CS 189 Final Note Sheet

Rishi Sharma and Peter Gao

Bayesian Decision Theory

Bayes Rule: $P(\omega|x) = \frac{P(x|\omega)P(\omega)}{P(x)}, P(x) = \sum_i P(x|\omega_i)P(\omega_i)$

$P(x,w) = P(x|w)P(w) = P(w|x)P(x)$

$P(error) = \int_{-\infty}^{\infty} P(error|x)P(x)dx$

$P(error|x) = \begin{cases} P(\omega_1|x) & \text{if we decide } \omega_2 \\ P(\omega_2|x) & \text{if we decide } \omega_1 \end{cases}$

0-1 Loss: $\lambda(\alpha_i|\omega_j) = \begin{cases} 0 & i = j \text{ (correct)} \\ 1 & i \neq j \text{ (mismatch)} \end{cases}$

Expected Loss (Risk): $R(\alpha_i|x) = \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j)P(\omega_j|x)$

0-1 Risk: $R(\alpha_i|x) = \sum_{j\neq i}^{c} P(\omega_j|x) = 1 - P(\omega_i|x)$

Probabilistic Motivation for Least Squares

$y^{(i)} = \theta^\intercal x^{(i)} + \epsilon^{(i)}$ with noise $\epsilon(i) \sim \mathcal{N}(0,\sigma^2)$

Note: The intercept term $x_0 = 1$ is accounted for in θ

$\implies p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)}-\theta^\intercal x^{(i)})^2}{2\sigma^2}\right)$

$\implies L(\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)}-\theta^\intercal x^{(i)})^2}{2\sigma^2}\right)$

$\implies l(\theta) = m \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y^{(i)} - \theta^\intercal x^{(i)})^2$

$\implies \max_\theta l(\theta) \equiv \min_\theta \sum_{i=1}^{m}(y^{(i)} - h_\theta(x))^2$

Gaussian noise in our data set $\{x^{(i)}, y^{(i)}\}_{i=1}^{m}$gives us least squares

$min_\theta ||X\theta - y||_2^2 \equiv \min_\theta \theta^\intercal X^\intercal X\theta - 2\theta^\intercal X^\intercal y + y^\intercal Y$

$\nabla_\theta l(\theta) = X^\intercal X\theta - X^\intercal y = 0 \implies \boxed{\theta^* = (X^\intercal X)^{-1}X^\intercal y}$

Gradient Descent: $\theta_{t+1} = \theta_t + \alpha(y_t^{(i)} - h(x_t^{(i)}))x_t^{(i)}, \quad h_\theta(x) = \theta^\intercal x$

Least Squares Solution

Min norm soln: $\min_x ||Ax - y||_2^2 \implies x^* = A^\dagger y$

Soln set: $x_0 + N(A) = x^* + N(A)$

$$A^\dagger = \begin{cases} (A^\intercal A)^{-1}A^\intercal & A \text{ full column rank} \\ A^\intercal(AA^\intercal)^{-1} & A \text{ full row rank} \\ V\Sigma^\dagger U^T & \text{any } A \end{cases}$$

L2 Reg: $\min_x ||Ax - y||_2^2 + \lambda||x||_2^2 \implies x^* = (A^TA + \lambda I)^{-1}X^Ty$

The above variant is used when A contains a null space. L2 Reg falls out of the MLE when we add a Gaussian prior on x with $\Sigma = cI$. We get L1 Reg when x has a Laplace prior.

Logistic Regression

Classify $y \in \{0,1\} \implies$ Model $p(y=1|x) = \frac{1}{1+e^{-\theta^T x}} = h_\theta(x)$

$\frac{dh_\theta}{d\theta} = (\frac{1}{1+e^{\theta^T x}})^2 e^{-\theta^T x} = \frac{1}{1+e^{\theta^T x}}\left(1 - \frac{1}{1+e^{-\theta^T x}}\right) = h_\theta(1-h_\theta)$

$p(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y} \implies$

$L(\theta) = \prod_{i=1}^{m}(h_\theta(x^{(i)}))^{y^{(i)}}(1-h_\theta(x^{(i)}))^{1-y^{(i)}} \implies$

$l(\theta) = \sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)})) \implies$

$\nabla_\theta l = \sum_i (y^{(i)} - h_\theta(x^{(i)}))x^{(i)} = X^\intercal(y - h_\theta(X))$, (want max $l(\theta)$)

Stochastic: $\boxed{\theta_{t+1} = \theta_t + \alpha(y_t^{(j)} - h_\theta(x_t^{(j)}))x_t^{(j)}}$

Batch: $\boxed{\theta_{t+1} = \theta_t + \alpha X^\intercal(y - h_\theta(X))}$

Multivariate Gaussian $X \sim \mathcal{N}(\mu,\Sigma)$

$f(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

$\Sigma = E[(X-\mu)(X-\mu)^T] = E[XX^T] - \mu\mu^T$

Σ is PSD $\implies x^T\Sigma x \geq 0$, if inverse exists Σ must be PD

If $X \sim N(\mu,\Sigma)$, then $AX + b \sim N(A\mu + b, A\Sigma A^T)$

$\implies \Sigma^{-\frac{1}{2}}(X-\mu) \sim N(0,I)$, where $\Sigma^{-\frac{1}{2}} = U\Lambda^{-\frac{1}{2}}$

The distribution is the result of a linear transformation of a vector of univariate Gaussians $Z \sim \mathcal{N}(0,I)$ such that $X = AZ + \mu$ where we have $\Sigma = AA^\intercal$. From the pdf, we see that the level curves of the distribution decrease proportionally with $x^\intercal\Sigma^{-1}x$ (assume $\mu = 0$) $\implies$

$$c\text{-level set of } f \propto \{x : x^\intercal\Sigma^{-1}x = c\}$$

$$x^\intercal\Sigma^{-1} = c \equiv x^\intercal U\Lambda^{-1}U^\intercal x = c \implies$$

$$\underbrace{\lambda_1^{-1}(u_1^\intercal x)^2}_{\text{axis length: } \sqrt{\lambda_1}} + \cdots + \underbrace{\lambda_n^{-1}(u_n^\intercal x)^2}_{\text{axis length: } \sqrt{\lambda_n}} = c$$

Thus the level curves form an ellipsoid with axis lengths equal to the square root of the eigenvalues of the covariance matrix.

LDA and QDA

Classify $y \in \{0,1\}$, Model $p(y) = \phi^y\phi^{1-y}$ and

$l(\theta,\mu_0,\mu_1,\Sigma) = log\ \Pi_{i=1}^{m}p(x^{(i)}|y^{(i)};\mu_0,\mu_1,\Sigma)p(y^{(i)};\Phi)$ gives us

$\phi_{MLE} = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}$,$\mu_{k_{MLE}}$ = avg of $x^{(i)}$ classified as k,

$\Sigma_{MLE} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_{y_{(i)}})(x^{(i)} - \mu_{y_{(i)}})^T$.

Notice the covariance matrix is the same for all classes in LDA.

If $p(x|y)$ multivariate gaussian (w/ shared Σ), then $p(y|x)$ is logistic function. The converse is NOT true. LDA makes stronger assumptions about data than does logistic regression.

$h(x) = arg\max_k -\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) + log(\pi_k)$

where $\pi_k = p(y=k)$

For QDA, the model is the same as LDA except that each class has a unique covariance matrix.

$h(x) = arg\max_k -\frac{1}{2}log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + log(\pi_k)$

Optimization

Newton's Method: $\theta_{t+1} = \theta_t - [\nabla_\theta^2 f(\theta_t)]^{-1}\nabla_\theta f(\theta_t)$

Gradient Decent: $\theta_{t+1} = \theta_t - \alpha\nabla_\theta f(\theta_t)$, for minimizing

Lagrange Multipliers:

Given $\min_x f(x)\ s.t.\ g_i(x) = 0,\ h_i(x) \leq 0$, the corresponding Lagrangian is: $L(x,\alpha,\beta) = f(x) + \sum_{i=1}^{k}\alpha_i g_i(x) + \sum_{l}^{l}\beta_i h_i(x)$

We min over x and max over the Lagrange multipliers α and β

Support Vector Machines

In the strictly separable case, the goal is to find a separating hyperplane (like logistic regression) except now we don't just want any hyperplane, but one with the largest margin.

$H = \{\omega^Tx + b = 0\}$, since scaling ω and b in opposite directions doesn't change the hyperplane our optimization function should have scaling invariance built into it. Thus, we do it now and define the closest points to the hyperplane x_{sv} (support vectors) to satisfy: $|\omega^Tx_{sv} + b| = 1$. The distance from any support vector to the hyper plane is now: $\frac{1}{||\omega||_2}$. Maximizing the distance to the hyperplane is the same as minimizing $||\omega||_2$.

The final optimization problem is:

$$\boxed{\min_{\omega,b}\frac{1}{2}||\omega||_2\ s.t.\ y^{(i)}(w^Tx^{(i)} + b) \geq 1, i = 1,\ldots,m}$$

Primal: $L_p(\omega,b,\alpha) = \frac{1}{2}||\omega||_2 - \sum_{i=1}^{m}\alpha_i(y^{(i)}(w^Tx^{(i)} + b) - 1)$

$\frac{\partial L_p}{\partial\omega} = \omega - \sum\alpha_i y^{(i)}x^{(i)} = 0 \implies \omega = \sum\alpha_i y^{(i)}x^{(i)}$

$\frac{\partial L_p}{\partial b} = -\sum\alpha_i y^{(i)} = 0$, Note: $\alpha_i \neq 0$ only for support vectors.

Substitute the derivatives into the primal to get the dual.

Dual: $L_d(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^Tx^{(j)}$

KKT says $\alpha_n(y_n(w^Tx_n + b) - 1) = 0$ where $\alpha_n > 0$.

In the non-separable case we allow points to cross the marginal boundary by some amount ξ and penalize it.

$$\boxed{\min_{\omega,b}\frac{1}{2}||\omega||_2 + C\sum_{i=1}^{m}\xi_i\quad s.t.\quad y^{(i)}(w^Tx^{(i)} + b) \geq 1 - \xi_i}$$

The dual for non-separable doesn't change much except that each α_i now has an upper bound of C $\implies 0 \leq \alpha_i \leq C$

Loss Functions

In general the loss function consists of two parts, the loss term and the regularization term. $J(\omega) = \sum_i Loss_i + \lambda R(\omega)$

Nearest Neighbor

Key Idea: Store all training examples $\langle x_i, f(x_i)\rangle$

NN: Find closest training point using some distance metric and take its label.

k-NN: Find closest k training points and take on the most likely label based on some voting scheme (mean, median,...)

Behavior at the limit: 1NN $lim_{N\to\infty}\ \epsilon^* \leq \epsilon_{NN} \leq 2\epsilon^*$

ϵ^* = error of optimal prediction, ϵ_{nn} = error of 1NN classifier

KNN $lim_{N\to\infty,K\to\infty}, \frac{K}{N} \to 0, \epsilon_{knn} = \epsilon^*$

Curse of dimensionality: As the number of dimensions increases, everything becomes farther apart. Our low dimension intuition falls apart. Consider the Hypersphere/Hypercube ratio, it's close to zero at $d = 10$. How do deal with this curse:

1. Get more data to fill all of that empty space
2. Get better features, reducing the dimensionality and packing the data closer together. Ex: Bag-of-words, Histograms,...
3. Use a better distance metric.

Minkowski: $Dis_p(x,y) = (\sum_{i=1}^{d}|x_i - y_u|^p)^{\frac{1}{p}} = ||x - y||_p$

0-norm: $Dis_0(x,y) = \sum_{i=1}^{d} I|x_i = y_i|$

Mahalanobis: $Dis_M(x,y|\Sigma) = \sqrt{(x-y)^T\Sigma^{-1}(x-y)}$

In high-d we get "Hubs" s.t most points identify the hubs as their NN. These hubs are usually near the means (Ex: dull gray images, sky and clouds). To avoid having everything classified as these hubs, we can use cosine similarity.

K-d trees increase the efficiency of nearest neighbor lookup.

Gradients

$$\frac{\partial\mathbf{y}}{\partial\mathbf{x}} \triangleq \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}, \frac{\partial(A\mathbf{x})}{\partial\mathbf{x}} = A^T, \frac{\partial(\mathbf{x}^TA)}{\partial\mathbf{x}} = A,$$

$\frac{\partial(\mathbf{x}^T\mathbf{x})}{\partial\mathbf{x}} = 2\mathbf{x}, \frac{\partial(\mathbf{x}^TA\mathbf{x})}{\partial\mathbf{x}} = (A + A^T)\mathbf{x}, \frac{\partial(trBA)}{\partial A} = B^T$

Generative vs. Discriminative Model

Generative: Model class conditional density $p(x|y)$ and find $p(y|x) \propto p(x|y)p(y)$ or model joint density $p(x,y)$ and marginalize to find $p(y=k|x) = \int_x p(x,y=k)dx$

Discriminative: Model conditional $p(y|x)$.

Decision Trees

Given a set of points and classes $\{x_i, y_i\}_{i=1}^n$, test features x_j and branch on the feature which "best" separates the data. Recursively split on the new subset of data. Growing the tree to max depth tends to overfit (training data gets cut quickly $\implies$ subtrees train on small sets). Mistakes high up in the tree propagate to corresponding subtrees. To reduce overfitting, we can prune using a validation set, and we can limit the depth.

DT's are prone to label noise. Building the correct tree is hard.

Heurisitic: For classification, maximize information gain

$$\max_j \quad \mathrm{H}(D) - \sum_{x_j \in X_j} P(X_j = x_j) \cdot \mathrm{H}(D|X_j = x_j)$$

where $\mathrm{H}(D) = -\sum_{c \in C} P(y = c) \log[p(y = c)]$ is the entropy of the data set, C is the set of classes each data point can take, and $P(y = c)$ is the fraction of data points with class c.

For REGRESSION, minimize the variance. Same optimization problem as above, except H is replaced with var. Pure leaves correspond to low variance, and the result is the mean of the current leaf.

Random Forests

Problem: DT's are unstable: small changes in the input data have large effect on tree structure $\implies$ DT's are high-variance estimators.

Solution: Random Forests train M different trees with randomly sampled subsets of the data (called bagging), and sometimes with randomly sampled subsets of the features to de-correlate the trees. A new point is tested on all M trees and we take the majority as our output class (for regression we take the average of the output).

Boosting

Weak Learner: Can classify with at least 50% accuracy.

Train weak learner to get a weak classifier. Test it on the training data, up-weigh misclassified data, down-weigh correctly classified data. Train a new weak learner on the weighted data. Repeat. A new point is classified by every weak learner and the output class is the sign of a weighted avg. of weak learner outputs. Boosting generally overfits. If there is label noise, boosting keeps upweighing the mislabeled data.

AdaBoost is a boosting algorithm. The weak learner weights are given by $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$ where $\epsilon_t = Pr_{D_t}(h_t(x_i) \neq y_i)$ (probability of misclassification). The weights are updated $D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor.

Clustering

Unsupervised Learning (no labels).

Hierarchical:

- Agglomerative: Start with n points, merge 2 closest clusters using some measure, such as: Single-link (closest pair), Complete-link (furthest pair), Average-link (average of all pairs), Centroid (centroid distance).
  Note: SL and CL are sensitive to outliers.
- Divisive: Start with single cluster, recursively divide clusters into 2 subclusters.

Partitioning: Partition the data into a K mutually exclusive exhaustive groups (i.e. encode k=C(i)). Iteratively reallocate to minimize some loss function. Finding the correct partitions is hard. Use a greedy algorithm called K-means (coordinate decent). Loss function is non-convex thus we find local minima.

K-means: Choose clusters at random, calculate centroid of each cluster, reallocate objects to nearest centroid, repeat.

K-means++: Initialize clusters one by one. D(x) = distance of point x to nearest cluster. Pr(x is new cluster center)$\propto D(x)^2$

K-medians: Works with arbitrary distance/dissimilarity metric, the centers μ_k are represented by data points. Is more restrictive thus has higher loss.

General Loss: $\sum_{n=1}^N \sum_{k=1}^K d(x_n, \mu_k) r_{nk}$ where $r_{nk} = 1$ if x_n is in cluster k, and 0 o.w.

Vector Quantization

Use clustering to find representative prototype vectors, which are used to simplify representations of signals.

Parametric Density Estimation

(Mixture Models). Assume PDF is made up of multiple gaussians with different centers. $P(x) = \sum_{i=1}^{n_c} P(c_i)P(x|c_i)$ with objective function as log likelihood of data. Use EM to estimate this model.

E Step: $P(\mu_i|x_k) = \frac{P(\mu_i)P(x_k|\mu_i)}{\sum_j P(\mu_j)P(x_j|\mu_j)}$

M Step: $P(c_i) = \frac{1}{n_e}\sum_{k=1}^{n_e} P(\mu_i|x_k)$

$\mu_i = \frac{\sum_k x_k P(\mu_i|x_k)}{\sum_k P(\mu_i|x_k)}$

$\sigma_i^2 = \frac{\sum_k (x_k - \mu_i)^2 P(\mu_i|x_k)}{\sum_k P(\mu_i|x_k)}$.

Non-parametric Density Estimation

Can use Histogram or Kernel Density Estimation (KDE).

KDE: $P(x) = \frac{1}{n}\sum K(\mathbf{x} - \mathbf{x_i})$ is a function of the data.

The kernel K has the following properties:

Symmetric, Normalized $\int_{\mathbb{R}^d} K(x)dx = 1$, and $\lim_{||x|| \to \infty} ||x||^d K(x) = 0$.

The bandwidth is the width of the kernel function. Too small = jagged results, too large = smoothed out results.

Mode Seeking

To find "Bumps" in the distribution, we can just estimate the gradient.

Mean Shift: Translate kernel window by m(x).

$$m(x) = \left[\frac{\sum_{i=1}^n x_i g(\frac{||x-x_i||^2}{h})}{\sum_{i=1}^n g(\frac{||x-x_i||^2}{h})} - x\right]$$

Where g is a constant Kernel with radius R (Can use any Kernel for generalized results).

To deal with saddle points, perturb modes and prune by taking the highest mode.

Pro's:

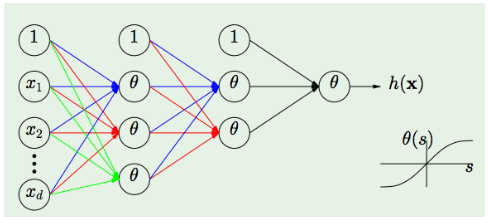
- Auto convergence speed, near maxima the steps are small.
- Only one parameter to choose (window size).
- Does not assume prior shape on data.

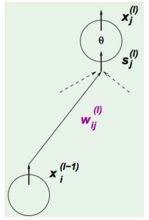
Con's:

- The window size is non-trivial, and incorrect window size can cause modes to be merged (or can cause additional "shallow" modes to be generated)

Neural Networks

Neural Nets explore what you can do by combining perceptrons, each of which is a simple linear classifier. We use a soft threshold for each activation function θ because it is twice differentiable.





Activation Functions:

$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} \implies \theta'(s) = 1 - \theta^2(s)$

$\theta(s) = \sigma(s) = \frac{1}{1+e^{-s}} \implies \theta'(s) = \sigma(s)(1 - \sigma(s))$

Error Functions:

Cross Entropy Loss $\sum_{i=1}^{n_{out}} y \log(h_\theta(x)) + (1-y)\log(1 - h_\theta(x))$

Mean Squared Error $\sum_{i=1}^{n_{out}} (y - h_\theta(x))^2$

Notation:

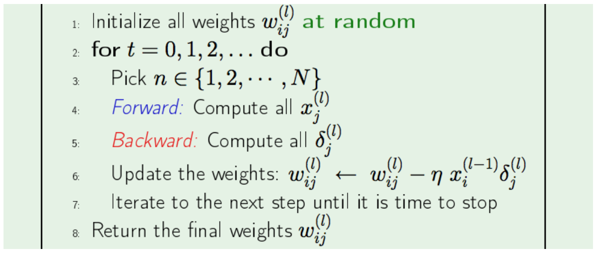
1. $w_{ij}^{(l)}$ is the weight from neuron i in layer $l-1$ to neuron j in layer l. There are $d^{(l)}$ nodes in the l^{th} layer.
2. L layers, where L is output layer and data is 0th layer.
3. $x_j^{(l)} = \theta(s_j^{(l)})$ is the output of a neuron. It's the activation function applied to the input signal. $s_j^{(l)} = \sum_i w_{ij}^{(l)} x_i^{(l-1)}$
4. $e(w)$ is the error as a function of the weights

The goal is to learn the weights $w_{ij}^{(l)}$. We use gradient descent, but error function is non-convex so we tend to local minima. The naive version takes $O(w^2)$. Back propagation, an algorithm for efficient computation of the gradient, takes $O(w)$.

$$\nabla e(w) \to \frac{\partial e(w)}{\partial w_{ij}^{(l)}} = \frac{\partial e(w)}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} x_i^{(l-1)}$$

Final Layer: $\delta_j^{(L)} = \frac{\partial e(w)}{\partial s_j^{(l)}} = \frac{\partial e(w)}{\partial x_j^{(L)}} \frac{\partial x_j^{(L)}}{\partial s_j^{(L)}} = e'(x_j^{(L)})\theta'_{out}(s_j^L)$

$$\text{General: } \delta_i^{(l-1)} = \frac{\partial e(w)}{\partial s_i^{(l-1)}} = \sum_{j=1}^{d^{(l)}} \frac{\partial e(w)}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}} = \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})$$



```
1: Initialize all weights $w_{ij}^{(l)}$ at random
2: for $t = 0, 1, 2, \ldots$ do
3:     Pick $n \in \{1, 2, \cdots, N\}$
4:     Forward: Compute all $x_j^{(l)}$
5:     Backward: Compute all $\delta_j^{(l)}$
6:     Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\, x_i^{(l-1)} \delta_j^{(l)}$
7:     Iterate to the next step until it is time to stop
8: Return the final weights $w_{ij}^{(l)}$
```

CS 189 Final Note Minicards

Che Yeon // Li // Sean

SVM-like classifiers work with a boundary, a hyperplane (a line for 2D data) that separates two classes. Support vectors are the point(s) closest to the boundary. γ is the margin, the distance between the boundary and the support vector(s). The parameter θ is a vector. $\boxed{\theta \cdot x}$ gives predictions. About θ:

- The direction of θ defines the boundary. We can choose this.
- $\|\theta\|$ must be $1/\gamma$, as restricted by $\forall i : y^i \theta \cdot x^i \geq 1$
  We cannot explicitly choose this; it depends on the boundary. This restriction is turned into a cost in soft-margin SVM.

Perceptron [2:11, 3:6] picks misclassified point and updates θ just enough to classify it correctly:

$\boxed{\theta \leftarrow \theta + x^i}$ or $\boxed{\theta \leftarrow \theta - \nabla J(\theta)}$

Overfits when outliers skew the boundary. Converges iff separable.
Batch eqn $\theta \cdot x = \sum_i \alpha^i y^i x^i \cdot x$,
where $\alpha = \#$ times point was misclassified

Hard-margin SVM [3:36] maximizes the margin around the boundary. Technically, it minimizes the distance between boundary and the vectors closest to it (the support vectors):

$$\boxed{\min_\theta \|\theta\|^2 \quad \text{such that } \forall i : y^i \theta \cdot x^i \geq 1}$$

Sometimes removing a few outliers lets us find a much higher margin or a margin at all. Hard-margin overfits by not seeing this.
Converges iff separable.
Batch eqn $\theta = \sum_i \alpha^i y^i x^i$, where $\alpha^i = \mathbf{1}_{i \text{ is support vector}}$

Soft-margin SVM [3:37] is like hard-margin SVM but penalizes misclassifications:

$$\boxed{\min_\theta \|\theta\|^2 + C \sum_{i=1}^{n} \left(1 - y^i \theta \cdot x^i\right)_+}$$

Hyperparameter C is the hardness of the margin. Lower C means more misclassifications but larger soft margin.
Overfits on less data, more features, higher C

Regression

Linear regression [9:7]

Lasso vs ridge regression
which one yields sparse results?

More classifiers

KNN [14:4] Given an item x, find the k training items "closest" to x and return the result of a vote.
Hyperparameter k, the number of neighbors.
"Closest" can be defined by some norm (l_2 by default).
Overfits when k is really small

Decision trees: Recursively split on features that yield the best split. Each tree has many nodes, which either split on a feature at a threshold, or all data the same way.
Hyperparameters typically restrict complexity (max tree depth, min points at node) or penalize it. One particular one of interest is d, the max number of nodes.
Overfits when tree is deep or when we are allowed to split on a very small number of items.
Bagging: Make multiple trees, each with a random subset of training items. To predict, take vote from trees.
Hyperparameters # trees, proportion of items to subset.
Random forests is bagging, except, for each node, consider only a random subset of features to split on.
Hyperparameters proportion of features to consider.

AdaBoost [dtrees3:34] Use any algorithm (i.e., decision trees) to train a weak learner, take all the errors, and train a new learner on with the errors emphasized*. To predict, predict with the first algorithm, then add on the prediction of the second algorithm, and so on.

* For regression, train the new learner on the errors.
For classification, give misclassified items more weight.

Hyperparameters B, the number of weak learners; λ, the learning rate.

Things to do

Spectral Theorem

- generative vs discriminative
- estimating post vs prior
- early slides
- cry