1 Boosting

1.1 Ada

Greedy algorithm for combining weak classifiers. using exponential loss as surr.
We can also view it as a greedy algorithm for fitting α under the sparsity constraint $\|\alpha\|_0 \le M$.

$H(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + ... + \alpha_m h_m(\mathbf{x})$
$\hat{h}(\mathbf{x}) = sign(H(\mathbf{x}))$

Algorithm: Each point equal weight $W_i^{(0)} = 1/N$; For M iterations; Fit h' with error weighted by W_i s.t. 0/1 loss $\epsilon' < 1/2$ so that $\alpha' > 0$; Snuggness of hypothesis $\alpha' = \frac{1}{2}\log\frac{1-\epsilon'}{\epsilon'}$; Update weight with normed exp loss $\mathbf{W}'_i = \mathbf{W}_i e^{-\alpha' y_i h'(\mathbf{x}_i)}/\sum_j \mathbf{W}'_j$.

α' is the vote for a function is – big alpha for small error. W' represents a distribution of who is angriest with the current predictor. $y_i h'(\mathbf{x}_i)$ will be 1 if they agree and -1 if they disagree, so will push the weights/contentness in one way or the other.

Exponential loss
$L(H(\mathbf{X}), \mathbf{y}) = \sum_{i=1}^N e^{-y_i H(\mathbf{x}_i)}$
$= \sum e^{-y_i H(\mathbf{x}_i)} e^{-\alpha' y_i h'(\mathbf{x}_i)}$
$= \sum \mathbf{W}_i e^{-\alpha' y_i h'(\mathbf{x}_i)}$
$= e^{-\alpha'} \sum_{y_i \neq h'(\mathbf{x}_i)} \mathbf{W}_i + e^{\alpha'} \sum_{y_i = h'(\mathbf{x}_i)} \mathbf{W}_i$

We get α' by minimizing loss.
$\frac{\partial L}{\partial a_t} = -e^{-\alpha'} \sum_{y_i \neq h'(\mathbf{x}_i)} \mathbf{W}_i + e^{\alpha'} \sum_{y_i = h'(\mathbf{x}_i)} \mathbf{W}_i = 0$
$e^{-\alpha'}(1-\epsilon) = e^{\alpha'}\epsilon$
$\alpha' = \frac{1}{2}\log\frac{1-\epsilon}{\epsilon}$

0/1 loss; $\epsilon' = \sum_{y_i \neq h'(\mathbf{x}_i)} \mathbf{W}_i$

Exponential loss is an upper bound for 0/1 loss, is differentiable. Train error of H goes down, ϵ' goes up, α' goes down, exponential loss goes strictly down. Even after training, test error can go down. This is because it increases the margin of training examples. $\gamma(\mathbf{x}_i) = y_i \frac{H'(\mathbf{x})}{\sum_i \alpha_i}$

1.2 Gradient

Fit to the gradients or residuals of the current model. The gradient of squared loss is the residual.
$\hat{y}(\mathbf{x}) = F_M(\mathbf{x}) = f_1(\mathbf{x}) + ... + f_m(\mathbf{x})$

Algorithm: $F_1(\mathbf{x}) = \frac{1}{N}\sum_i y_i$; For M iterations; $F'(\mathbf{X}) = F(\mathbf{X}) + f'(\mathbf{X}) = F(\mathbf{X}) - fit(y_i - F(\mathbf{X}))$ since residuals are negative gradients of loss.

Works for nonlinear data with high bias but low variance. Yeah NGL kinda lost on this.

2 SVM

Maximizes margin in classification, hinge loss.
If not specified, constraints hold $\forall i = 1, \ldots, N$.
Distance to boundary: $y(\mathbf{w}\cdot\mathbf{x} + w_0)/\|\mathbf{w}\|$
Maximize margin/maximize distance to closest point: $\arg\max_{\mathbf{w}, w_0} \min_i\{distance_i\}$
Assume separable, $ls = y_i(w_0 + \mathbf{w}\cdot\mathbf{x}_i) - 1 \ge 0$

$\hat{y} = sign(\hat{w_0} + \sum_{\alpha_i > 0} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x})$

Primal: $\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2$ s.t. ls
$= \min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + \sum \max_{\alpha_i \ge 0} \alpha_i[1 - y_i(w_0 + \mathbf{w}\cdot\mathbf{x}_i)]$
$= \max_{\alpha \ge 0} \min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + \sum \alpha_i[1 - y_i(w_0 + \mathbf{w}\cdot\mathbf{x}_i)]$
Distance preserved by scaling weight and bias because the norm will be scaled by the same amount, set min distance to 1. $\|\mathbf{w}\|$ is largest when $\|\mathbf{w}\|^2$ is smallest. Introduce a lagrange multiplier to convert the constraint to a loss, infinity if violated, else 0. To max, if point is further away (distance < 0), $\alpha_i = 0$, else not a ls predictor. Use KKT and strong duality (convex and affine) to get maxmin. Minimizing wrt $\mathbf{w}$ and w_0, $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$ and $0 = \sum \alpha_i y_i$. But $\alpha_i > 0$ only if you are a support vector, so $\hat{w} = \sum_{\alpha_i > 0} \alpha_i y_i \mathbf{x}_i$.
Dual: $\max\{\sum_i^N \alpha_i - \frac{1}{2}\sum_{i,j}^N \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\}$ s.t. $\sum \alpha_i y_i = 0, 0 \le \alpha_i$

For nonseparable, ls assumption violated, so cap our cost to C. Introduce slack variable ξ_i that is the hinge loss.
Hinge loss: $\max\{0, 1 - y_i(w_0 + \mathbf{w}\cdot\mathbf{x}_i)\}$.
Primal: $\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum \xi_i$
Dual: $\max\{\sum_i^N \alpha_i - \frac{1}{2}\sum_{i,j}^N \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\}$ s.t. $\sum \alpha_i y_i = 0, 0 \le \alpha_i \le C$

Kernel trick: Basically applied when dot product is present because it measure similarity. Kernel must be continuous, symmetric, and positive definite.
RBF: $K(\mathbf{x}, \mathbf{z}; \sigma) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/\sigma^2)$
σ controls spread; if 0, then all points become support vectors.

SVM for regression
$y_i \le f(\mathbf{x}_i) + \epsilon + \xi_i$ and $y_i \le f(\mathbf{x}_i) - \epsilon - \xi'_i$
$\min C\sum_i(\xi_i + \xi'_i) + \frac{1}{2}\|\mathbf{w}\|^2$
Anything within ϵ is okay, but penalize with hinge loss after that.

3 Trees

Regression
$f(\mathbf{x}) = \sum_{m=1}^M f_m I_{x \in R_m}$, $f_m = \sum_{pt \in R_m} y_i/N_m$

Want to minimize squared loss. Not computationally tractable, so do a greedy splitting algorithm. Use split s to divide left and right, then our cost is,
$\min_{f_L} \sum_{pt \in R_L}(y_i - f_L)^2 + \min_{f_U} \sum_{pt \in R_U}(y_i - f_R)^2$

Easily overfits. Bad idea to split if gain is small if –_– data. Use pruning instead.
$C(T; \lambda) = \sum_{m=1}^{|T|} N_m Q_m(T) + \lambda|T|$
Where Q is the leaf error and lambda decides if the region is worth it.

Classification
Similar idea, want to minimize 0/1 loss per leaf.
$\hat{y} = \arg\max_c \hat{p}_{m,c}$, $\hat{p}_{m,c} = I_{x \in R_m}/N_m$
$Q_m(T) = \sum_{c=1}^C \hat{p}_{m,c}(1 - \hat{p}_{m,c})$

3.1 Boosting

Ensemble by summing low variance, high bias (shallow, underfitting). ± 1 if x_j is above or below a threshold. Complexity depends on number of classifiers and complexity of classifiers. Basically, for each stump and class, we add up the scores for the class for the leafs that the data point goes to in the trees.

3.2 Bagging/random forests

Ensemble by averaging high variance, low bias (deep, overfitting). Use bootstrapping and sampling to introduce randomness and make trees look different. **Bootstrapping** samples N points with replacement. **Sample features** by only consider splits along a subset of features. Grow a bunch of trees. To make a prediction, either take the average or vote. Tune on the number of trees or the feature set size, tree depth/leaves.

4 Generative

4.1 Bayes/Discriminant

$h(\mathbf{x}) = \arg\max_c p(y_c|\mathbf{x}) = \arg\max_c \frac{p(\mathbf{x}|y_c)p(y_c)}{p(\mathbf{x})}$
Since we know that $p(\mathbf{x})$ will divide all equally, and that each class is equally likely $p(y = c|\mathbf{x}) = 1/C$, we can simplify the discriminant function to be,
$h(\mathbf{x}) = \arg\max_c\{\log p(\mathbf{x}|y = c)\} = \arg\max_c\{\delta_c(\mathbf{x})\}$

For the binary case,
$h(\mathbf{x}) = \arg\max_{c=\pm 1} \delta_c(\mathbf{x}) = sign(\delta_{+1}(\mathbf{x}) - \delta_{-1}(\mathbf{x})) = sign\left(\log\frac{p(\mathbf{x}|y=+1)}{p(\mathbf{x}|y=-1)}\right)$
If we assume that $p(\mathbf{x}|y)$ are normal, then we can show we are modeling the same thing as logistic regression. Using naive bayes, we basically look at the points in a class, and memorize what the feature look like. This works given enough separation between features for classes.

4.2 Mixture

Detect k components within a class.
$p(\mathbf{x}; \pi) = \sum_{c=1}^k p(y = c)p(\mathbf{x}|y = c)$
We want but don't have binary indicators $\mathbf{z}$ for components, but we can't compute it. However, if we take the expectation wrt posterior of $\mathbf{z}$, we can get the responsibilities $\gamma_{i,c}$ that estimate $\mathbf{z}$.
$E_{z_{ic} \sim \gamma_{ic}}[z_{ic}] = \sum_{z \in 0,1} z \cdot \gamma_{i,c}^z = \gamma_{i,c}$.

Algorithm: Start with a guess of μ, π, e.g. $\pi_c = 1/k$. Expectation to get $\gamma_{i,c}$, then MLE to get parameters. Repeat until convergence. Evidently, this depends a lot on initalization.

Gaussian mixture
$p(\mathbf{X}, Z; \pi, \mu_1, ...) = \prod_{i=1}^N \prod_{c=1}^K (\pi_c \mathcal{N}(\mathbf{x}_i; \mu_c, \Sigma_c))^{z_{ic}}$
$E[l('')] = \sum_i^N \sum_c^K \gamma_{ic}(\log \pi_c + \log \mathcal{N}(\mathbf{x}_i; \mu_c, \Sigma_c))$
$\gamma_{ic} = \frac{\pi_c p(\mathbf{x}_i; \mu_c)}{\sum_{l=1}^k \pi_l p(\mathbf{x}_i; \mu_l)}$ by bayes
$N_c = \sum_{i=1}^N \gamma_{ic}$
$\hat{\pi}_c = \frac{N_c}{N}$
$\hat{\mu}_c = \frac{1}{N_c}\sum_{i=1}^N \gamma_{ic}\mathbf{x}_i$ by MLE
$\hat{\Sigma}_c = \sum_{i=1}^N \gamma_{ic}(\mathbf{x}_i - \hat{\mu}_c)(\mathbf{x}_i - \hat{\mu}_c)^T$

One challenge, possibility of getting unlikely and overfitting to one point and getting a singularity, but we can imporse a prior on a covariance matrix based on n hallucinated observations. One other challenge, how do we choose k? Validate on a heldout data set or penalizing the model directly.

Probably should look at EM for regression? But tbh, don't remember this lecture...

5 Neural

5.1 Perceptron

Mistake-driven classification algorithm.

Perceptron loss: $\begin{cases} 0 & y_i(\mathbf{w}\cdot\mathbf{x}_i) > 0 \\ y_i\mathbf{x}_i & \text{else} \end{cases}$

Assume linearly separable or else will not stop. $\mathbf{w} = \sum_{i=1}^{M} \alpha_{m(i)}\mathbf{x}_{i(m)}$ where $\alpha_{m(i)}$ is 0 or y_i and $i(m)$ is the index of example used in the mth iteration.

5.2 Neural

Nested functions, backpropagation to perform gradient descent and assign blame from loss.
$\hat{y}(\mathbf{x};\mathbf{w}) = f(\sum_{j=1}^{m} w_j^2 h(\sum_{i=1}^{d} w_{ij}^1 + w_{0j}^1) + w_0^2)$
You define your loss, e.g. squared loss,
Setup: $j \in J$ feeding into t, t feeding into $s \in S$, r also feeding into S.
$z_t = h(a_t) = h(\sum_j w_{jt} z_j)$
$z_s = h^*(a_s) = h^*(\sum_r w_{rs} h(a_r))$
aka $a_t = \sum_j w_{jt} z_j$ and $a_s = \sum_r w_{rs} h(a_r)$

$\delta_t = \frac{\partial L}{\partial a_t} = \sum_s \frac{\partial L}{\partial a_s}\frac{\partial a_s}{\partial a_t} = \sum_s \delta_s (w_{ts} h'(a_t))$
$\partial a_t / \partial w_j t = z_j$
The blame placed on a_t is the sum of losses from nodes it outputs to weighted by its contribution and how much it can change. a_t distributes this loss to its incoming weights, which is dependent on the strength of the input.
$\frac{\partial L(\hat{y},y)}{\partial w_{jt}} = \frac{\partial L}{\partial a_t}\frac{\partial a_t}{\partial w_{jt}} = \delta_t z_j = [\sum_s \frac{\partial L}{\partial s}(w_{ts} h'(a_t))] z_j$

Regularization: weight decay (L2 reg), early stopping.
Stability of GD: Apply momentum by weighting the previous weight. $\Delta\mathbf{w}_t = \mu\Delta\mathbf{w}_{t+1} - \eta_t \nabla_w J$
Vanishing gradient due to saturating nonlinearity: ReLU $max(0,a)$
Co-adaptation aka upper levels learn the same things: use dropout to randomly remove units.

6 Regression

6.1 Linear

Fits linear line to data using squared loss. Has closed form.
$f(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} = w_0 + \mathbf{w}\cdot\mathbf{x}$

Squared loss; $L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2$;
$= \frac{1}{N}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$
$= \frac{1}{N}[\mathbf{y}^T\mathbf{y} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}]$

ERM
$\frac{\partial L}{\partial \mathbf{w}} = \frac{-2}{N}[\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w}] = 0$
$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

MLE on Gaussian noise model,
Discriminative, assumption that joint distribution is a function plus some noise, $y = f(\mathbf{x};\mathbf{w}) + \nu$.
$p(y|\mathbf{x};\mathbf{w},\sigma) = \mathcal{N}(y; f(\mathbf{x};\mathbf{w}), \sigma^2)$
$= \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(y-f(\mathbf{x};\mathbf{w}))^2}{2\sigma^2}\right)$
$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$
$\hat{\sigma}^2 = (1/N)\sum_{i=1}^{N}(y - f(\mathbf{x};\mathbf{w}))^2$
This shows us that maximizing log likelihood is always equivalent to minimizing log loss, and maximizing log likelihood under the Gaussian noise model is the same as minimizing squared loss.

MAP on Gaussian noise model, surprisingly the same as L2 regularization.
$\hat{\mathbf{w}} = (\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$
L2 ridge regularization by ERM.
$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}}\{\sum_{i=1}^{N}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2 - \lambda\sum_{j=1}^{d} w_j^2\}$

L1 lasso regularization
$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}}\{\sum_{i=1}^{N}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2 - \lambda\sum_{j=1}^{d} |w_j|\}$
Lasso prefers sparsity. Eliminates smallest norm first.

Errors uncorrelated with training data
Errors are from LSQ regression $\hat{w}$ because we are performing ERM, so expected loss is 0.
$\frac{\partial R}{\partial \hat{\mathbf{w}}} = \int_x \int_y (y - \hat{\mathbf{w}}\cdot\mathbf{x}) p(\mathbf{x},y) d\mathbf{x} dy = 0$

Best unrestricted predictor
$f^*(\mathbf{x}_0) = E_{p(y|\mathbf{x})}[y_0|\mathbf{x}_0]$ by chain rule. Since each point is equally likely, we can just minimize the inner conditional wrt f.

Bias-variance
Let $\overline{\theta} = E[\hat{\theta}]$
$E[(\hat{\theta} - \theta)^2] = E[(\hat{\theta} - \overline{\theta} + \overline{\theta} - \theta)^2]$
$= E[(\hat{\theta} - \overline{\theta})^2] + (\overline{\theta} - \theta)^2 = var + bias^2$
Bias is the approximation error limitation of your hypothesis class. Some bias attributed to noise. **Variance** is estimation error how far you are from best in class due to finite data. The more you regularize, the more vairance, but the less the bias.

6.2 Logistic

Linar classification, use -log p as loss as surr.

$h(x) = sign(w_0 + \mathbf{w}\cdot\mathbf{x})$
$l(\mathbf{w}) = -\log p(\mathbf{Y}|\mathbf{X};\mathbf{w})$, $\sigma(w_0 + \mathbf{w}\cdot\mathbf{x}_i) - y_i$

Really want to minimize risk from 0/1 loss.
$R(h|\mathbf{x}) = \sum_{c=1}^{C} L_{0/1}(h(\mathbf{x}), c) p(y = c|\mathbf{x})$
$= 1 - p(y = h(\mathbf{x})|\mathbf{x})$
Model the log odds ratio,
$h(\mathbf{x}) = c^*$ iff $\log\frac{p(y=c^*|\mathbf{x})}{p(y=c|\mathbf{x})} = w_0 + \mathbf{w}\cdot\mathbf{x} = 0$
$p(y = 1|\mathbf{x}) = \frac{1}{1+\exp(-w_0 - \mathbf{w}\cdot\mathbf{x})} = \sigma(w_0 + \mathbf{w}\cdot\mathbf{x})$
$\log p(\mathbf{Y}|\mathbf{X}) = \log\prod_{i=1}^{N}\sigma^{y_i}(1-\sigma)^{1-y_i}$

Softmax
Multiclass classification.
$p(y = c|x) = \frac{\exp(\mathbf{w}_c\cdot\mathbf{x} - a)}{\sum_{k=1}^{C}\exp(\mathbf{w}_k\cdot\mathbf{x} - a)}$
Minus a for overflow, since posterior is invariant to shifting scores. E.g. let a be the max score.

6.3 Misc

GD for linear:
$\mathbf{w}' = \mathbf{w} - \eta(\mathbf{y} - f(\mathbf{X};\mathbf{w}))$
GD for logistic:
$\mathbf{w}' = \mathbf{w} - \eta(\sum_{i=1}^{N}(\sigma(w_0 + \mathbf{w}\cdot\mathbf{x}_i) - y_i))[1, \mathbf{x}_i]^T$
Feature maps
Still linear, but in a higher dimension. $\phi(\mathbf{x})$. Exponential compleixity though.

7 Misc.

Assumption that train and test x, y drawn iid from joint probability $p(x,y)$.
Loss: $l : \mathcal{Y}, \mathcal{Y} \to \mathbb{R}$
Risk: $E_{(\mathbf{x}_0, y_0)\sim p(\mathbf{x},y)}[l(f(\mathbf{x}_0;\mathbf{w}), y_0)]$
Bayes Risk: $f^* = \arg\min_{f:\mathcal{X}\to\mathbb{R}} R(f)$
ERM assumes that training set is representative of the underlying distribution, so the empirical loss serves as a proxy for the risk.
$\hat{f} = \arg\min_f l(y, f(\mathbf{x}))$
Generative approach normalizes $p(\mathbf{x}, y)$ using Bayes to get $p(y|\mathbf{x})$.
$p(y|\mathbf{x}) = p(\mathbf{x}|y)p(y)/p(\mathbf{x})$ post = likelihood * prior
Discriminative approach estimates $p(y|\mathbf{x})$ directly from the data. MLE and MAP. MLE performs point estimation on the highest likelihood parameters. MAP uses some belief about parameter before seeing data.
$MLE = \max\log[p(y|\mathbf{x};\theta)]$
$MAP = \max\log[p(y|\mathbf{x},\theta)p(\theta)]$