**Types** Supervised Learning: Learn from labeled training data, make predictions about unseen/future data; classification (discrete class labels, binary/multiclass), regression (continuous output value) Reinforcement Learning: The system (aka agent) improves its performance based on interactions

with an environment; trial and error; agent receives feedback (reward) from the environment. Unsupervised Learning: Unlabeled data/data of unknown structure, explores the structure of data. Preprocessing: Feature selection, extraction & scaling, Dimensionality reduction, Sampling. Learning Process: Model selection, Cross-validation, Performance metric, Hyperparameter optimization. Evaluation & Prediction: use the test dataset to estimate how well it performs on unseen data.

**Artificial Neuron** z = dot(w, x); (x0 = 1, w0 = -theta, called the bias unit) phi(z) = 1 ? z >= 0 : -1; **Perceptron Learning Rule** for each training sample x[i], compute output y\_hat[i], update weights w[j] += eta \* (y[i] – y\_hat[i]) \* x[i][j]; the convergence of the perceptron is only guaranteed if the two classes are linearly separable & the learning rate is sufficiently small; for not linearly separable samples, set a max epoch count. **Adaptive Linear Neuron** Improvement on Perceptron algorithm, weights are updated based on a linear activation function: phi(z) = z; use threshold function to make the final prediction; cost function J(w) = (1/2) \* sum((y[i] – phi(z[i]))^2); differentiable & convex, can be minimized with gradient descent. Weight update: w[j] += eta \* sum((y[i] – phi(z[i])) \* x[i][j]); Batch gradient descent: weight update is calculated based on all samples in the training set. Feature Scaling: Standardization: for feature j, x[j] = (x[j] – average(x[,j])) / standard\_deviation(x[,j]); Stochastic Gradient Descent: Instead of updating the weights based on the sum of the accumulated errors over all samples, update all the weights incrementally for each training sample; Typically reaches convergences faster because of the more frequent weight updates, Can escape shallow local minima more readily for nonlinear cost functions, Can be used for online learning (model is trained on the fly as new training data arrives); shuffle the training set for every epoch. Mini-Batch Learning: A compromise between batch gradient descent and stochastic gradient descent, apply batch gradient descent to smaller subsets of the training data; converge faster than batch gradient descent.

**Multiclass Classification** One-versus-Rest: Train one classifier per class where the class is treated as positive & other classes are considered negative to classify a new data sample, use all classifiers, assign the class label with the highest confidence to the particular sample. **Logistic Regression**: Logit function: defined as the logarithm of odds, domain = (0, 1), range = real numbers, logit(p) = ln(p) – ln(1 - p); logit(P(y = 1|x)) = z = dot(w, x); Activation function phi(z) = logit\_inverse(z). Threshold function y\_hat = 1 ? phi(z) >= 0.5 : 0; Loss function J(phi(z), y) = -y \* ln(phi(z)) – (1 - y) \* ln(1 – phi(z)); **Overfitting & Underfitting** model may perform well on training data but does not generalize well to test data; overfitting (high variance), underfitting (high bias); Regularization: introduce additional information (bias) to penalize extreme parameter (weight) values; useful method to handle collinearity (high correlation among features), filter out noise from data & eventually prevent overfitting; requires feature scaling e.g. standardization; L2 Regularization: add (lambda / 2) \* sum(w[j] ^ 2) to the cost function; lambda is the regularization parameter, larger means more regularization strength. **Support Vector Machine** An extension of the perceptron (Perceptron minimizes misclassification errors, SVM maximizes margin). Large Margin Classification: Fitting the widest possible street; fully determined by samples located on the edge of the street (the support vectors), adding more training instances off the street will not affect the decision boundary. Decision Rules: y = 1 ? dot(w, x) + w0 >= 0 : -1; then constrain is: for all i, y[i] \* (dot(w, x[i]) + w0) >= 0; maximize 2 / |w|, minimize |w|; SVMs are sensitive to the feature scales. Hard margin classification: all instances must be off the street & on the right side; only works if the data is linearly separable, sensitive to outliers. Soft margin classification: find a good balance between keeping the street as large as possible and limiting the margin violations. Slack variable: (QP form) minimize (|w|^2) / 2 + C \* sum(slack[i]) under constraint y[i] \* (dot(w, x) + w0) >= 1 – slack[i]; smaller C value increases bias, lowers variance. Hinge Loss Function: equivalent to QP form, hinge(i) = max(0, 1 – y[i] \* (dot(w, x) + w0)); minimize (|w|^2) / 2 + C \* sum(hinge(i)); Logistic regression tries to maximize the conditional likelihoods of the training data & is more prone to outliers than SVMs, logistic regression is a simpler model, can be easily updated. Kernel SVM: project not linearly separatable data into a higher dimension space to make them linearly separatable. Kernel Trick: replace the dot product with a more efficient operation. gaussian\_kernel(x[i], x[j]) = exp(-gamma \* |x[i] – x[j]|^2); Larger gamma value increases influence of training samples & results in tighter decision boundaries. **Decision Tree** Model learns to ask a series of questions & bases the prediction on answer; split data on feature that results in largest Information Gain, iterative repeat splitting until leaves are pure/max depth. Gini Impurity (minimize the probability of misclassification) Ig(t: Node) = 1 – sum(P(i|t)^2); Information Gain IG(d: Node) = I(d) – (n\_left / n) \* I(d\_left) - (n\_right / n) \* I(d\_right); Entropy (maximize the mutual information in the tree) Ih(t: Node) = -sum(P(i|t) \* log(P(i|t))); Classification Error (useful for pruning, not recommended for growing decision trees) Ie(t: Node) = 1 – max(P(i|t)); Random Forests: An ensemble of decision trees. Draw a random bootstrap sample of size n, grow a decision tree from the bootstrap sample, (at each node randomly select d features without replacement, split the node using the feature that provides the best split), Repeat k times, aggregate the prediction by each tree to assign the class label by majority vote. Decreasing increases diversity among trees, lowers overall performance; Increasing n increases overfitting. **K-Nearest Neighbor** classifier immediately adapts to new training data, bad computational complexity, cannot discard training samples, susceptible to overfitting due to the curse of dimensionality.

**Data Preprocessing** Missing Data: discard entire column/row, interpolation. Encoding Class Labels: nominal (cannot be sorted)/ordinal (can be sorted) features; use one-hot coding for nominal features. Scaling: Normalization: for feature j, x[j] = (x[j] – min(x[,j])) / (max(x[,j]) - min(x[,j])); Feature Selection: can simplify model for easier interpretation, shorten training time, avoid curse of dimensionality, reduce overfitting. L1/L2 regularization; Define a criterion function J, for each iteration remove one feature that causes least performance loss. Data Extraction: dimensionality reduction

**PCA** (unsupervised linear transformation)identifies a hyperplane and then projects data onto it. The 1st principal component (PC) is the axis that accounts for the largest amount of variance, the 2nd PC is orthogonal to the 1st and accounts for the largest amount of remaining variance, all PC must be orthogonal. PCs can be found with singular value decomposition on the training set, features should be standardized first. Explained Variance Ratio: the proportion of the dataset’s variance that lies along each principal component **Locally Linear Embedding** (nonlinear, manifold learning, does not rely on projections) measures how each training instance linearly relates to its closest neighbors, then looks for a low-dimensional representation of the training set where these local relationships are best preserved; good at unrolling twisted manifolds, especially when there is not too much noise. Algorithm: For each training sample x[i], select k closest neighbors and try to find a lerp of these neighbors that has the closest distance to x[i], subject to sum(weight[i][j]) = 1; then for each point in the dimension-reduced space z[i], find the z[i] so that the distance between z[i] and sum(weight[i][j] \* z[j]) is minimized.

**Validation** to estimate its performance on data that the model hasn't seen before & find an acceptable bias-variance trade-off. Holdout Method: classic approach, split the initial dataset into a separate training & test dataset; performance estimate may be very sensitive to the partitioning. K-Fold Cross-Validation: Randomly split the training dataset into k folds without repetition, use k-1 folds for training & one for evaluation, repeat k times, calculate the average performance. A good value for k is 10, larger datasets use smaller k values. Leave-One-Out Cross-Validation: Let k = number of training samples, only one sample used for testing, for very small datasets. Stratified Cross-Validation: class proportions are preserved in each fold to ensure that each fold is representative of the class proportions in the training dataset. Nested Cross-Validation: an outer k-fold cross-validation loop to split the data into training and test folds, an inner loop is used to select the model using k-fold cross-validation on the training fold. **Evaluation Metrics** confusion\_matrix = {true\_positive, true\_negative, false\_positive, false\_negative}; precision = true\_positive / (true\_positive + false\_positive); recall = true\_positive / (true\_positive + false\_negative); F1 = 2 \* precision \* recall / (precision + recall); Receiver Operating Characteristic (ROC): a graph with true positive rate (TPR) & false positive rate (FPR) as axes, useful tool to select models for classification based on their performance, computed by shifting the decision threshold of the classifier; the diagonal of an ROC graph can be interpreted as random guessing, perfect classifier would fall into the top left corner of the graph with a TPR of 1 and an FPR of 0; ROC Area Under the Curve (ROC AUC) to characterize the performance of a classification model. FPR = false\_positive / (false\_positive + true\_negative); TPR = true\_positive / (true\_positive + false\_negative); Multiclass Score: extended from binary classification with one-vs-all classification; micro average is calculated using individual points, weights each instance/prediction equally; macro average is the average of the score of each class, weights all classes equally, evaluates the overall performance of a classifier with regard to the most frequent class labels. **Class Imbalance** Samples from some classes are over-represented in a dataset. Focus on metrics other than accuracy, e.g. precision, recall, ROC curve; class imbalance also affects the model fitting process, causing bias towards majority classes, can be solved by applying larger penalty to minority classes, or upsampling minorities + downsampling majorities.

**Ensemble Learning** Majority/Plurality Voting: select the class label that has been predicted by the majority of classifiers; majority for binary, plurality for multiclass. Bagging: draw bootstrap samples (random samples with repetition) from the initial training set, train each model on each sample, then majority voting; Effective in reducing model variance but not the bias. Adaptive Boosting: consists of weak learners, focus on training samples that are hard to classify, learners subsequently learn from misclassified training samples to improve the ensemble’s performance. Algorithm: initialize sample weight vector to uniform weights; for j out of m boosting rounds: train a weak learner C[j] = train(x, y, w), predict class labels y\_hat = predict(C[j], x), compute weighted error rate error = dot(w, [1 if y\_hat[i] != y[i] else 0]), compute trainer coefficient alpha[j] = ln((1 - error) / error) / 2, update weights weight \*= exp(-alpha[j] \* y \* y\_hat), normalize weights; final prediction y\_hat = sum(alpha[j] \* predict[C[j], x]) > 0. Ensemble learning increases the computational complexity compared to individual classifiers; computational cost must be considered in practice.

**Linear Regression** model the relationship between one or multiple features and a continuous target variable. Correlation Matrix: quantify linear relationships between variables, two features have perfect positive correlation if r = 1, or no correlation if r = 0, or perfect negative correlation if r = -1. Can be calculated as r = covariance(x, y) / (variance(x) \* variance(y)); Correlation of Determination R^2 = 1 – sum((y[i] – y\_hat[i])^2) / sum((y[i] – average(y))^2) = 1 - sum((y[i] – y\_hat[i])^2) / (n \* variance(y)); Random Sample Consensus: Select a random number of samples and perform regression; test all data against this model, add samples that fall within a user-defined tolerance to the inliers; recalculate the model with all the inliers; repeat until the performance meets a user-defined threshold/max iteration reached. Ridge Regression: Add L2 regularization to cost function. LASSO: Add L1 regularization to cost function. Elastic Net: A lerp of Ridge Regression & LASSO. Random Forest & Decision Tree Regression: can be understood as the sum of piecewise linear functions, subdivide the input space into smaller regions that become more manageable; does not require transformation for non-linear data, or normalization & standardization; impurity metric = mean squared error at each node; predicted target variable is calculated as the average prediction over all decision trees.

**Clustering** can be partitional (each data object is in exactly one cluster)/hierarchical (nested clusters organized as a hierarchical tree). Types: Well-Separated (any point in a cluster is closer to every other point in the cluster than to any point not in the cluster), Center-Based (any point in a cluster is closer to the centroid of the cluster than to any other centroids), Contiguous (nearest neighbor/transitive, a point in a cluster is closer to some other points in the cluster than to any point not in the cluster), Density-Based (a dense region of points separated by low-density regions), Defined by Objective Function (Finds clusters that minimize or maximize an objective function, enumerate all possible ways of dividing the points into clusters and evaluate them according to the objective function (NP Hard), objective can be global (typically for partitional) or local (typically for hierarchical)). **K-means** partitional; select k centroids, form k clusters around the centroids and update each centroid according to each cluster, until centroids do not change/relatively few centroids change; most converging happens during the first iterations. Sum of Squared Errors: sum of each point’s Euclid distance to its centroid. Handling the Initial Centroids Problem: multiple runs, use hierarchical clustering to determine initial centroids, select more than k centroids at first and then pick from them (choose the sparsest), post-processing, bisecting K-means. Handling empty clusters: take a point that adds the most to SSE/from a cluster with the highest SSE. Incremental Approach: update the centroids after each assignment, each assignment updates zero/two centroids, more expensive, order dependency, avoid empty clusters. ­Pre-processing: normalization, remove the outliers. Post-processing: remove clusters that might represent outliers, split loose clusters/merge tight clusters. Bisecting K-means: initialize a list of one large cluster, for each iteration, take one cluster from the list and bisect it several times with the basic K-means algorithm, and select two of them with the lowest SSE and add them back to the list, repeat until the list has k clusters. Can be partitional or hierarchical. K-Means Limitation: have problems when clusters have different sizes, non-globular shapes, or when data contain outliers. (solution: find more than k clusters then group some clusters together) **Hierarchical Clustering** no assumptions for number of clusters, may correspond to meaningful taxonomies. Agglomerative (start with individual points, merge until one cluster left)/Divisive (start with one cluster, split until each is one point). Agglomerative Algorithm: Compute the proximity matrix, make one cluster for each point; merge the two closest clusters & update the proximity matrix, repeat until only one cluster remain. Similarity Metrics: Min/Single Link: based on two closest points, determined by one pair of points, sensitive to noise; Max/Complete Linkage: based on two most distant points, determined by all pairs of points, less sensitive to noise, tends to break large clusters & biased towards globular shapes; Group Average: Proximity of two clusters is the average of pairwise proximity between points in the two clusters, compromise between single & complete link, less sensitive to noise & outliers but biased towards globular clusters; Ward’s Method: Similarity of two clusters is based on the increase in squared error when merged, less sensitive to noise & outliers but biased towards globular clusters, hierarchical analogue of K-means (can be used to initialize K-means). Problems: cannot undo a merge, no objective function is directly minimized, different schemes have problems with one or more of the following: sensitivity to noise and outliers/difficulty handling different sized clusters and convex shapes/breaking large clusters. Minimum Spanning Tree Divisive Clustering: Compute a minimum spanning tree for the proximity graph; repeatedly remove one edge of the largest distance, until only singleton clusters remain.

**Multi-layer Neural Networks** has one input layer, one output layer & any amount of hidden layer. Error gradients calculated via backpropagation become smaller as more layers are added to network, therefore special algorithms have been developed to help train such deep neural network structures (Deep Learning). Learning Procedure: start at the input layer, forward propagate through the network to generate output; calculate the error according to a cost function; backpropagate the error, find its derivative with respect to each weight in the network & update the model. Multi-layer Perceptron (MLP): Feedforward (no loops, contrast to Recursive Neural Networks), each neuron is typically a sigmoid unit, return values in the continuous range between 0 and 1. Back Propagation: a special case of reverse automatic differentiation; forward propagation involves expensive matrix-matrix multiplication, backward uses matrix-vector multiplication and is more efficient. Calculation: error[current] = (transpose(weights) \* error[next]) \* derivative(activation\_function(neuron\_output)).

**Convolution Neural Networks** Convolution Layer: Apply a filter over an image; move the filter over the image by several pixels at a time, each time generates one pixel of the output image. Convolution Network is a sequence of Convolution Layers & Activation Functions (RELU). Filters learn one template per class for linear classifier/bank of whole image templates for MLP. First layer filters learn local image templates (often oriented edges, opposing colors). Layer Hyperparameters: Number of Filters; Filter Size; Stride = number of pixel offsets after each time; Padding = added layer of 0s to the border of the image. Pooling: image downsampling, makes the representations smaller and more manageable. No learnable parameters. Hyperparameters: spatial extent (size), stride. Fully Connected Layer: Contains neurons that connect to the entire input volume.