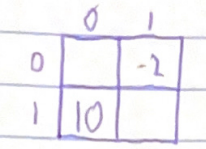
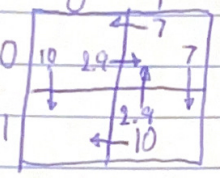
Exam 2

Evan Nuss

1a.





$Q((0,0), down) = 10$

$Q((1,1), left) = 10$

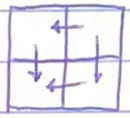
$Q((0,1), left) = 0 + .7(10) = 7$

$Q((0,1), down) = 0 + .7(10) = 7$

$Q((0,0), right) = -2 + .7(7) = 2.9$

$Q((1,1), up) = -2 + .7(7) = 2.9$

optimal policy:



~~pick random action~~
~~reward = state reward + discount_factor * argmax(rewards for reachable next_states)~~

b. state = initial_state
Do until optimal policy determined
state. pick_random_action
Q ~~reward~~(state, action) = $reward_{new\_state}$ + discount_factor * highest Q-value for leaving new_state
State = new_state

2a. centroid 1 = (1, 1)
centroid 2 = (1, 2)

$dist_{AC} = |1-3| + |2-5| = 6$ $dist_{AD} = |1-5| + |1-6.5| = 9.5$

$dist_{BC} = |1-3| + |2-5| = 5$ $dist_{BD} = |1-5| + |2-6.5| = 8.5$

cluster1 = {A}
cluster2 = {B, C, D}

centroid 1 = (1, 1)

$\frac{1+3+5}{3} = 3$ $\frac{2+5+6.5}{3} = 4.5$

centroid 2 = (3, 4.5)

$dist_{c2-B} = |3-1| + |4.5-2| = 4.5$ $dist_{c1-B} = |1-1| + |1-2| = 1$

$dist_{c2-C} = |3-3| + |4.5-5| = 0.5$

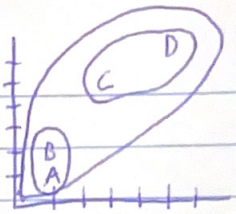
$dist_{c2-D} = |3-5| + |4.5-6.5| = 4$

cluster 1 = {A, B} centroid1 = (1, 1.5)
cluster 2 = {C, D} centroid2 = (4, 5.75)

b. $dist_{AB} = \boxed{1}$ $dist_{AC} = 6$ $dist_{AD} = 9.5$

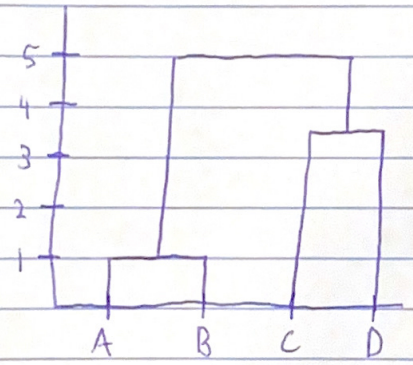
$dist_{BC} = 5$ $dist_{BD} = 8.5$

$dist_{CD} = |3-5| + |5-6.5| = 3.5$



$dist_{BC} = 5$ $dist_{BD} = 8.5$ $dist_{CD} = \boxed{3.5}$

$dist_{BC} = \boxed{5}$



3a. Genetic Algorithm

```
t = 0
initialize population P(t)
Evaluate P(t)
Do until given stopping criteria is met {
    t += 1
    Parent_selection P(t)
    recombine P(t)
    Mutate P(t)
    Evaluate P(t)
    Survive P(t)
```

b. Assumptions:

- The data is represented such that the attributes of the genome are their own chunk within the genome, meaning any crossover & mutation must not affect only a part of the attribute.
- I will use 2 point crossover & any mutation will be single point. That part & it's new value will be picked at random.
- Parent selection will be based on the pizzas with the best fitness score. However, to maintain diversity, a lower performing pizza will be selected at times.
- Survival will be determined by fitness score. The pizzas with the best fitness scores will survive, as well as some with lower scores in order to maintain diversity.

	crust	amount of cheese	meat/veggie ratio	# of pepperoni
~~P_1~~	~~[thin~~	~~high~~	~~4:1~~	~~20]~~
~~P_2~~	~~[stuffed~~	~~medium~~	~~1:6~~	~~2]~~
P_3	[thick	high	2:1	10]
~~P_4~~	~~[thin~~	~~low~~	~~1:9~~	~~0]~~

- fitness score is on a scale 0-10

~~$F_1 = 7$~~ ~~$F_2 = 5$~~ $F_3 = 9$ ~~$F_4 = 3$~~

- P_2 & P_3 will be parents. We will do 2 point crossover & single point mutation.

P_5 = [stuffed, high, 1:6, 10] => [stuffed, high, 1:3, 10]

P_6 = [thick, medium, 2:1, 2] => [thick, medium, 2:1, 10]

- $F_5 = 9$ $F_6 = 8$
- Kill off P_4 & P_1 to maintain population of 4
- P_2 & P_5 will be parents

P_7 = [stuffed, high, 1:6, 10] => [thin, high, 1:6, 10]

~~P_8 = [stuffed, medium, 1:3, 2]~~ => [stuffed, low, 1:3, 2]

~~$F_7 = 8$~~ Kill off P_8 & P_2

~~$F_8 = 2$~~

population = {P_3, P_5, P_6, P_7}

4a. prior_p = 4/7 prior_n = 3/7

L => likelihood

L_senor_p = 2/4 L_soph_p = 1/4 L_fresh_p = 1/4

L_med_p = 3/4 L_hi_p = 1/4 L_lo_p = 0/4

L_true_p = 3/4 L_false_p = 1/4

L_senor_n = 2/3 L_med_n = 1/3 L_false_n = 2/3

Find posterior for each output class based on new point

posterior_p = prior_p (L_senor_p)(L_med_p)(L_false_p)
= 4/7 (2/4)(3/4)(1/4)
= .0536

posterior_n = prior_n (L_senor_n)(L_med_n)(L_false_n)
= 3/7 (2/3)(1/3)(2/3)
= .0635

probability_p = $\frac{.0536}{.0536+.0635}$ = probability_p = .458

probability_n = $\frac{.0635}{.0536+.0635}$ = probability_n = .542

b. The assumption is that using only past data will be enough to get accurate probabilities given a new datapoint. It is "naive" of the current data.

5a. Un-normalized attributes will cause points to be significantly further apart, esp if there is a feature with a very wide range of values. This will mean distances between points are often much higher which will decrease the effect that feature has on the predicted class, which could be very detrimental to the model's accuracy, especially if using inverse distance weighting.

b. An irrelevant feature that is one of the k nearest neighbors will still have a say in what the output value ends up being. It could have as much as, or even more influence on the final prediction, as a very relevant feature, depending on how close it is to the point to be classified. This influence will be very detrimental to the model's accuracy.

6a. Bagging:

```
b = # of models
for x in range(b):
    training_sets[x] = uniform randomly sampled subset of dataset
for i in range(b)
    models[i].train(training_sets[i])

for model in models:
    outputs.append(model.predict(whole_dataset, datapoint))

output = class that appears most in outputs
```

Boosting:

X = dataset

D = distribution of how likely each datapoint is selected.

For each model:

subset = sampling of x using D

model.train(subset)

Update D based on points model incorrectly predicted

For each model:

vote = model.predict * model.training_accuracy

votes[i] += vote

~~class~~ output_class = argmax(votes)

b. High variance comes from letting noise affect the model's predictions too much. By using bagging, the many models each have an equal vote in the output class, which means a model with high variance doesn't have a disproportionate influence & if its vote is incorrect, that will become negligible since the majority of the other models will not have the same predicted value. The results with high variance are unlikely to make it through the voting system.

c. If enough of the models end up overfitting & having really good accuracy on the training set, they will have a much stronger vote on the output class & could cause the whole ensemble to overfit. Whereas, the models that don't overfit might have worse training accuracy, & therefore not as much say in the final output class decision. This is b/c in boosting, votes are weighted by the models training accuracy.

KNN

Manhattan distance: $|x[0] - y[0]| + |x[1] - y[1]| + \dots$

1. Calculate distances between new point & old ones.
2. Choose the K closest points (smallest distance).
3. Whichever class has most points in list of K neighbors wins.

Distance Weighting

For each class:

votes[class] += $\frac{1}{\text{neighbor\_distance}^2}$

Regression

For each distance-label pair:

weight = $1/\text{distance}^2$

weights.append(weight)

values.append(label * weight)

output = $\frac{\text{sum of all values}}{\text{sum of all weights}}$

- irrelevant features have big effect. Dimensionality & distances tend toward infinity.
  - use PCA, filters/wrappers, attribute weighting, decision tree to trim features.

RBF

For each prototype node:

pick whichever is greater, 0 or 1 - manhattan_dist(datapoint, prototype)

outputs = rbf-results * weights. whichever output value is greatest, that's the class for that point.

Use delta rule to calculate weight changes.

$\Delta w_i = C(t - \text{outputs})x_i$

data mining

identify task, gather & prepare data, build & evaluate model(s), deploy model(s) & evaluate business related results, repeat in order to improve model

un-labeled data use clustering

HAC

- manhattan dist between each point. shortest distance. form 2 points form cluster
- repeat depending on single link/complete link.

Silhouette

$a(i)$ = avg distance between i & all other points in cluster

$b(i)$ = calc dist btw i & points in other cluster

$S(i) = \frac{b-a}{\max(a,b)}$ S(cluster) = avg(S(points in cluster))

- precision: organized as you reduce # of false pos
- recall: how to reduce # false negs.

Naive Bayes / Bayesian learning

- set priors of each output class (class instances / total instances)
- for each output class, set likelihood of that class given certain attribute. total with that attribute / total of that class.
- do posterior of each class = prior of class * likelihood of attribute * likelihood of attribute
- normalized probability of each class

total = $\sum_{i=0}$ posterior$_i$

$\text{prob}_{class} = \text{posterior}_{class} / \text{total}$

K-means

- pick initial centroids
- calc distance btw each centroid & each point
- assign point to cluster based on which centroid it's closer to.
- recalc centroids by taking average of points in cluster.
- repeat until the centroids no longer change.

Q-learning

Q(state, action) = reward of state you'll be in next + discount factor * highest Q leaving the square you're going to.

Genetic algorithm

$E = \Delta$

init pop $P(t)$

Eval $P(t)$

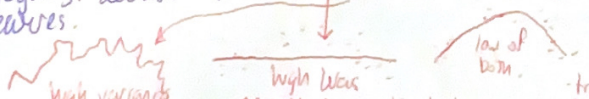
until done:

$t = t+1$

parent selection, recombine, mutate, evaluate, and survive.

Ensembles

- bias error causes underfitting, comes from bad assumptions.
- variance error causes overfitting, comes from paying too much attention to noise.



Bagging

- uniformly randomly sample dataset b times
- b = # of models. Train each model independently on its dataset.
- then for predicting, each model gets whole data set & sees unweighted vote on class.

Boosting

- x dataset. D determines how likely each point is to be chosen.
- train a model, then update D to be higher for misclassified points to be picked by next model.
- each model votes on class, it's vote is scaled by its accuracy for its turn.

Overall guesume except when we have a bunch of models because it then becomes very expensive.