CPSC 340 Formula Sheet

Basics

Entropy: $-\sum_{c=1}^{k} p_c \log p_c$
L0 norm: $|r|_0$ counts number of non-zero elements in r
L1 norm: $|r|_1 = \sum|r_i|$
L2 norm: $|r|_2 = \sqrt{\sum r_i^2}$
L∞ norm: $|r|_\infty = \max|r_i|$
Dot product: $a^T b = b^T a$
Norm: $\|a\|^2 = a^T a$
Transpose: $(A+B)^T = A^T + B^T \qquad (AB)^T = B^T A^T$

Supervised Learning

Notation
Feature matrix $X = [n \times d]$
Example i: x_i. Column j: x^j
Label vector $y = [n \times 1]$
Prediction vector $\hat{y} = [n \times 1]$
Test data/predictions: $\tilde{X}$ and $\tilde{y}$
Concepts
Training accuracy: accuracy on training data
Test accuracy: accuracy on new (test) data
Golden rule: never train on test data
Overfitting: low accuracy on test data since model is too specific to training data
IID assumption: assume training data reflects test data
Training and test error: E_{train} and E_{test}.
Generalization gap: $E_{\text{gap}} = E_{\text{test}} - E_{\text{train}}$
Fundamental trade-off: for complex model, E_{train} decreases but E_{gap} increases
Validation: use part of training data to approximate test data
Optimization bias: find good model by chance, but have overfit to validation set
Cross validation: partitition section of data for validation, then average all of the validation errors to get a more accurate estimate of the test error

Decision Trees

Decision stump: split data on one feature, then pick most common labels for predictions. $O(nkd)$, k is number of thresholds.
Decision tree: greedy splitting or info gain (better). Aim to decrease entropy.
info gain = entropy$(y) - \frac{n_1}{n}$entropy$(y_1) - \frac{n_2}{n}$entropy(y_2)

Naive Bayes

$P(A \mid B) = \frac{P(B|A)P(A)}{P(B)}$
$P(\text{spam} \mid \text{words}) = \frac{P(\text{words}|\text{spam})P(\text{spam})}{P(\text{words})}$
$P(\text{words} \mid \text{spam}) = \prod_{j=1}^{d} P(\text{word}_j \mid \text{spam})$
Estimate $P(\text{word}_j \mid \text{spam})$ and $P(\text{spam})$ based on proportion in training data
Laplace smoothing: avoid no occurences for a word in a class. Add β to numerator and βk to denominator of $P(\text{word}_j \mid \text{spam})$

KNN

Finds k nearest training examples, then takes most common label.
Lazy learning: does not train, just memorizes data. $O(nd)$ to classify one example.
Nonparametric: model size depends on number of training examples
Problematic for high-dimensional data or features with different scales

Ensemble Methods

Voting: take majority vote of multiple (possibly different structure) models
Stacking: fit a classifier on the predictions of other classifiers
Bootstrapping: sample with replace to create new training sets
Random trees: only consider subset of features at each split (typically $\sqrt{d}$)
Random forests: bootstrap data to train multiple random trees, then vote

Clustering

K-Means: assigns each point to closest mean, then update means. $O(ndk)$ to assign examples, $O(nd)$ to update means. Does not work well with non-convex clusters.
Label switching: cluster label $\hat{y}_i$ is meaningless
Vector quantization: replace examples with mean of their cluster
Density-based clustering: ε threshold for neighbour. MinNeighbours: number of neighbours needed for dense region.
1. Find core points that have MinNeighbours points nearby
2. Merge core points into clusters (can be reached by traversing through other core points)
3. Expand clusters based on existing and newfound core points
Naive case $O(dn^2)$, but can be sped up to $O(dn \log n)$.
Hierarchical clustering: tree of clustering which splits data into small clusters.
Bottom up clustering: each point starts as own cluster, then merge closest clusters repeatedly.
Biclustering: cluster training examples and features. Heatmap visualizes clusters.

Outliers

1. Probabilistic: assume data is generated by a distribution, then find points with low probability
2. Graphical: visualization of data. Limited by dimensionality
3. Cluster based: find points far from clusters, or small clusters
4. Distance based: find points far from other points, KNN
Local distance: outlierness $= \frac{\text{average distance of } i \text{ to KNN}}{\text{average distance of neighbours to KNN}}$
5. Supervised learning

Linear Regression

1D objective: minimize $f(w) = \frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2$
1D solution: $w = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}$
Weights $w = [d \times 1]$
Data matrix $X = [n \times d]$
Example $x_i = [d \times 1]$
Outputs $y = [n \times 1]$
Objective: $f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{2}\|Xw - y\|^2$
Matrix gradients:
$\nabla[c] = 0 \qquad \nabla\left[w^T b\right] = b$
$\nabla\left[\frac{1}{2} w^T A w\right] = Aw$ for symmetric A.
2D solution: $X^T X w = X^T y$ (normal equations)
Normal equations time: $O(nd^2 + d^3)$
Bias: create matrix Z with extra feature of 1s
Polynomial: $Z = \begin{bmatrix} 1 & x_1 & \cdots & (x_1)^p \\ \vdots & \vdots & \ddots & \\ 1 & x_n & \cdots & (x_n)^p \end{bmatrix}$
Transformed data matrix: $Z = [n \times k]$
Transformed normal equations: $Z^T Z v = Z^T y$

Gradient Descent

Iteration equation: $w^{t+1} = w^t - \alpha^t \nabla f(w^t)$
Gradient descent time: $O(ntd)$
Convex functions: lines between points are above function
Convexity criterion:
1. 1-variable, twice-differentiable, $f''(w) \geq 0$
2. convex function * non-negative constant
3. linear functions, norms, squared norms
4. sum/max of convex functions
5. composition of convex function with linear function $f(w) = g(Xw - y)$, g is convex

Modified Linear Regression

Robust regression: L1 loss $f(w) = \sum_{i=1}^{n}\left|w^T x_i - y_i\right|$
Huber loss: smooth approximation to L1.

$$h(r_i) = \begin{cases} \frac{1}{2} r_i^2 & \text{if } |r_i| \leq \varepsilon \\ \varepsilon(|r_i| - \frac{1}{2}\varepsilon) & \text{otherwise} \end{cases}$$

Brittle regression: L∞ norm minimizes highest error
Log-sum-exp: smooth approximation to L∞
$\max_i\{z_i\} \approx \log(\sum_i \exp(z_i))$
Information criteria: score$(p) = \frac{1}{2}\|Z_p v - y\|^2 + \lambda k$
Degrees of freedom: $k = (p+1)$ for polynomial
Akaike information criterion (AIC): $\lambda = 1$
Bayesian information criterion (BIC): $\lambda = \frac{1}{2}\log(n)$

Feature Selection

Association: compute correlation between feature and output. Performs poorly because it ignores variable interactions
Regression weight: keep features with large weights. Has major problems with collinearity
Search and score: pick best subset of features based on score (often validation error). Struggles with large number of features
Forward selection: start with no features, greedily add features that improve score
L0-penalty: $f(w) = \frac{1}{2}\|Xw - y\|^2 + \lambda\|w\|_0$

Regularization

L2-Regularization (Ridge): $f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$
Normal equations: $(X^TX + \lambda I)w = X^Ty$
Regularization path: plot weights against λ
Standardize features: replace y_i with $\frac{y_i - \mu_y}{\sigma_y}$
Radial basis functions (RBF): replace x_i with
$z_i = (g(\|x_i - x_1\|), \dots, g(\|x_i - x_n\|))$
Gaussian RBF: $g(\varepsilon) = \exp(-\frac{\varepsilon^2}{2\sigma^2})$
Hyperparameter optimization: exhaustive search σ and λ, or random search
L1-Regularization (LASSO): $f(w) = \frac{1}{2}\|Xw - y\|^2 + \lambda\|w\|_1$
- Lower test error, requires gradient methods
- Non-unique and sparse solutions
- Can learn with exponential irrelevant features
Ensemble: designed to reduce false positives/false negatives
Bootstrap: only take features selected in all bootstraps (reduces false positives)

Linear Classifiers

Prediction: $o_i = w^Tx_i$
Least squares: not good because "too right" predictions are penalized
0-1 loss: number of classification errors
0-1 loss = $\|\text{sign}(o_i) - y\|_0$
Non-convex: difficult to minimize
Perceptron: update weights when prediction is wrong
$w^0 = 0, \quad w^{t+1} = w^t + y_ix_i$
Minimizes 0-1 loss if data is linearly separable
Degenerate convex: $f(w) = \sum_{i=1}^n \max\{0, -y_io_i\}$
Hinge loss: $\max\{0, 1 - y_io_i\}$
Support vector machine (SVM): hinge loss with L2 regularization
- $f(w) = \sum_{i=1}^n \max\{0, 1 - y_iw^Tx_i\} + \frac{\lambda}{2}\|w\|^2$
- $f(w) = C\sum_{i=1}^n \max\{0, 1 - y_iw^Tx_i\} + \frac{1}{2}\|w\|^2$
- $C = \frac{1}{\lambda}$, low C means high regularization
Logistic loss: $\log(1 + \exp(-y_iw^Tx_i))$
Probability: $p(y_i = 1 \mid w, x_i) = \frac{1}{1+\exp(-w^Tx_i)}$
One vs All: one classifier for each class, then take most confident (highest value of o_{ic})
$W = [k \times d]$, each row is a classifier
Multi-class SVM: $w_{y_i}^Tx_i > w_c^Tx_i$ for all $c \neq y_i$
- Sum: $\sum_{c \neq y_i} \max\{0, 1 - w_{y_i}^Tx_i + w_c^Tx_i\}$
- Max: $\max_{c \neq y_i}\{\max\{0, 1 - w_{y_i}^Tx_i + w_c^Tx_i\}\}$
- Softmax: $w_{y_i}^Tx_i > \max_c\{w_c^Tx_i\}$
- $\implies$ minimize $-w_{y_i}^Tx_i + \log(\sum_{c=1}^k \exp(w_c^Tx_i))$
Multi-class probability: $p(y = c \mid z_1, z_2, \dots, z_k) = \frac{\exp(z_c)}{\sum_{c'=1}^k \exp z_{c'}}$
Matrix form: predict $\text{argmax}(XW^T)$

Feature Engineering

Discretization: convert continuous to categorical, good for counting-based methods
Standardize: convert to same units/normal distribution, good for distance-based methods
Non-linear transforms: polynomial, exponential/logarithm, sinusoidal, RBFs, good for regression-based methods
Bag of words: represent sentences/documents by word counts
N-gram: ordered sets of n words. Captures local context
Kernel trick:
- L2-regularized least squares: $f(v) = \frac{1}{2}\|Zv - y\|^2 + \frac{\lambda}{2}\|v\|^2$
- Solution: $v = \underbrace{(Z^TZ + \lambda I)}_{k \times k}^{-1}Z^Ty$
- Equivalent solution: $v = Z^T\underbrace{(ZZ^T + \lambda I)}_{n \times n}^{-1}y$.
- $\hat{y} = \tilde{Z}v = \tilde{Z}Z^T(ZZ^T + \lambda I)^{-1}y = \tilde{K}(K + \lambda I)^{-1}y = \tilde{K}u$
- Gram matrix $K = ZZ^T = [n \times n]$
- $\tilde{K} = \tilde{Z}Z^T = [t \times n]$
- Kernel function $k(x_i, x_j) = z_i^Tz_j$
- Degree-p polynomial: $k(x_i, x_j) = (1 + x_i^Tx_j)^p$
- RBF: $k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$
- $K_{ij} = (1 + x_i^Tx_j)^p, \quad \tilde{K}_{ij} = (1 + \tilde{x}_i^Tx_j)^p$
- Training cost $O(n^2d)$, prediction cost $O(ndt)$

Stochastic Gradient Descent

Sample gradient: $w^{t+1} = w^t - \alpha^t\nabla f_i(w^t)$
Use cases: minimize average, can't do brittle regression
Decreasing step sizes: needed for convergence
- Use $\alpha^t = O(1/\sqrt{t})$
Mini-batch: sample B examples for calculating gradient
Termination: stop when error after k iterations is less than ε

MLE and MAP

Regression tree: predict mean of training examples in leaf
XGBoost: each tree corrects previous trees
Likelihood: for dataset D and parameters w, pmf is $p(D \mid w)$
Maximum likelihood estimation (MLE): choose
$\hat{w} \in \text{argmax}_w\{p(D \mid w)\}$
Negative log-likelihood (NLL): minimize
$\text{argmin}_w\{-\log(p(D \mid w))\}$
IID: $p(D \mid w) = \prod_{i=1}^n p(D_i \mid w) = \prod_{i=1}^n p(y_i \mid w, x_i)$
Least squares is MLE under Gaussian likelihood:
$p(y_i \mid x_i, w) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{(w^Tx_i - y_i)^2}{2})$
MLE of w is minimum of $f(w) = \frac{1}{2}\|Xw - y\|^2$
Absolute error is MLE under Laplace likelihood:
$p(y_i \mid x_i, w) = \frac{1}{2}\exp(-|w^Tx_i - y_i|)$
MLE of w is minimum of $f(w) = \|Xw - y\|_1$
Logistic loss is MLE under sigmoid likelihood
Maximum a posteriori (MAP): choose
$\hat{w} \in \text{argmax}_w\{p(w \mid D)\}$
$p(w \mid D) = \frac{p(D|w)p(w)}{p(D)} \propto p(D \mid w)p(w)$
Prior $p(w)$: belief that w is correct before seeing data
Gaussian likelihood + prior gives L2-regularized least squares
Laplace likelihood + Gaussian prior gives L2-regularized robust regression

PCA

Data matrix: $X[n \times d]$, k components
Outputs $Z = [n \times k], W = [k \times d]$
Approximation: $\hat{x}_{ij} = \langle w^j, z_i\rangle, \hat{x}_i = W^Tz_i, X \approx ZW$
Applications: dimensionality reduction, visualization, outlier detection
Objective: $f(W, Z) = \sum_{i=1}^n \|W^Tz_i - x_i\|^2$
Centering: each column of X has mean zero
Choosing k: based on explained variance in data
Prediction: replace $\tilde{x}_{ij}$ with $\tilde{x}_{ij} - \mu_j$, then
$\tilde{Z} = \tilde{X}W^T(WW^T)^{-1}$
Sequential fitting: find orthonormal PCs one at a time
Alternating minimization: fix Z and find optimal W, then fix W and find optimal Z
SGD: works well for large dataset X
Outliers in X: can use L1 loss
$f(W, Z) = \sum_{i=1}^n \sum_{j=1}^d |\langle w^j, z_i\rangle - x_{ij}|$
L2-regularized PCA:
$f(W, Z) = \frac{1}{2}\|ZW - X\|^2 + \frac{\lambda_1}{2}\|W\|^2 + \frac{\lambda_2}{2}\|Z\|^2$
Recommender Systems:
- Content-based filtering (supervised): extract features x_i of users and items, builds model to predict rating y_i given x_i
- Collaborative filtering (unsupervised): only have labels y_{ij}
Matrix factorization: PCA over available ratings
Multi-dimensional scaling (MDS): preserve distances
$f(Z) = \sum_{i=1}^n \sum_{j=i+1}^n (\|z_i - z_j\| - \|x_i - x_j\|)^2$
t-distributed stochastic neighbour embedding (t-SNE): preserve neighbour distances

Neural Networks

Neuron: $\hat{y}_i = v^Th(Wx_i)$
Add bias: $\hat{y}_i = v^Th(Wx_i) + b$
Sigmoid: $\frac{1}{1+\exp(-w_c^Tx_i)}$, smooth approximation to 0-1
Regression: minimize squared residual
Binary classification: minimize logistic loss
Multi-class classification: minimize log softmax
Deep learning: many hidden layers
Backpropagation: compute gradients of loss wrt weights
- m layers, z_i have k elements $\implies O(dk + mk^2)$
Vanishing gradients: gradients become very small
Relu: $\max\{0, z_{ic}\}$, avoids vanishing gradients
Skip connections: add shortcuts between layers
Dropout: randomly set some activations to zero
ResNet: $a^{l+2} = h(a^l + W^{l+1}h(W^la^l))$
Parameter initialization: weights cannot be the same initially
Learning rate decay: decrease learning rate over time
Momentum:
$w^{t+1} = w^t - \alpha^t\nabla f_i(w^t) + \beta^t(w^t - w^{t-1})$, usually $\beta^t = 0.9$
Weight decay: L2-regularize v and W

Convolution Neural Network

Convolution: apply sliding filter
Max pooling: take max of each region