Final Study Guide

Tuesday, December 12, 2017 8:21 PM

In this guide, I assume you understand basic logic notation, as well as first order notation.

Introduction

- Denis defines AI as the science of using knowledge to solve problems effectively and efficiently. Minsky says AI is the field of research concerned with making machines do things that people require intelligence.
- The Turing Test has a human judge, and an unknown entity. There are 2 computer terminals in different rooms. The human judge will sit at one computer, and converse with the entity via a chat window for 5 minutes. The judge decided if the entity at the other terminal is a human or computer.
- For an AI to pass the test, they have to generate coherent language, and respond appropriately to unexpected inputs. It does not have to show understanding, the ability to learn, and to store knowledge.
- The Turing test does not show a rational AI. Rational here means:
    - We maximally achieve pre-defined goals.
    - We only care about what decisions are made
    - The goals we try to achieve are expressed in terms of the utility of every possible outcome
    - Rationality logically follows as the outcome that maximizes expected utility

Intelligent Agents

- PEAS is a phrase used to characterize the problem that an agent solves. This means we have to formulate problems using the following characteristics:
    - **Performance** is how the agent is measuring its desired outcomes.
    - **Environment** is what populates the task's world.
    - **Actuators** are what the agent uses to take action within the world.
    - **Sensors** are how the agent can perceive the world around it.
- An agent perceives its environment through sensors, and acts on it with actuators. It uses an **Agent Function** to choose the action based on the information provided to it. **A rational agent function maximizes the performance characteristic of the agent.** Different types of agents do this in different ways:
    - The **reflex agent** can make decisions either using the current percept of the world it has (Simple), or an internal model it has saved of the environment AND the current percept (Model). These are only rational when the correct action is taken in every reaction it has.
    - **Goal based** agents use an action sequence to get from a current state to some goal.
    - Learning agents repeat a problem many times, or use features of states within the problem many times to decide on how they take action within the world.
- **Environments** can be:
    - Partially or Fully observable, where some or all of the info needed to act is precepted.
    - Single or Multi Agent, with agents acting with or against each other.
    - Deterministic or Stochastic, where the next state of the world is ONLY determined by the current state and agent action, or not.
    - Episodic or Sequential, where each decision made does not affect other ones later. Opposite, they do.
    - Static or Dynamic, where the world does not change while an agent chooses an action, or does.
    - Discrete or Continuous, where possible states and actions are distinct and change in steps, or don't.
    - Known or unknown, where the agent knows the rules of taking actions within the

environment, or it doesn't.

- **Utility** is defined by the **utility function**, which is an agent's preferences over possible outcomes can be captured by a function that maps these outcomes to a real number; the higher the number the more that agent likes that outcome.
- So utility is the score the agent gives each outcome that it tries to maximize. The best possible outcome.

Uninformed Search

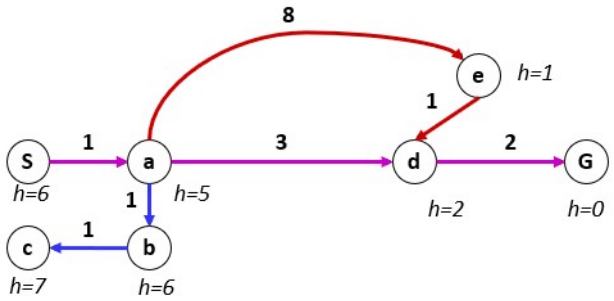
- Solving a search problem involves finding a best sequence of actions to solve a problem. Search problems consist of:
    - A state space, every possible state within the problem.
    - A successor function, that defines moving from one state to the next.
    - A start state and a goal test, where the agent begins, and terminates upon reaching a problem solution.
    - A solution, which is a sequence of actions which transforms the start state to a goal state.
- State space graphs are nodal representations of the state space. Each node is a configuration of the state, and edges are state transitions. Neighbors of a node are all possible future states from the current state. State space trees take the start state, and define all possible future states. The same occurs down the tree for each node. Thus, a path from root to leaf in the tree defines a solution.
- There are a couple of ways to build out a search tree, but they all share a common solution:
    - Initialize the search tree with the initial state.
    - Expand your potential plans to build your fringe, meaning determine the next possible node to expand from all nodes available to the initial state.
    - Expand that node.
    - Repeat.
- The search algorithms all have different characteristics:
    - Completeness, can it find a solution if it exists?
    - Optimal, will it find the path with the least cost?
    - How time complex is it?
    - How space complex is it?
- **Breadth first search** uses a FIFO Queue to manage its fringe nodes. It will start at the start node, and then add all possible connected nodes in the search graph to the queue. This way, it looks at the search tree in "tiers", with all first discovered nodes being expanded before their children shared with previously expanded nodes.
    - Complete
    - Optimal if edge costs are 1
    - O(number of nodes in initial state expansion ^ tier of shallowest goal state node)
- **Depth first search** uses a LIFO Stack to manage its fringe nodes. It will start at the start node, choose the leftmost node (or some other arbitrary node, this is due to the other initial future states being on the bottom of the stack) and continue adding it's children to the stack. The idea is it will roll through all possible sequences of actions underneath the first node it chooses (hence depth, it rolls deep into the search tree). This is due to how the stack manages the fringe, as the next node chosen from the fringe will always be the last node added. So the children of the root node tend to only be explored after an entire depth of the search tree has resulted in failure.
    - Not optimal
    - Not complete
    - O(number of nodes in initial state expansion ^ tier of first goal found)
- **Uniform Cost Search** uses a Priority Queue, and assumes the actions taken to state transition have costs associated with them. Nodes are chosen for expansion based on having the lowest cumulative cost(future transition cost plus all previous transition costs to this node summed). So the cost at a node can be defined as the cost of all the nodes before it plus the cost it took to get there. Thus, the node with the lowest value of that is chosen first for expansion out of the fringe.

The issue with this is that it doesn't focus on a path when searching, it will jump all over the fringe of the tree exploring all of its options which costs us memory and time. Meaning, it expands every node in the search graph.

- Complete
- Optimal
- Not great with respect to space or time

Informed Search

- **A Star Search** is an informed search, which means it uses information about the goal state to choose which node in the fringe it should expand. It assigns a score to each node of the backward cost (the action cost from uniform cost search, sits on an edge) and the forward cost (which is determined by goal proximity, which itself is calculated by a heuristic, assigned to nodes). This is cumulative for the entire search graph, so as A* expands its fringe, it adds the previous total back/forward cost in the path to that node to the current back/forward cost of the node it chooses to expand. It will use a Priority Queue to choose which node to expand like uniform cost. A* is optimal and complete.



- A **heuristic** a function that estimates how close a state is to a goal. These are designed for search problems. A heuristic is **admissible/optimistic** if at every node the heuristic cost is less than the true cost (edge weight from UCS) to the nearest goal. When a heuristic is admissible, it is a good heuristic. It is a property of the state, not the action taken to get to a state.

Adversarial Search

- The idea of adversarial agents exist. They seek to minimize the utility of a utility maximizing agent. This means they look at all possible future states of an agent after their turn, and try and force it to take the smallest one. This is what is known as a zero sum game, in which one player maximizes their utility, and the other player minimizes their utility.
- The minimax algorithm seeks to give a maximizing agent the best possible utility. The way it works is like so:
  - The agent will choose an action to take, but only after looking at what the minimizing action the mini agent will take. The mini agent looks at all possible actions it could take after the max agent has moved, and takes the one that will provide the terminal state (end game state) with the least utility for the max agent. Thus, it chooses from the minimum of the actions it can take for the max agent. The max agent calculates this, and chooses the max action that forces the mini agent to take the largest possible minimizing action. This algorithm is recursive, and could continue for multiple **plies**, which are when all agents have taken a turn. So there could be like 5 moves each of the mini and the max agent, and this method would still work. The best way to think about it is to see what the lowest minimizing agent in the tree will do, and feed that information back up the tree.
  - For example, the minimizing action would choose both -8 (left branch) and -10 (right branch) as they are the smallest utility values max could end up at. Then, max would choose the maximum out of -8 and -10, and go for the -8 terminal state for its final utility.
  - Alpha beta pruning is a way to make minimax more efficient. While looking at minimizing

agents, if we have previously seen a minimizing score at say, 5, and we happen across a possible state a mini agent can force, like 7, we stop evaluating that mini agent node. There is no point, we will just choose the other node, as it gives less utility. However, if we find a node less than 5, like 3, we do expand that. For the maximizing agent, if we have found a max node with say 100, and we happen upon one that can only give 4, we cut it. We would choose 100. However, if we found 3129087, we would definitely have to go for that one.

Logical Agents

- Knowledge based agents or logical agents are a little different from search agents. Search agents use **atomic states**, meaning it knows exactly what the next world state will be to a T. It uses knowledge about the goal to guide itself. Logical agents have a state space of attributive states. They know about characteristics of the next states, but not the entire state itself. These are **attributive** states, meaning they are partial and are chosen by the successor function using information about the world, not the goal. Basically, atomic has every feature it needs to make a decision, attributive has attributes about the next state, but not a picture of the complete state.
- A **knowledge base** contains logical sentences that describe the state of the world. It will change these sentences based on agent interactions with the environment the sentences describe. These are basically models, or what the agent knows about the world. It has background knowledge, or the rules of the world, and situational knowledge, what is true about the agent's current state. The sentences are:
    - Derived, or logically created based on other KB sentences.
    - Axiomatic, or given to the KB initially.
- For the following Forward and Backward chaining algorithms, all logical sentences in the knowledge bases have to be written as **Horn Clauses** only. These take the form of implications, with a large conjunction in the precedent, and a single literal in the antecedent.
- **Forward Chaining** reasons forward to infer new facts from existing knowledge. It first takes all the known true symbols in the KB and adds them to the agenda queue. It then pops off the top of the queue, and goes through all the precedents of the HC's in the KB, and checks off those known true symbols in the inferences. If all symbols in a precedent are proven true, the symbol in the antecedent is proven true and added to the agenda. This continues until the agenda is empty. Every symbol dequeued from the agenda is a proven true symbol.
- **Backward Chaining** gets a query, and reasons backwards to find a chain of inference to prove it. It finds an instance of the queried symbol it wants, and tries to prove every symbol in its precedent. So it takes those symbols, and finds their precedents, and goes on to prove those. When it finally proves the deepest precedents, it marks them as true, and uses them to prove earlier precedents until it finally gets to the queried symbol.
- In First Order Logic, when populating a knowledge base, we need concrete knowledge for inference to be done using the above two algorithms. Thus, we use **instantiation** to resolve quantifiers for inference by grounding the free variables in the quantifiers to one or more concrete objects:
    - For Universal Quantifiers, we can substitute in as many objects from the database as desired for the free variables in the quantifier. For example, if we have objects Tim, and universal quantifiers Fashionable and Smelly, we can say Tim is Fashionable instead of all free variables x.
    - For Existential Quantifiers, we can add a net object to the knowledge base to replace the free variable.
- An **ontology** is a structured framework of concepts, like database tables. A KB is like the database rows of the ontology, meaning the logical assertions about a table/object in the ontology. It is useful to structure our knowledge in an ontology, as it allows us to talk about the general properties of objects. We can categorize objects and their properties, and use inheritance to pass properties of parent categories into their subcategories to make inferences about the children. For example, if an object is a Basketball, we can infer from the Round property of the parent category Ball that the Basketball is round. We can also differentiate between a Disc object and a Ball object

using just the properties. We can also restrict relations to hold between specific categories.

Markov Decision Processes and Offline Learning

- Deterministic outcomes of actions involve a 100% probability of your next intended action actually occurring. Stochastic outcomes of actions have a probability assigned to the multiple possible states resulting from an action.
- We use **Markov Decision Processes** to deal with the various possible outcomes due to stochastics. An MDP has:
    - A set of states S
    - A set of actions, A
    - A transition function T(state, action, next state) that returns the probability that action from state leads to next state.
    - A reward function R(state, action, next state) that determines the utility of the action
    - A start state
    - A terminal state
- Remember in search problems we wanted an optimal plan, getting the best sequence of actions? Well, for MDPs we want an optimal **policy**, which gives an action to take during each state. A policy is optimal if it maximizes expected utility if followed.
- **Expected utility** with deterministic outcomes are simple. You take an action, and the utility of the resulting state is your expected utility. With stochastic outcomes, we have probabilities for all possible outcomes based on one action from a state, so we sum all utilities multiplied by their likelihoods.
- **Value Iteration** is an algorithm that uses the Bellman Equation. It runs like so:
    - Assign an arbitrary value to each state's utility. Usually you would make all states 0 except for the goal states.
    - Plug those into the right side of the Bellman equation to update all the utilities.
    - Repeat above two until utility of each state changes less than a pre-decided number/threshold.
- $$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$
- The Bellman Equation states: The new utility of a state is equal to the reward of that state plus the discount value (0 < gamma < 1) times the maximum expected utility action to take from that state.

Reinforcement and Online Learning: Monte Carlo, Temporal Difference, Sarsa and Q Learning

- **Reinforcement Learning** is like an MDP, but with some different features:
    - A set of states, S
    - A set of actions, A
    - A start state
    - A possible terminal state
    - No transition function
    - No reward function
- With reinforcement learning, we can OBSERVE the end state after taking an action from a state. We can also OBSERVE the reward for entering that state.
- The basic idea is that the agent must learn to maximize the expected rewards based on the actions it takes. All learning is based on the observed samples of outcomes it encounters.
- The difference between offline and online learning is that online must actually take the actions to update the utility based on the observed rewards and action outcomes. MDPs and offline learning does not have to do this, they know the transition function and reward functions.
- Model based offline learning means brute forcing an agent through a problem so many times we can build a model to do an MDP on it. This is inefficient.

- A **Q Function** which takes in a pair of a state and an action, and returns a utility.
- The **Monte Carlo** method is used to compute Q values for each state given a pre-defined policy.
    - You run the policy on the problem multiple times, recording the sum of discounted rewards that the agent observes at each state. The sum of discounted rewards is every reward function value generated by the policy on the state, and for all states observed after the calculated state.
    - You then average those states, using the number of times you ran the policy.
- **Temporal Difference learning** operates the same as Monte Carlo, but uses lookaheads to get the utilities for states. It follows this function:
- $$U_{t+1}^{\pi}(s) = U_t^{\pi}(s) + \alpha[R(s') + \gamma U_t^{\pi}(s') - U_t^{\pi}(s)]$$
  $$= (1-\alpha)U_t^{\pi}(s) + \alpha[R(s') + \gamma U_t^{\pi}(s')]$$
- Where the new utility of a state is (1 - the learning rate) times the current utility of the state plus the learning rate times the reward of the new observed state plus the discount value times the utility of the new observed state, which is recursively calculated based on the same function.
- This makes the difference of a faster convergence to the true utility value for every state faster than Monte Carlo could pull off, and the managing of infinite sequences is now possible. We do not have to run the policy a bunch to get the average values anymore. However, both still follow a policy.
- **SARSA** is identical to Temporal Difference, but instead uses Q Values for the chosen action in every state instead of an expected utility value of the next state:
- $$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha[R(s') + \gamma Q_t(s',a') - Q_t(s,a)]$$
- After SARSA finishes running, we can extract a policy from the data and run that for an optimal policy.
- SARSA is on policy learning, as we use a policy to decide an action to learn from.
- **Q Learning** operates like SARSA, except that instead of choosing an action based on a policy like in SARSA, it looks at all possible next actions in a state, and updates a Q value of a state based on the best possible action:
- $$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha\left[R(s') + \gamma \max_{a' \in A(s')} Q_t(s',a')\right]$$
- So we can say the main difference between on policy and off policy learning is that on policy takes an action and then learns the needed values, and off policy learns the needed values and takes the next action.
- Our motivation for exploration in reinforcement learning is to discover the most optimized policy or Q Values on a state so our agent operates optimally. If we do not explore, we may find a solution, but it might not be the most optimal solution.

Probability

- To estimate the probability of events from observed samples, we count up the observations of a category in a sample, and then divide by the total number of observations.
- In a probability table:
    - **Joint Probability** is the probability of two observations together, like the probability that the weather is hot AND sunny. Denoted as P(a,b).
    - **Conditional Probability** is the probability of one thing being true, given that something else is true. Denoted as P(a|b), and equal to the joint probability of the two events divided by the probability of the given event.
    - **Independent Probabilities** are the probability of a single event occurring within the table. So, something like P(a).
- To calculate the joint probability from a conditional probability, use the **Product Rule**:

- $P(a,b) = P(a|b)P(b) = P(b|a)P(a)$
- **Bayes Rule** states this:
- $$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$
- In the above, P(a|b) is the posterior probability, the adjusted likelihood of observing a taking observed evidence into account. The P(a) is the prior probability, or the initial likelihood of observing a before observing evidence. The observed evidence is everything else on the right side of the equality.
- Two random variables X and Y are independent if their joint probability is equal to their individual probabilities.
- X is **conditionally independent of Y given Z** if:
- $$P(X|Y,Z) = P(X|Z)$$
- $$P(catches|cavity, toothache) = \frac{P(catches, cavity, toothache)}{P(cavity, toothache)}$$
- (The second graphic just shows what P(X|Y,Z) is equivalent to).
- We use the **t-test** and **Chi Squared** tests to determine if two distributions are significantly different in a process called **hypothesis testing**.
- o evaluate the observed distribution using test statistics, you plug the observations into the test statistic, get out a p value, and compare the p value to the chosen significance level. If it is less than the significance level, we reject the null, otherwise we do not. (NOTE THIS CHANGES FOR T)
- The **t test** relies on the means of each sample set, the standard deviations of each sample set, and the size of each sample set:

Calculated as:

$$t = \frac{|\mu_1 - \mu_2|}{\sqrt{A * B}}$$

Where:

- $$A = \frac{N_1 + N_2}{N_1 * N_2} \qquad B = \frac{[(N_1 - 1)\sigma_1^2 + (N_2 - 1)\sigma_2^2]}{N_1 + N_2 - 2}$$

  - Resulting *t* value can be compared against a pre-established **critical value** for the given significance level (tables available anywhere); if greater, H_1 is supported
  - Critical values specified for **degrees of freedom** $(N_1 + N_2 - 2)$

- The t value is compared against critical values for the p value you choose that can be found in tables. If it is greater than these, reject the null.
- **Chi Squared** relies on X1 and X2, the set of frequencies of each category in each sample set. If you have just one sample set, you use N, the expected frequency of category c under the expected distribution, and Oc, the observed frequency of category c in the sample set. As with the t test, you compare chi squared against a critical value on a table, and if it is greater than the critical value, reject the null.

Calculated as:

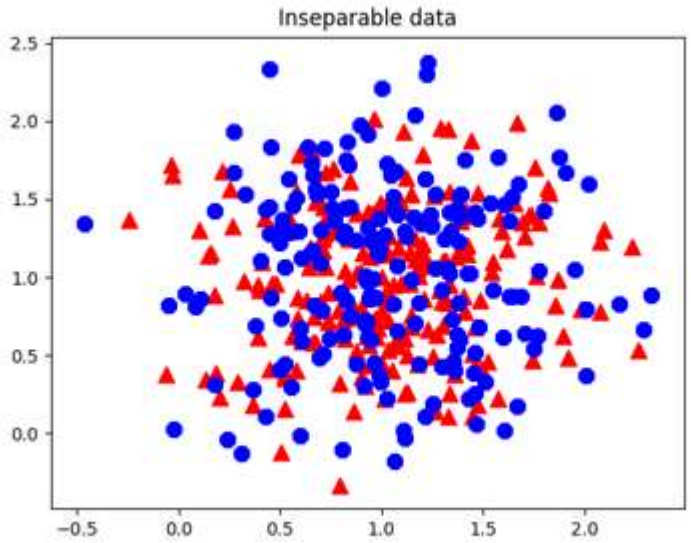
$$\chi^2 = \sum_{i=1}^{|C|} \frac{(X_{2,i} - X_{1,i})^2}{X_{1,i}}$$

- Where $X_{1,i}$ is the frequency of category i in X_1 (and the same for X_2)
  - Note that this is **not** symmetric; if using two actual observed distributions, X_1 is considered the "expected" distribution
  - As with Student's t test, compare calculated χ^2 value against the **critical value** specified for the degrees of freedom (here $|C| - 1$)
  - This breaks if any expected $X_{1,i} < 10$

Machine Learning

- Machine Learning follows a specific idea. That is, given pairs of observed inputs and outputs for some distribution, learn a function such that f(input) = output for every input output pair in the distribution. This function is called the **model**.
- There are three kinds of problems in Machine Learning:
  - **Classification,** which takes data points with some categorical label, and learns to predict that label using the features of the input data.
  - **Regression**, which takes data points with some continuous label, and learns to predict that label from input features.
  - **Clustering**, which takes data points with NO label, and finds a descriptive structure in the data, meaning it finds the data points similar to other data points.
- To learn these things, we start with an initial model, or our prior, sample data and get feedback on how well our model works on describing the samples , and use the feedback to adjust the model for a better posterior representation.
- **Supervised Learning** takes in data inputs with correct outputs to train the model, then inputs test data to get the predicted output data the model generates, then uses the error between the prediction and the training data to update the model. Classification and regression are supervised.
- **Unsupervised Learning** takes inputs only, describes patterns in the data with the current model, and updates the model to get better patterns. Clustering is unsupervised.
- When we design a model, we have to consider the model structure, and the coefficients of the model. The structure is the general form or family of the model (think trig functions versus linear functions), and the parameters are how we manipulate this general form. We do a lot of work in designing models using these attributes so our model can generalize to new data well, and can be complex enough to properly model data.
- We also need to consider how we pull information from out input data as well. We use **feature functions** to do this, numerical data that describes input data. These functions define the parameters of the model we design. We want features to be:
  - Informative to the task at hand
  - Easy to compute
  - Not redundant considering other chosen features
- The **KNN, k nearest neighbors** classification algorithm is a *nonparametric* method, meaning it uses a model structure, but no explicit parameters to learn. The algorithm goes like:
  - Given an input test point P, find the k (k is user chosen) points in the training set with the smallest distance to P
  - Count up their labels as votes
  - Label P as the class with the highest vote count
- **K-Means Clustering** is unsupervised learning, with the goal of describing the data with k clusters, and each point in the distribution belonging to the center whose centroid its most similar to, where the centroid is the average of its points. The algorithm goes like:

    - Initialize your k cluster centroids, with are uniformly random within your distribution or random points in the distribution/dataset. Both have their downfalls. A poorly chosen random point can be far away from your data and useless, but uniform distributions can break based on random chance with your dataset.
    - Assign all points to the cluster with the nearest centroid.
    - Update the centroid to the mean of the new points in the cluster.
    - If any cluster labels/assignments changed, GOTO 2. If none changed, end algorithm.
- **Gradient Descent** takes the parameters of a linear regression and updates them by iteratively finding the local minima of the linear regression function. This iteration continually takes partial derivatives of the model to find where the slope is steepest, and updates the model to morph towards that steepest location, where it will update until convergence, which is when it finds the local minima it was searching for. The updates are made on the model parameters.
- A **logistic regression** is a differentiable classifier model, a la Supervised Classification learning. It updates a function that separates data within a dataset into separate classifications using gradient descent. A **linear regression** is a differentiable continuous model, a la Supervised Regression learning. It updates a function that takes input data and returns what output it should be.
- Sometimes too much error exists in a set of data, and it cannot be split into classifications using logistic regression:
-



- A **neural net** consists of two types of things. Linear transformations and nonlinear functions. You have to alternate these to build up a net. A perceptron takes in inputs, and gives a functional output. We can say this input is f(x1,x2) = w1x1 + w2x2 + w0. Before the perceptron does the function on the inputs using the weights assigned to the inputs, this function is done before it is fed to h(z), the actual perceptron. The perceptron then returns a vector based on the input function. So in summary, the first layer of a network takes a linear transformation as input into its perceptrons, and the resulting scalar is passed into the nonlinear function of the perceptron to return a final scalar. Multiple perceptrons return multiple scalars, which can all be put into a final vector at the end of the operation. After a specific input point is passed through the net, we calculate the error from what our prediction is versus what output data is. To do this, we use the error/loss function (prediction-result)^2.
- We then do **backpropagation**, where we change the weights based on the loss function to train the neural net.

Experimental Design

- In any experimental design, we should first understand the data we are working with. What are we looking at, how can we describe it, what format should our output be
- We then create a broad hypothesis, such that it can be later converted into one or more specific

and testable hypotheses. These can then have experiments designed for them and can be evaluated. The broad is not testable.

- To have a testable hypothesis, we have to find the reference distribution and a way to evaluate correctness. To find the reference distribution, we compare against a baseline system, like a random baseline or majority classifier. To evaluate correctness, we need to compare the predictions made by the system to the labels we already have somehow, usually by using accuracy and precision
- A good example of a testable hypothesis follows the format: "Using testing method, ML algorithm trained on percentage of the data will make different and better decisions on the remaining percentage than some baseline you choose."
- We also need to set a **control** in an experiment. This allows us to handle other potential sources of variation in our data. Some sources of variation that we can limit are biases in the data itself, as well as sampling bias, which is when you sample more from one sample than another.
- To perform McNemars test (to see if we are getting significantly different results from a baseline distribution using our new model):

With contingency table like:

		S_1	
		T	F
S_2	T	a	b
	F	c	d

Calculate test statistic as:

$$\chi^2 = \frac{(b-c)^2}{b+c}$$

- Roughly follows a χ^2 distribution with one degree of freedom
- So calculate the test statistic and compare to χ^2 critical value for desired level of significance

- We use this to see if the results are significantly different from the baseline, but not to see if they are better. We do this with performance metrics and a ground truth dataset:

		Ground Truth	
		T	F
Predictions	T	TP	FP
	F	FN	TN

TP = True Positive TN = True Negative
FP = False Positive FN = False Negative

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \qquad F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

- Accuracy says how well you got the correct labels overall, precision says how predictive you were in making positive decisions, recall says how sensitive you were to finding all positive cases, and f1 combines both precision and recall. Higher accuracy and precision is good.
- You have to run the process of comparing the two samples from the distributions (the test data derived from your training of a ML algorithm), (random baseline or other) and how they agree/disagree on specific labels for every testable hypothesis you get from your broad hypothesis.
- To choose a good model, look at others or just try a bunch!

Speech And Language Applications:

- The goal of these notes is to familiarize me with a bunch of different applications of AI methods, so you can figure out what's next. Today, we discuss speech processing and natural language processing.
- You need to be able to describe these applications, not implement them. An example of the homework we would be given is to design experiments around these applications. This ties into last week's experimental design.
- Two kinds of sound collection exist, clean speech and dirty/noisy speech. Noisy has a lot of background noise.
- Data for speech can come in many forms, like raw waveforms, spectrograms, and binned frequency statistics (reading different frequencies). All of these represent the same thing.
- There are different kinds of data, like single and multi-speaker speech, as well as children vs adults and conversational vs read.
- There are three main steps to **Speech Recognition**:
    - Puts the signal into phonemes, which are meaningful sounds within a language. English has like 152.
    - Tries to take the phonemes and turn them into words through pronunciation modeling. It will give a lot of different choices for phonemes on each phoneme due to similar sounding phonemes. After this is done, actual words are formed.
    - Language modeling takes all of the predicted words and puts them into actual sentences.
- Audio frames are sequential. We use a sequential model like a Hidden Markov Model to label each audio frame with our predicted phonemes. HMM treats the frames as observed data with nodes, and assigns hidden states to these nodes containing the phonemes. You can think of it like a row of observed nodes, and columns of nodes above each of the observed nodes that contain every possible phoneme. All phonemes have probability edges between each other from left to right, and all have probability edges from themselves to the observed data. Your goal is to calculate both the transmission and emission probabilities. To calculate the transmission, you use training data and a neural network on the waveform frames to predict the chance the next frame is a specific phoneme. To calculate the emission, you have to train a neural net to identify phonemes from raw waveforms.
- You have a lot of experimental considerations in ASR, like noise, speaker-mic distance and room characteristics (think reverb), and speech characteristics.
- **Speech Separation** follows this process: When two people are speaking into a mic, the actual waveform will compose multiple complex waveforms. From this, splitting up the two people involves extracting the individual complex waveforms from the composite.
- To do this, we want to recover the clean signal. We do this by masking part of the signal to try to remove the parts we're not interested in. The binary mask is a [0,1] multiplier, meaning something can even be 1 or 0. You learn the mask (where to apply it at which frequency frame) with ML. You would then apply the mask to the composite waveform to separate the speakers.
- **Speech synthesis** takes in a textual input and outputs synthesized speech. The steps are:
    - Decompose your words into sequences of characters.
    - Convert sequence of characters to sequence of phonemes. These are context sensitive. This also has to be done sequentially, and is not one to one, pronunciations differ regionally and contextually.
    - Compose your phonemes with prosodic signals. Prosody relates to pitch, speed, and intonation. It is the tonal quality of your speech. The speaker and situation change the prosody.
    - Concatenate prosodic phonemes to an utterance. You need smooth transitions to avoid clicking between pasted together signals.
- Natural Language Processing involves taking written language and processing it.
- **Speech/POS** tagging involves taking in an input of words and outputting a sequence of part of speech tags. This means each word in the input is labeled as a verb, a noun, adjective, etc etc.
- You can use words like "the" as a baseline, but HMMs are also used for this. The transition probabilities are the probably that you go from something like a noun to a verb, or that a word is a

verb given the previous is a noun. The emission probabilities are the probabilities of a word given a POS type.

- **Information extraction** extracts structured info from unstructured text. We want to fill a database with scientific names of animals, and we will use the Wikipedia articles of those animals for example. This can be rule based, meaning heuristic rule based approaches can get you pretty far with pattern matching. Basically, we take in unlabeled text as input and we get database entries as output.
- A more general way to do this is to mark entities referred to in the text that a relevant to the data you want to extract.
- You then have to recognize relationships between the entities. This is the task of relation extraction, where we classify the relations between the identified entities within the text.
- You then match this to database rows and columns.
- **Information retrieval** is the flip of info extraction like in audio analysis. It takes an input like a query (structured data/unstructured text) and outputs a list of records, ranked by relevance to the query
- IR is more than just language, it runs PageRank in a structured way, and medical document retrieval systems based on keywords
- Running IR experiments can have some big questions, like how well does my model generalize, and do frequency models change my results

Vision and Robotics Applications

- Classifying image data takes in a lot of data, like single images, video, different color formatted photos etc. This data has many factors like lighting, distance to subject, lens shape etc
- We can use computer vision for image classification, motion tracking, gestural control like on Kinect etc
- **Image classification** takes in a single image, runs it through a classifier, and outputs a label describing image content. To extract features, many things can be done:
    - We can detect edges within the picture to get data. We can detect edges by finding sharp changes in pixel value gradients within a chosen threshold
    - Region segmentation breaks images into contiguous regions of color, and this is done with k means clustering among others
- We can use convolutional neural networks to learn HOW to extract useful features for classification like edges, faces. We do this with local connections, meaning we look at specific regions, or windows of the image that map directly to nodes in the neural net. These neural nets are called filters.
- It is good to use different filters on the same image, which are just linear combinations of pixels. Each filter should learn a different feature
- So the actual process of image classification is done by extracting features from labeled images, and plugging them into a classification model like a logistic regression or neural net (again)
- The main idea of **gestural control** is to use hand gestures to control computer applications, like tracking a user's hand in 3D. This needs to interpret pre-defined gestures as discrete commands, using one or more cameras in real time.
- In **robotics** you have to deal with problems in your:
    - Physical environment, like not seeing all the stuff that's there
    - Physical components, which are unreliable and can fail
    - Sensors, and their limited ability to sense stuff
- Robotics have two settings, industrial and swarm.
    - Industrial robots are in a controlled environment. The agent has little movement, knows what other agents will be around, and knows the physical conditions. They are highly specialized for a single task with hard constraints on i/o. Occasionally they are used for multiple tasks in a variable environment, like warehouse robots that pick up goods.
    - Swarm robotics have hundreds of small robots operating together. Their sensors are limited and noisy, can compute very little, and can communicate with other agents in the swarm.

The idea behind these guys is to support complex environment interaction with a minimal amount of resources

Philosophy

- The **mechanical definition of AI** basically covers the Chinese Room Experiment, where a man in an obscured room takes in Mandarin text, uses a codebook to directly create a response to it, and passes it outside of the room. The main questions were if the operator understands Mandarin, and if the system understands Mandarin. Class consensus on this leaned towards no for both, the operator because he is translating symbols without understanding of the text, and the system because it could not formulate original thought. Arguments for this room as a mechanical definition of intelligence states that if the system can output impressive results correctly, it can be directly conversed with and it formulates rational speech, it has a sort of intelligence.
- When we talk about **AI as a hierarchy of single task components**, we are talking about the **Society of Mind,** which states that "What we call intelligence can be a product of the interaction of non-intelligent parts". This sets an AI agent as a society, with each node in the society being a smaller agent responsible for a single task. The **agency** of a society is the observed behavior of all of these agents operating together. Minsky argues that while none of the agents are intelligent, the control flow and organization the society uses makes the agency it creates intelligent.

Ethics

- There are a lot of ways you can make a misstep in your ethical behavior when designing an AI.
- With data collection, we have to ask if our process of data collection is ethical. An engineer's responsibility is to collect enough data to train the model from a population that represents the target. Important questions include:
    - Are you capturing the diversity of the real target population? Collection only one side of data in an argument biases the model.
    - Are you collecting more data than you need/should?
- Building the model structure around assumptions about the data distribution has these questions:
    - Is it possible to harm users with your assumptions?
    - Are the features you use bias. Like number of traffic stops by police because it will bias towards African Americans.
- Using the trained model to make predictions on data from the right kind of distribution has these questions:
    - Is the purpose for the AI ethical?
    - Is the way that you're getting/processing predictions ethical?
- Some AI ethics violations include the Zo bot from Microsoft, a teenage girl simulation that almost immediately became a horny Nazi due to the internet being the internet. Failures in ethics took place in the creation of the model structure, due to Microsoft's inability to reject data that would tip Zo off to the wonders of Mein Kampf or the Kama Sutra.