The final will have 4 parts:

1. Compare and contrast (20%) - given two concepts, identify similarities and differences.
2. Advantages and disadvantages (20%) - given a concept, discuss the advantages and disadvantages.
3. Discussion (30%) - answer a specific question about some concept, and discuss the implications.
4. Algorithms (30%) - How does a specific algorithm work? You can draw a picture. Mathematical descriptions are not necessary, as long as you can show the intuition behind the algorithm.

**Classification**

* Bayes Classifier:
  + Training: For each class c, estimate class conditional densities: p(x|y=c) and p(y) using MLE (Maximum Likelihood Method)
  + Classification: For given sample x\*, pick the class with the largest p(y=c|x). This means this highest chance to be class c, given x
* Naive Bayes Classifier:
  + Same as Bayes, but for multiple features
  + Generative classification model:
    - Estimate probability distributions of features generated from each class
    - Given an observation predict class with largest posterior probability
  + PROS:
    - Works on **small amount of data**
    - Works with **multiple classes**
  + CONS:
    - Accuracy depends on selecting **appropriate distribution**

**Linear classifiers - separate data by using a linear surface (hyperplane) y = (wTx + b)**

* Logistic regression
  + Using probabilistic approach
  + For multiclass: divide into multiple binary classifier problems
  + Training: Maximize the likelihood of the training data
  + Classification: p(y = (+1)|x) = σ(f (x)), else 1- σ(f (x))
* Support Vector Machine (SVM)
  + Using geometric approach
  + Parameter **C**: penalty for each training point to violate the margin: **Loss Function**
  + For multiclass: **pairs of classes** or **1-vs-all**
* PROS:
  + SVM works well on **high-dimensional features**
* CONS:
  + Decision surface can only be **linear**!
    - Does not work well for **non-linear data trends**

**Non-linear classifiers**

* Idea: Transfer data into **other high-dimension space** where the data is linearly separable, then do **linear classification** -> implemented via Kernels
* Kernel: poly (degree) and RBF (gamma -> smoothness)
* Kernel SVM Classifier
  + Training: Maximize the margin of the data
    - Use CV to pick C and gamma (RBF) or C and degree (poly)
  + PROS:
    - Handles for complex classification problems
  + CONS:
    - Sensitive to kernel function
    - Sensitive to hyperparameters, especially C
    - computationally expensive due to CV
* **Ensemble Classifiers**
  + Idea: Use **multiple classifiers together** to form a better classifier (More experts)
  + **Bagging:** Training multiple classifiers from a **random selection training data**
  + **Boosting:** Training multiple classifiers, each **focusing on the errors** made by previous classifiers
  + AdaBoost (Adaptive Boosting)
    - Picks “weak learners” consequently to improve the accuracy
    - Iteration 1: weigh all samples equal and pick the **best weak learner**
    - Iteration 2: samples are **re-weighted**, increase misclassified and decrease correctly classified samples. Train **another weak learner** based on the weighted samples
    - Keen on iterating… The classifier function is a **weighted sum** of all the weak learners.
    - **PROS:** Good *generalization* performance (does not **overfit** easily)
    - **CONS:** Sensitive to outliers
  + Decision tree
    - Simple “rule based” classifier
    - Leaf nodes contains the predictions
    - PROS: Uses conjunction of rules
    - CONS: Easy to overfit (Bagging can fix this)
  + Random Forest Classifier
    - Uses **bagging** to make an **ensemble of Decision Tree Classifiers**
    - Parameters for CV: max\_features, max\_deapth
    - **PROS:** Good *generalization* performance
    - **CONS:** Sensitive to outliers

**Supervised Learning**

* Linear Regression: predict output *y* from input *x*, where y is a real number (not a class)
* Ordinary Least Squares (**OLS**)
* Feature Selection
  + **Ridge Regression**
  + Uses as shrinkage term to shrink some weights to zero
  + Select using CV (→ OLS)
  + **LASSO**
  + Forces some weight to be zero (instead of close to 0)
  + Larger gives fewer terms (due to more aggressive shrinkage)
* Removing outliers
  + **RANSAC**
    - Works with any regression model
    - IDEA: split the data into inliers and outliers and learn the model. Do this multiple times and use the one with the largest consensus (amount of inliers compared to a threshold)
* Non-linear regression
  + Polynomial regression
    - Using polynomial terms instead of linear
    - Use CV to select degree: To high will overfit the data
  + **Kernel Ridge Regression**
  + Apply **Kernel Trick** to ridge regression
    - Polynomial kernel
      * Parameters: and degree
    - RBF kernel
      * Uses gamma as smoothing parameter (small smooth, large wiggly). Best selected with CV
    - **PROS:** Closed-form solution (?)
    - **CONS:** Slow due to the need off CV
  + **SVR** kernel (Support Vector Regression) **LINEAR**
    - Borrows ideas from SVM: Bank of width around the function. Allow some point outside the tube, penalty decided by **C**
    - Can also be **Kernelized (turns into non-linear)**
      * Polynomial and RBF
    - **PROS:** Fast at making predictions
    - **CONS:** Slow to train due to iterative solution
  + **Random Forest Regression**
    - Uses average predictions over many decision trees
    - **PROS:** Fast predictions
    - **CONS:** Cannot learn a completely smooth function

**Unsupervised Learning** (Clustering, does not classify)

* Supervised learning considers input-output pairs (x,y)
  + classification: output y = 0,1,2…
  + regression: output y = R
* Unsupervised learning **only** considers the input data X
* **Goal:** Discover inherent properties in the data
  + Clustering
    - Find clusters of similar items in the data
    - Feature normalization is typically required for clustering
  + Dimensionality Reduction
    - Transfers high-dimensional vectors into low-dimensional vectors
    - Reduce computational cost, removes noise and good for preprocssing
  + Manifold Embedding
    - Project high-dimensional vectors into 2- or 3-dimensional space

**Clustering** (Unsupervised Learning)

* Parametric clustering
  + **K-means** (Circular)
    - Pick initial cluster centers (User picks **K**)
    - Assign point to closest cluster center
    - Change cluster center to be average of the points in the cluster
    - Repeat the 2 steps above and solution will converge eventually
    - **NOTE:** Bad initializations can yield poor result (local minimum)
    - **PROS:**
    - **CONS:**
  + **Gaussian mixture model (GMM)** (Elliptical)
    - User picks number of members in each cluster
    - Covariance matrix: full, diag or spherical
  + **Dirichlet Process GMM** (Elliptical)
    - Concentration parameter controls range on K values that are preferred, high → more clusters and low → less clusters
* Non-parametric clustering - **Mean shift algorithm** (concentrated compact)
  + Automatically selects the number of clusters
  + **IDEA:** Iteratively shift towards the largest concentration of points
    - Start from an initial point **x**
    - Find the nearest neighbor to **x** within some radius
    - Set **x** to be the mean of the neighboring points, and repeat
  + Run for many initial points: data points that converge to the same center belong to the same cluster, remove the duplicate centers
  + The number of clusters can be controlled by the bandwidth
  + **PROS:** Automatically selects K via bandwidth
  + **CONS:** Can be slow
* **Spectral Clustering** - What if clusters are not compact (Irregular shapes)
  + **IDEA:** Clustering with graph information
    - Each point is a node in the graph
    - Edge weight between two nodes
  + **GOAL:** Cut the graph into clusters such that weights of cut edges is small compared to the total edge weight within each cluster
    - Uses gamma as parameter: small → far away point are considered similar
* **DBSCAN -** Density-Based Spatial Clustering of Applications with Noise
  + Finds cores of high density
  + Recursively label the neighbors as core points
  + Parameters **eps:** maximum distance to be considered a neighbor and **min\_samples:** min number of neighbors to be considered a core sample
  + **PROS:** Can handle clusters of any shape, can detect outliers
  + **CONS:** Sensitive to parameters

**Linear Dimensionality Reduction** (Unsupervised Learning)

* For images: Extract features and represents them as weights
* **Principal Component Analysis (PCA)** (Vectors)
  + GOAL: Preserve the **variance of the data** as much as possible
    - Project data linearly on the basis vector plane
  + User chooses dimension of hyperplane, pick the numbers of PC:s to get a certain percentage of explained variance **(95%)**
  + **PROS:** Removes **redundant dimensions** and **noise**
* If very high-dimensional data, we can generate **Random Projections** as basis vectors instead. This reduces computation time, but reduces accuracy
  + **PROS:** Fast, preserves pairwise distances between points
* **Fisher’s Linear Discriminant (FLD)** (Vectors)
  + GOAL: Maximize class separation
  + **PROS:** Preserves class separation
  + **CONS:** Requires class information
* **Latent Semantic Analysis (LSA)** (Text)
  + Bag-of-words representation (Term Frequency)
  + Approximate each document vector as a weighted sum of topic vectors
  + Objective: minimize squared reconstruction error (similar to PCA)
  + Represent each document by its topic weights
  + **PROS:** Find relationships between terms
  + **CONS:** In the topic vector, the “frequency” can be negative, does not make sense. Does not consider end goal
* **Non-negative Matrix Factorization (NMF)** (Text)
  + Constraints the topic vector and weights to be non-negative
  + **CONS:** Can be too sparse (most weights for a documents are zero), does not consider end goal (classification)
* **Latent Dirichlet Allocation (LDA)** (Text)
  + Using a probabilistic framework to model topics and documents
  + Parameters: *n\_components* - number of topics, *doc\_topic\_prior* - smoothing parameter for the topic weights, *topic\_word\_prior* - smoothing parameter for the topic vector
  + **PROS:** Robust when dataset i small
  + **CONS:** Training can be slow for larger datasets

**Non-linear Dimensionality Reduction, Manifold Embedding** (What if data is not flat?)

* **Kernel Principal Component Analysis (KPCA)**
  + Apply a high-dimensional feature transformation to the data
  + Project high-dim data to a linear surface
  + Can use **Poly**-kerner or **RBF**-kernel
* **Manifold Embedding**
  + Reduce high-dimensional data to 2 or 3 dims for visualization
  + Trying to preserve inherent structure of the data
    - **1**, Preserve local neighborhood structure
    - **2**, Preserve pairwise distances between point
  + **Locally-linear Embedding (LLE) (1)**
  + **Multidimensional Scaling (MDS) (2)**
  + **Isomap (Isometric Mapping) (2)**
  + **Spectral Embedding (Laplacian Eigenmaps) (SE) (2)**
  + **t-Distributed Stochastic Neighbor Embedding (t-SNE) (2)**

**Neural Networks and Deep Learning**

The idea is to simulate neurons in the brain. Take inputs, multiply by weights and then sum and threshold to get binary output

* **Multi-layer perceptron**
  + Adding hidden layers between input and output neurons
* **Training algorithm for single perception**
  + Look at one point at a time. If the point is misclassified, update weight so that the point is **no longer misclassified** according to the learning rate gradient
* This only works if data is linearly separable
* Use **loss function** for each perception, optimize loss function using stochastic gradient descent
* With multiple layers we can **model more complex problems**
* Each layer need an **Activation Function**
  + Old ones: Sigmoid, Tanh (Vanishing gradient)
  + Used often today: Linear, Softmax, **Rectifier Linear Unit (ReLU)**
* Can use **Backpropagation** to re-train data that classified wrongly
* Stochastic Gradient Descent (SGD) can be used to only use a **small portion of the data set** at a time, if the dataset is large, this saves time
* It can be a good idea to stop iterating at a certain validation loss to prevent **overfitting,** can also be limited to number of iterations

**Convolutional Neural Network (CNN)**

When used on pictures a dataset, normal Deep Learning does not take into account **spatial relationship** between pixels in the image. CNN solves this problem. We can use **convolutional layers** as hidden layers in neural networks

**Regularization with “Dropout”**

During training, we can “drop” certain nodes randomly. These nodes are then not used when calculating weight updating. This prevents **overfitting** and improves **training time**

**Regularization with Weight Decay**

Adds a penalty term to the loss function

**Data augmentation**

Artificially permute the data to increase the dataset size.

* GOAL: Make the network invariant to the permutations
* Translate image, flip image, add pixel noise, rotate image, deform image and so on…
* Increases accuracy

**PROS:** Lots of parameters using stochastic gradient descent

**CONS:** Easy to overfit data due to many parameters, making it difficult to train, sensitive to initialization

**Feature pre-processing**

* Classifiers can be sensitive to the scale of feature values
  + Larger values can dominate the function
* **Normalizing** and **Standardizing** of the data is common practice
* Method 1: Scale features to have mean 0 and variance 1
  + , mean and standard of training set, use the same on the test set
* Method 2: Scale features to a fixed range (e.g. -1 to 1)

**Text pre-processing**

* Stemming
  + Convert related word into a common root word: testing, tests -> “test”
* Lemmatisation
  + Similar to stemming, groups words together: gone, went, going -> “go”
* Removing numbers and punctuation