CS 189 Final Note Sheet – Rishi Sharma, Peter Gao, et. al.

Probability & Matrix Review

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

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 $p(y=k|x) = \int_x p(x,y=k)dx$ (posterior)

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

class conditional $P(X\|Y)$	posterior $P(Y\|X)$
prior $P(Y)$	evidence $P(X)$

Probabilistic Motivation for Least Squares

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

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

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

Gaussian class conditionals lead to a logistic posterior.

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

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.

$H = \{\omega^T x + b = 0\}$, 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: $|\omega^T x_{sv} + b| = 1$. 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:

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

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

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

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

Substitute the derivatives into the primal to get the dual.

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

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

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

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

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

Lagrangian

$$\boxed{L(x,\lambda) = f_0(x) + \sum_{i=1}^{m}\lambda_i f_i(x)}$$

- Think of the λ_i as the cost of violating the constraint $f_i(x) \leq 0$.
- L defines a saddle point game: one player (MIN); the other player (MAX) chooses λ to maximize L. If MIN violates a constraint, $f_i(x) > 0$, then MAX can drive L to infinity.
- We call the original optimization problem the primal problem. It has value $p* = \min_x \max_{\lambda\geq 0} L(x,\lambda)$
  (Because of an infeasible x, $L(x,\lambda)$ can be made infinite, and for a feasible x, the $\lambda_i f_i(x)$ terms will become zero.)
- Define $g(\lambda) := \min_x L(x,\lambda)$, and define the dual problem as $d* = \max_{\lambda\geq 0} g(\lambda) = \max_{\lambda\geq 0}\min_x L(x,\lambda)$
- In a zero sum game, it's always better to play second: $p* = \min_x \max_{\lambda\geq 0} L(x,\lambda) \geq \max_{\lambda\geq 0}\min_x L(x,\lambda) = d*$This

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

Other Classifiers

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 $lim_{N\to\infty}\, \varepsilon^* \leq \varepsilon_{NN} \leq 2\varepsilon^*$
ε^* = error of optimal prediction, ε_{nn} = error of 1NN classifier

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

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.

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

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 $w_{ij}^{(l)}$

Unsupervised Learning

Clustering

Unsupervised learning (no labels).

Distance functions. Suppose we have two sets of points.

- **Single linkage** is minimum distance between members.
- **Complete linkage** is maximum distance between members.
- **Centroid linkage** is distance between centroids.
- **Average linkage** is average distance between all pairs.

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

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-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.

Loss Functions

- **Binomial deviance** $= \log\left[1 + e^{-yf(x)}\right]$
  minimizing function $f(x) = \log \frac{\mathrm{P}[Y=+1|x]}{\mathrm{P}[Y=-1|x]}$
- **SVM hinge loss** $= [1 - yf(x)]_+$
  minimizing function $f(x) = \mathrm{sign}\left(\mathrm{P}[Y=+1 \mid x] - \frac{1}{2}\right)$
- **Squared error** $= [y - f(x)]^2 = [1 - yf(x)]^2$
  minimizing function $f(x) = 2\mathrm{P}[Y=+1 \mid x] - 1$
- **"Huberized" square hinge loss**
  $= \begin{cases} -4yf(x) & \text{if } yf(x) < -1 \\ [1-yf(x)]_+^2 & \text{otherwise} \end{cases}$
  minimizing function $f(x) = 2\mathrm{P}[Y=+1 \mid x] - 1$

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

Gradients

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

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

is called weak duality.

- If there is a saddle point $(x*, \lambda*)$, so that for all x and $\lambda \geq 0$, $L(x*, \lambda) \leq L(x*, \lambda*) \leq L(x, \lambda*)$, then we have strong duality: the primal and dual have the same value, $p* = \min_x \max_{\lambda \geq 0} L(x, \lambda) = \max_{\lambda \geq 0} \min_x L(x, \lambda) = d*$

Using notation from Peter's notes:

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

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

Regression

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

L2 regularization results in **ridge regression**.
Used when A contains a null space. L2 reg falls out of the MLE when we add a Gaussian prior on x with $\Sigma = cI$.
$\min_x ||Ax - y||_2^2 + \lambda ||x||_2^2 \implies x^* = (A^T A + \lambda I)^{-1} X^T y$

L1 regularization results in **lasso regression**.
Used when x has a Laplace prior. Gives sparse results.

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

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).

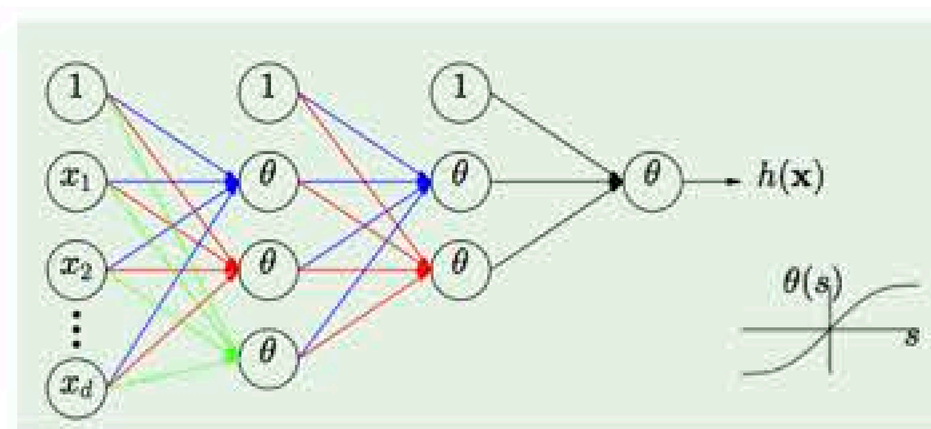
Boosting

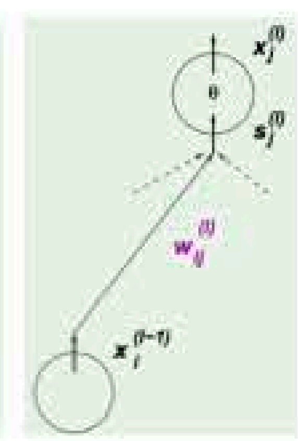
Weak Learner: Can classify with at least 50% accuracy.

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 $\alpha_t = \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$ where $\varepsilon_t = Pr_{D_t}(h_t(x_i) \neq y_i)$ (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.

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:

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

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

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. Works with: spherical, well-separated clusters of similar volumes and count.
- **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.

Principal Component Analysis

First run **singular value decomposition** on pattern matrix X:

1. Subtract mean from each point
2. (Sometimes) scale each dimension by its variance
3. Compute covariance $\Sigma = X^T X$ (must be symmetric)
4. Compute eigenvectors/values $\Sigma = VSV^\intercal$ (spectral thm)
5. Get back $X = X\Sigma = (XV)SV^\intercal = USV^\intercal$

S contains the eigenvalues of the transformed features. The larger the S_{ii}, the larger the variance of that feature. We want the k largest features, so we find the indices of the k largest items in S and we keep only these entries in U and V.

CS 189 ALL OF IT – Che Yeon, Chloe, Dhruv, Li, Sean

Past Exam Questions

Spring 2013 Midterm

(a) False: In SVMs, we maximize $\frac{\|w\|^2}{2}$ subject to the margin constraints.
(b) False: In kernelized SVMS, the kernel matrix K has to be positive definite.
(c) True: If two random variables are independent, then they have to be uncorrelated.
(d) False: Isocontours of Gaussian distributions have axes whose lengths are proportional to the eigenvalues of the covariance matrix.
(e) True: The RBF kernel $K\left(x_i,x_j\right)=\exp\left(-\gamma\left\|x_i-x_j\right\|^2\right)$ corresponds to an infinite dimensional mapping of the feature vectors.
(f) True: If (X,Y) are jointly Gaussian, then X and Y are also Gaussian distributed.
(g) True: A function f(x,y,z) is convex if the Hessian of f is positive semi-definite.
(h) True: In a least-squares linear regression problem, adding an L2 regularization penalty cannot decrease the L2 error of the solution w on the training data.
(i) True: In linear SVMs, the optimal weight vector w is a linear combination of training data points.
(j) False: In stochastic gradient descent, we take steps in the exact direction of the gradient vector.
(k) False: In a two class problem when the class conditionals $P[x \mid y=0] and P[x \mid y=1]$ are modeled as Gaussians with different covariance matrices, the posterior probabilities turn out to be logistic functions.
(l) True: The perceptron training procedure is guaranteed to converge if the two classes are linearly separable.
(m) False: The maximum likelihood estimate for the variance of a univariate Gaussian is unbiased.
(n) True: In linear regression, using an L1 regularization penalty term results in sparser solutions than using an L2 regularization penalty term.

Spring 2013 Final

(a) True: Solving a non linear separation problem with a hard margin Kernelized SVM (Gaussian RBF Kernel) might lead to overfitting.
(b) True: In SVMs, the sum of the Lagrange multipliers corresponding to the positive examples is equal to the sum of the Lagrange multipliers corresponding to the negative examples.
(c) False: SVMs directly give us the posterior probabilities $\mathrm{P}(y=1 \mid x)$ and $\mathrm{P}(y=-1 \mid x)$.
(d) False: $V(X)=\mathrm{E}[X]^2-\mathrm{E}[X^2]$
(e) True: In the discriminative approach to solving classification problems, we model the conditional probability of the labels given the observations.
(f) False: In a two class classification problem, a point on the Bayes optimal decision boundary x* always satisfies $\mathrm{P}[y=1 \mid x*]=\mathrm{P}[y=0 \mid x*]$.
(g) True: Any linear combination of the components of a multivariate Gaussian is a univariate Gaussian.
(h) False: For any two random variables $X\sim N\left(\mu_1,\sigma_1^2\right)$ and $Y\sim\mathcal{N}\left(\mu_2,\sigma_2^2\right), X+Y\sim\mathcal{N}\left(\mu_1+\mu_2,\sigma_1^2+\sigma_2^2\right)$.
(i) False: For a logistic regression problem differing initialization points can lead to a much better optimum.
(j) False: In logistic regression, we model the odds ratio $\frac{p}{1-p}$ as a linear function.
(k) True: Random forests can be used to classify infinite dimensional data.

Spring 2014 Final

(a) False: The singular value decomposition of a real matrix is unique.
(b) True: A multiple-layer neural network with linear activation functions is equivalent to one single-layer perceptron that uses the same error function on the output layer and has the same number of inputs.
(c) False: The maximum likelihood estimator for the parameter θ of a uniform distribution over $[0,\theta]$ is unbiased.
(d) True: The k-means algorithm for clustering is guaranteed to converge to a local optimum.
(e) True: Increasing the depth of a decision tree cannot increase its training error.
(f) False: There exists a one-to-one feature mapping ϕ for every valid kernel k.
(g) True: For high-dimensional data data, k-d trees can be slower than brute force nearest neighbor search.
(h) True: If we had infinite data and infinitely fast computers, kNN would be the only algorithm we would study in CS 189.
(i) True: For datasets with high label noise (many data points with incorrect labels, random forests would generally perform better than boosted decision trees.

(a) In Homework 4, you fit a logistic regression model on spam and ham data for a Kaggle Comp. Assume you had a very good score on the public test set, but when the GSIs ran your model on a private test set, your score dropped a lot. This is likely because you overfitted by submitting multiple times and changing the following between submissions: λ, your penalty term; ε, your convergence criterion; your step size; fixing a random bug.
(b) Given d-dimensional data $\{x_i\}_{i=1}^N$, you run principal component analysis and pick P principal components. Can you always reconstruct any data point x_i for i from 1 to N from the P principal components with zero reconstruction error? Yes, if $P=d$.
(c) Putting a standard Gaussian prior on the weights for linear regression ($w\sim N(0,I)$) will result in what type of posterior distribution on the weights? Gaussian.
(d) Suppose we have N instances of d-dimensional data. Let h be the amount of data storage necessary for a histogram with a fixed number of ticks per axis, and let k be the amount of data storage necessary for kernel density estimation. Which of the following is true about h and k? h grows exponentially with d, and k grows linearly with N.
(e) John just trained a decision tree for a digit recognition. He notices an extremely low training error, but an abnormally large test error. He also notices that an SVM with a linear kernel performs much better than his tree. What could be the cause of his problem? Decision tree is too deep; decision tree is overfitting.
(f) John has now switched to multilayer neural networks and notices that the training error is going down and converges to a local minimum. Then when he test on the new data, the test error is abnormally high. What is probably going wrong and what do you recommend him to do? The training data size is not large enough so collect a larger training data and retain it; play with learning rate and add regularization term to objective function; use a different initialization and train the network several times and use the average of predictions from all nets to predict test data; use the same training data but use less hidden layers.

Spring 2015 Midterm

(a) True: If the data is not linearly separable, there is no solution to hard margin SVM.
(b) True: logistic regression can be used for classification

Discussion Problems

Discussion 9 – Entropy

Solution:

For simplicity, assume that log has base e, i.e. log = ln (the solution is the same no matter what base we assume). The optimization problem we are trying to solve is:

$$\underset{\vec{p}}{\operatorname{argmin}} \sum_{i=1}^{d} p_i \log p_i$$
$$\text{s.t.} \sum_{i=1}^{d} p_i = 1$$

Formulating the Lagrangian, we get

$$\mathcal{L}(\vec{p},\lambda)=\sum_{i=1}^{d} p_i \log p_i + \lambda\left(1-\sum_{i=1}^{d} p_i\right)$$

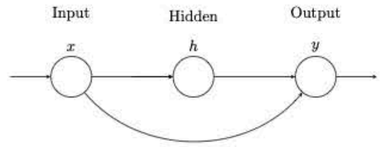
Taking the derivative w.r.t. p_i and λ:

$$\frac{\partial}{\partial p_i}\mathcal{L}(\vec{p},\lambda)=\log p_i+\frac{p_i}{p_i}-\lambda \implies \lambda-1=\log p_i$$
$$\frac{\partial}{\partial \lambda}\mathcal{L}(\vec{p},\lambda)=\sum_{i=1}^{d} p_i = 1$$

This says that $\log p_i = \log p_j, \forall i,j$, which implies that $p_i = p_j, \forall i,j$. Combining this with the constraint, we get that $p_i=\frac{1}{d}$, which is the uniform distribution.

Discussion 11 – Skip-Layer NN

Consider the simplest skip-layer neural network pictured below. The weights are $w=[w_{xh},w_{hy},w_{xy}]^T$.



Given some non-linear function g, calculate $\nabla_w y$. Don't forget to use δ's.

Solution: The output y is given by the function:

$$y=g(s_y)=g(w_{hy}h+w_{xy}x)=g(w_{hy}g(s_h)+w_{xy}x)=g(w_{hy}g(w_{xh}x)+w_{xy}x)$$

To calculate ∇y we need all the partial derivatives $\frac{\partial y}{\partial w_h}, \frac{\partial y}{\partial w_{hy}}, \frac{\partial y}{\partial w_{xy}}$. We'll start with the ones closest to the output.

$$\frac{\partial y}{\partial w_{hy}}=\frac{\partial y}{\partial s_y}\cdot\frac{\partial s_y}{\partial w_{hy}}=g'(s_y)h=\delta_y h$$
$$\frac{\partial y}{\partial w_{xy}}=\frac{\partial y}{\partial s_y}\cdot\frac{\partial s_y}{\partial w_{xy}}=g'(s_y)x=\delta_y x$$
$$\frac{\partial y}{\partial w_{xh}}=\frac{\partial y}{\partial s_y}\cdot\frac{\partial s_y}{\partial w_{xh}}=g'(s_y)\frac{\partial}{\partial w_{xh}}(w_{hy}h+w_{xy}x)$$
$$=g'(s_y)\left(\frac{\partial}{\partial s_h}(w_{hy}g(s_h))\frac{\partial s_h}{\partial w_{xh}}+\frac{\partial}{\partial w_{xh}}w_{xy}x\right)$$
$$=w_{hy}g'(s_y)g'(s_h)\frac{\partial s_h}{\partial w_{xh}}$$
$$=w_{hy}g'(s_y)g'(s_h)\frac{\partial(w_{xh}x)}{\partial w_{xh}}$$
$$=w_{hy}g'(s_y)g'(s_h)x=w_{hy}\delta_y g'(s_h)x$$

Discussion 12 – PCA

Solution:

We can massage the objective function (left's call if $f_0(u)$ in this way:

$$f_0(u)=\frac{1}{n}\sum_{i=1}^{n}\left(u^T x_i-u^T\hat{x}\right)^2$$
$$=\frac{1}{n}\sum_{i=1}^{n}\left((x_i-\hat{x})^T u\right)^2$$

Minicards

Gaussian distribution [7, 8]

1-var (normal): $p(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

Multivar: $p(x)=\frac{1}{\sqrt{|\Sigma|}\sqrt{2\pi}^d}\exp\left(-\frac{1}{2}(x-\mu)^\intercal\Sigma^{-1}(x-\mu)\right)$

The **covariance** Σ of variables X is a matrix such that each entry $\Sigma_{ij}=\mathrm{Cov}(X_i,X_j)$. This means that the diagonal entries $\Sigma_{ii}=\mathrm{Var}(X_i)$. If the matrix is diagonal, then the non-diagonal entries are zero, which means all the variables X_i are independent.

It's nice to have independent variables, so we try to diagonalize non-diagonal covariances.

Spectral Theorem [7:23]

1. Take definition of eigenvalue/vector: $Ax=\lambda x$
2. Pack multiple eigenvalues into $\Lambda=\mathrm{diag}\left(\lambda_1,\lambda_2,\ldots,\lambda_n\right)$ n eigenvalues exist iff A is symmetric.
3. Pack multiple eigenvectors into $U=\begin{bmatrix}x_1 & x_2 & \ldots & x_n\end{bmatrix}$
4. Rewrite equation using these: $\boxed{AU=U\Lambda\longrightarrow A=U\Lambda U^I}$. We can use this to diagonalize a symmetric A.

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 $\theta\cdot x=\sum_i\alpha^i y^i x^i\cdot x$:
$\alpha_i=$ # times point i 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 $\alpha^i=\mathbf{1}_i$ 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

(l) False: In boosting we start with a Gaussian weight distribution over the training samples.
(m) False: In Adaboost, the error of each hypothesis is calculated by the ratio of misclassified examples to the total number of examples.
(n) True: When $k = 1$ and $N \to \infty$, the kNN classification rate is bounded above by twice the Bayes error rate.
(o) True: A single layer neural network with a sigmoid activation for binary classification with the cross entropy loss is exactly equivalent to logistic regression.
(p) True: Convolution is a linear operation i.e. $(\alpha f_1 + \beta f_2) * g = \alpha f_1 * g + \beta f_2 * g$.
(q) True: The k-means algorithm does coordinate descent on a non-convex objective function.
(r) True: A 1-NN classifier has higher variance than a 3-NN classifier.
(s) False: The single link agglomerative clustering algorithm groups two clusters on the basis of the maximum distance between points in the two clusters.
(t) False: The largest eigenvector of the covariance matrix is the direction of minimum variance in the data.
(u) False: The eigenvectors of AA^T and A^TA are the same.
(v) True: The non-zero eigenvalues of AA^T and A^TA are the same.

(a) In linear regression, the irreducible error is σ^2 and $\boxed{E\left[(y - \mathrm{E}(y \mid x))^2\right]}$.
(b) Let S_1 and S_2 be the support vectors for w_1 (hard margin) and w_2 (soft margin). Then S_1 may not be a subset of S_2 and w_1 may not be equal to w_2.
(c) Ordinary least square regression assumes each data point is generated according to a linear function of the input plus $\mathcal{N}(0, \sigma)$ noise. In many systems, the noise variance is a positive linear function of the input. In this case, the probability model that describes this situation is $\boxed{P(y|x) = \frac{1}{\sigma\sqrt{2\pi x}} \exp(-\frac{(y-(w_0+w_1x))^2}{2x\sigma^2}}$.
(d) Averaging the outputs of multiple decision trees helps reduce variance.
(e) The following loss functions are convex: logistic, hinge, exponential. Misclassification loss is not.
(f) Bias will be smaller and variance will be larger for trees of smaller depth.
(g) If making a tree with k-ary splits, the algorithm will prefer high values of k and there will be $k-1$ thresholds for a k-ary split.

(b) True: logistic regression can be used for classification.
(c) False: Two ways to prevent beta vectors from getting too large are to use a small step size and use a small regularization value
(d) False: The L2 norm is often used because it produces sparse results, as opposed to the L1 norm which does not
(e) False: For multivariate gaussian, the eigenvalues of the covariance matrix are inversely proportional to the lengths of the ellipsoid axes that determine the isocontours of the density.
(f) True: In a generative binary classification model where we assume the class conditionals are distributed as poisson and the class priors are bernoulli, the posterior assumes a logistic form.
(g) False: MLE gives us not only a point estimate, but a distribution over the parameters we are estimating.
(h) False: Penalized MLE and bayesian estimators for parameters are better used in the setting of low-dimensional data with many training examples
(i) True: It is not good machine learning practice to use the test set to help adjust the hyperparameters
(j) False: a symmetric positive semidefinite matrix always has nonnegative elements.
(k) True: for a valid kernel function k, the corresponding feature mapping can map a finite dimensional vector to an infinite dimensional vector
(l) False: the more features we use, the better our learning algorithm will generalize to new data points.
(m) True: a discriminative classifier explicitly models $\mathrm{P}(Y \mid X)$.

(a) You can use kernels with SVM and perceptron.
(b) Cross validation is used to select hyperparameters. It prevents overfitting, but is not guaranteed to prevent it.
(c) L2 regularization is equivalent to imposing a Gaussian prior in linear regression.
(d) If we have 2 two-dimensional Gaussians, the same covariance matrix for both will result in a linear decision boundary.
(e) The normal equations can be derived from minimizing empirical risk, assuming normally distributed noise, and assuming $\mathrm{P}(Y \mid X)$ is distributed normally with mean B^Tx and variance σ^2.
(f) Logistic regression can be motivated from log odds equated to an affine function of x and generative models with gaussian class conditionals.
(g) The perceptron algorithm will converge only if the data is linearly separable.
(h) True: Newton's method is typically more expensive to calculate than gradient descent per iteration.
True: for quadratic equations, Newton's method typically requires fewer iterations than gradient descent.
False: Gradient descent can be viewed as iteratively reweighted least squares.
(i) True: Complementary slackness implies that every training point that is misclassified by a soft margin SVM is a support vector.
True: When we solve the SVM with the dual problem, we need only the dot product of x_i and x_j for all i, j.
True: we use Lagrange multipliers in an optimization problem with inequality constraints.
(j) $\|\Phi(x) - \Phi(y)\|_2^2$ can be computed exclusively with inner products.
But not $\|\Phi(x) - \Phi(y)\|_1$ norm or $\Phi(x) - \Phi(y)$.
(k) Strong duality holds for hard and soft margin SVM, but not constrained optimization problems in general.

$$= \frac{1}{n}\sum_{i=1}^{n}(u^T(x_i - \hat{x}))((x_i - \hat{x})^T u)$$
$$= u^T\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})(x_i - \hat{x})^T\right)u$$
$$= u^T\Sigma u$$

Show that the maximizer for this problem is equal to v_1, where v_1 is the eigenvector corresponding to the largest eigenvalue λ_1. Also show that optimal value of this problem is equal to λ_1.

Solution:
We start by invoking the spectral decomposition of $\Sigma = V\Lambda V^T$, which is a symmetric positive semi-definite matrix.

$$\max_{u:\|u\|_2=1} u^T\Sigma u = \max_{u:\|u\|_2=1} u^T V\Lambda V^T u$$
$$= \max_{u:\|u\|_2=1} (V^T u)^T \Lambda V^T u$$

Here is an aside: note through this one line proof that left-multiplying a vector by an orthogonal (or rotation) matrix preserves the length of the vector:

$$\|V^T u\|_2 = \sqrt{(V^T u)^T (V^T u)} = \sqrt{u^T V V^T u} = \sqrt{u^T u} = \|u\|_2$$

I define a new variable $z = V^T u$, and maximize over this variable. Note that because V is invertible, there is a one to one mapping between u and z. Also note that the constraint is the same because the length of the vector u does not change when multiplied by an orthogonal matrix.

$$\max_{z:\|z\|_2=1} z^T\Lambda z = \max_z \sum_{i=1}^{d}\lambda_i z_i^2 \ : \ \sum_{i=1}^{d} z_i^2 = 1$$

From this new formulation, it is obvious to see that we can maximize this by throwing all of our eggs into one basket and setting $z_i^* = 1$ if i is the index of the largest eigenvalue, and $z_i^* = 0$ otherwise. Thus,

$$z^* = V^T u^* \implies u^* = V z^* = v_1$$

where v_1 is the "principle" eigenvector, and corresponds to λ_1. Plugging this into the objective function, we see that the optimal value is λ_1.

Solution: (a) We have

$$J(\mathbf{v_2}, \mathbf{z_2}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x_i^T x_i} - z_{i1}\mathbf{x_i^T v_1} - z_{i2}\mathbf{x_i^T v_2} - z_{i1}\mathbf{v_1^T x_i} + z_{i1}^2\mathbf{v_1^T v_1} + \quad (1)$$
$$z_{i1}z_{i2}\mathbf{v_1^T v_2} - z_{i2}\mathbf{v_2^T x_i} + z_{i1}z_{i2}\mathbf{v_2^T v_1} + z_{i2}^2\mathbf{v_2^T v_2}) \quad (2)$$

Take derivative respect to $\mathbf{z_2}$, we have

$$\frac{\partial J}{\partial z_{i2}} = \frac{1}{n}(-\mathbf{x_i^T v_2} + z_{i1}\mathbf{v_1^T v_2} - \mathbf{v_2^T x_i} + z_{i1}\mathbf{v_2^T v_1} + 2z_{i2}\mathbf{v_2^T v_2}) = \frac{1}{n}(-2\mathbf{x_i^T v_2} + 2z_{i2}\mathbf{v_2^T v_2})$$

Set the derivative to 0 and we have

$$z_{i2}\mathbf{v_2^T v_2} = \mathbf{x_i^T v_2}$$

Since $\mathbf{v_2^T v_2} = 1$, we have $z_{i2} = \mathbf{x_i^T v_2}$
(b) Plug in z_{i2} into $J(\mathbf{v_2}, \mathbf{z_2})$, we have

$$J(\mathbf{v_2}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x_i^T x_i} - z_{i1}\mathbf{x_i^T v_1} - z_{i2}\mathbf{x_i^T v_2} - z_{i1}\mathbf{v_1^T x_i} + z_{i1}^2\mathbf{v_1^T v_1} - z_{i2}\mathbf{v_2^T x_i} + z_{i2}^2\mathbf{v_2^T v_2})$$
$$= \frac{1}{n}\sum_{i=1}^{n}(const - 2z_{i2}\mathbf{x_i^T v_2} + z_{i2}^2) = \frac{1}{n}\sum_{i=1}^{n}(-2\mathbf{v_2^T x_i x_i^T v_2} + \mathbf{v_2^T x_i x_i^T v_2} + const)$$
$$= -\mathbf{v_2^T C v_2} + const$$

In order to minimize J with constraints $\mathbf{v_2^T v_2} = 1$, we have Langrage $L = -\mathbf{v_2^T C v_2} + \lambda(\mathbf{v_2^T v_2} - 1)$ and take derivative of $\mathbf{v_2}$, we have

$$\frac{\partial L}{\partial \mathbf{v_2}} = -2\mathbf{C v_2} + 2\lambda v_2 = 0$$

Then, we have

$$\mathbf{C v_2} = \lambda \mathbf{v_2}$$

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.