, and other notes

7 Logistic Regression

We want to minimize the logistic loss (another upper bound of the $0-1$ loss).

$$\min_f \frac{1}{n} \sum_{i=1}^n \log \left(1 + e^{-y_i f(x_i)} \right)$$

where $f(x_i) = \sum_{j=1}^p \beta_j x_{ij} = \mathbf{x}_i \beta$.

8 Convex Optimization

8.1 The Primal Objective

We want to solve the "OPT" problem

$$\begin{gathered} \min_{x \in \mathbb{R}^n} f(x) \\ \text{subject to } g_i(x) \leq 0 \text{ for } i = 1, \ldots, m \\ \text{and } h_i(x) = 0 \text{ for } i = 1, \ldots, p \end{gathered}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and each $g_i : \mathbb{R}^n \to \mathbb{R}$ are differentiable and convex and each $h_i : \mathbb{R}^n \to \mathbb{R}$ is affine.

Note that $f(x)$ is convex if and only if $-f(x)$ is concave. Also note that an affine function is both convex *and* concave.

We can rewrite the OPT problem with the constraints *in* the objective.

$$\min_x \Theta_P(x) \doteq f(x) + \infty \sum_{i=1}^m \mathbb{I}_{[g_i(x)>0]} + \infty \sum_{i=1}^p \mathbb{I}_{[h_i(x)\neq 0]}$$

This isn't differentiable or even continuous, so we can consider a more convenient lower bound on it instead.

Note that $\alpha u \leq \infty \cdot \mathbb{I}_{[u>0]}$ for any $\alpha \geq 0$, and $\beta u \leq \infty \cdot \mathbb{I}_{[u\neq 0]}$ for any β. Thus we can form a lower bound

$$\Theta_P(x) \geq \mathcal{L}(x,\alpha,\beta) \doteq f(x) + \sum_{i=1}^m \alpha_i g_i(x) + \sum_{i=1}^p \beta_i h_i(x).$$

This lower bound is the **Lagrangian** of the primal objective Θ_P. It has primal variable x and dual variables α and β.

Proposition 1. *The maximum of $\mathcal{L}(x,\alpha,\beta)$ with respect to α and β is $\Theta_P(x)$.*

Proof. We consider two cases: when the constraints are satisfied and when they are violated.

1. Assume the constraints are satisfied, then $h_i(x) = 0$ and $g_i(x) \leq 0$ for all i. Since $\alpha_i \geq 0$, we clearly maximize $\mathcal{L}$ when $\alpha_i = 0$ for all i. Then $\mathcal{L}(x,\alpha,\beta) = \Theta_P(x)$.
2. Now assume the constraints are violated. If the g_i are violated, then $\mathcal{L}$ is clearly maximized when $\alpha \to \infty$. Then $\mathcal{L} = \infty = \Theta_P$. If the h_i are violated, then let $\beta \to \infty$ for $h_i > 0$ and let $\beta \to -\infty$ for $h_i < 0$. Then $\mathcal{L} = \infty = \Theta_P$.

□

Thus when $\alpha_i \geq 0$ for all I, $\Theta_P(x) = \max_{\alpha,\beta} \mathcal{L}(x,\alpha,\beta)$. From here on out, the condition that $\alpha_i \geq 0$ is implicit. The primal problem can be summarized

$$\min_x \Theta_P(x) = \min_x \left[\max_{\alpha,\beta} \mathcal{L}(x,\alpha,\beta)\right]$$

The solution to the primal problem is denoted by $x^* \in real^n$, and the optimal value of the primal objective by $p^* = \Theta_P(x^*)$. A point x is "primal feasible" if all g_i and h_i are satisfied by x.

Proposition 2. *The primal objective Θ_P is convex in x.*

Proof. We know

$$\Theta_P(x) = \max_{\alpha,\beta} \mathcal{L}(x,\alpha,\beta) = \max_{\alpha,\beta} f(x) + \sum_{i=1}^m \alpha_i g_i(x) + \sum_{i=1}^p \beta_i h_i(x).$$

We also know that $f(x)$ and each g_i are all convex. Since $\alpha_i \geq 0$ and the sum of convex functions is also convex, the sum $\sum_i \alpha_i g_i(x)$ is convex. Since each h_i is affine, the sum $\sum_i = \beta_i h_i(x)$ is convex regardless of the signs of the β_i's. The maximum of a collection of convex functions is clearly also convex, so $\Theta_p(x)$ is convex. □

8.2 The Dual Objective

We form the dual objective by swapping the order of the minimum and maximum.

$$\begin{aligned} \textbf{Primal:}\ \ & \min_x [\Theta_P(x)] = \min_x \left[\max_{\alpha,\beta} \mathcal{L}(x,\alpha,\beta)\right] \\ \textbf{Dual:}\ \ & \max_{\alpha,\beta} [\Theta_D(\alpha,\beta)] = \max_{\alpha,\beta} \left[\min_x \mathcal{L}(x,\alpha,\beta)\right] \end{aligned}$$

We have similar notation for the dual objective. The solution is denoted by (α^*, β^*), the optimal value of the dual objective is denoted by $d^* = \Theta_D(\alpha^*, \beta^*)$, and a pair (α, β) is "dual feasible" if $\alpha_i \geq 0$ for all i.

Proposition 3. *The dual objective $\Theta_D(\alpha, \beta)$ is a concave function of α and β.*

Proof. For fixed x, $f(x)$ is constant and each $\alpha_i f(x)$ and $\beta_i f(x)$ is affine with respect to α_i and β_i. Affine functions are concave, the sum of affine functions is affine, and the minimum of a set of concave functions is clearly also concave. □

Proposition 4. *If (α, β) is dual feasible, then $\Theta_D(\alpha, \beta) \leq p^*$.*

Proof. By construction, $\alpha_i g_i(x) \leq \infty \cdot \mathbb{I}_{[g_i(x)>0]}$ and $\beta_i h_i(x) \leq \infty \cdot \mathbb{I}_{[h_i(x)\neq 0]}$, so $\mathcal{L}(x, \alpha, \beta) \leq \Theta_P(x)$ for all x. This implies $\Theta_D(\alpha, \beta) = \min_x \mathcal{L}(x, \alpha, \beta) \leq \min_x \Theta_P(x) = p^*$. □

Proposition 5 (Weak Duality)**.** *For any pair of primal and dual problems, $d^* \leq p^*$.*

Proof. The previous proposition holds for any feasible (α, β), so it certainly holds for the optimal (α^*, β^*) as well. Thus we have $d^* = \max_{\alpha,\beta} \Theta_D(\alpha, \beta) \leq p^*$. □

Proposition 6 (Strong Duality)**.** *If a pair of primal and dual problems satisfies the "constraint qualifications", then $d^* = p^*$.*

The only condition we really care about is **Slater's Condition**, which says that there exists some primal feasible x for which all g_i are *strictly* satisfied, i.e. $g_i(x) < 0$ for all i. In most cases in ML, strong duality holds.

8.3 KKT conditions

For unconstrainted optimization tasks, we can set the gradient to 0 and solve. For constrained optimization, we can use the KKT conditions instead.