**Data Science 2**

**Fall 2022**

-Check expected knowledge document on eLC.

**CHAPTER 1:**

**LECTURE 1 – 8/22/22**

Types of prediction tasks: Binary classification (spam or no), regression (variables -> output scalar value), others (multi-class classification [img classification], ranking, structured prediction [translation], etc.)

Supervised learning = has labels; unsupervised = no labels.

Feature extraction: For input x, produce a set of (feature name, feature value) pairs.

Feature vector: vector of feature values from feature extraction.

Generalization: We want the model f to work for instances not seen in training data Dtrain.

**Classification: k-Nearest Neighbors**

Problem: For input (an email for example) x, construct a filter f to that outputs a classification y (such as spam or non-spam).

Solution: A new input is classified by a vote of the *k* closest neighbors in the dataset.

**Equation** for euclidean distance in N dimensional space:

d(x,y) = sqrt(sum of ( for each dimension(xi-yi)^2 ))

or, sqrt of the sum of squared differences between each point’s coordinate in each dimension.

Or,

D(x,y)=(Sum|xi-yi|^p)^(1/p) when p=2.

If p=1, you get the taxicab distance (for a grid)

If p = infinity, you get the largest single dimension distance between the two points

**Methods for calculating distance:**

- d(x1,x2)= 0  x1 = x2

- d(x1,x2) = d(x2,x1)

-Triangular inequality

**Norm:** Denoted as ||x|| for the norm of x

The norm of any number is >= 0

||x+y|| =< ||x|| + ||y||

||ax|| = |a| \* ||x||

Norm expresses the length of vector x, or the distance from x to the origin point using an abstract measurement method.

Distance between x and y = ||x-y||

**lp unit ball:**

l1 unit ball is a diamond, l2 unit ball is a circle, linfinity unit ball is a square

**So Euclidean distance is synonymous with L2/P2/circle space, but in some scenarios you need to measure distance differently/not in a straight line, so you can substitute different values of L/P into the distance formula to get different measurement methods.**

**Linear classifiers:**

Each input value x has a feature vector.

Each feature of x has a real number j

A classifier will have a weight vector w of real numbers that assigns probability weight to each features.

Prediction score= weighted sum of features

**Hyperplane/**Boundary line: set of points where predicted score is neither positive or negative. Perpendicular to the weights vector w and passes through the origin.

3 Steps in a linear models:

-Linear predictors

-Learning objective

-Optimization

Loss function L(x,y,w) outputs an error measurement you want to minimize

**Prediction score =** w^T \* x

**Margin =** prediction score \* ground truth

If your prediction score has the same sign as the ground truth, your prediction is correct. If your margin is negative, your prediction was wrong.

**Zero-One loss:**

**Linear Regression:** Different from linear classification, outputs a scaling number rather than a class id.

**Residual:** Difference between the prediction score and the ground truth, or (w^t \* φ **(**X)) – y.

**Squared loss = (**(w^t \* φ(X)) – y)^2

To find the line of best fit in a linear regression model, evaluate all training points with the loss function,

Overall **training loss** = average loss for each point in the training set.

**Matrix calculus introduction:**

For a given function “y = wx”, it follows that “dy/dw = x”.

Example 2: “y=w1x1+w2x2”

Dy/dw1=x1, dy/dw2 = x2.

Gradient wy =

[dy/dw1,

Dy/dw2]

=

[x1,

X2].

Example 3: y = w^T x, the gradient is [w]y = [x]

▽w a \* w = a

▽w ||w||22 = ▽w w\*w = 2w

▽w wT Cw = \*(C+CT)….

To optimize min(w) Ltrain(Dtrain,w):

-Closed-form solution (only works sometimes)

-**Gradient descent**: Iteratively find the fastest direction moving w that decreases Ltrain, move w towards that direction, and repeat until reaching a stable w.

Xw =

[ - xT1 -] ( [Xtn w – yn]

…

= Sumi (xi T w -yi)^2

**8/30/22**

Training loss on linear models:

Evaluating model performance:

Measurement error: unintentionally added noise when taking in data.

Overfitting: For example using a polynomial function with too high of an order of a polynomial (4th degree if 2nd degree needed for example). Training loss will be very low, but an overfitting model will not accurately be able to predict additional data points.

**In-sample error / Ein:**

**Out-of-sample error / Eout:**

Lower Ein results in higher Eout

If Eout > Ein, overfitting is probably occurring.

On a good model, the Ein and Eout curves should both decrease, but there will be a transition point marking where the model has become too complex and the Eout curve will begin to increase.

f\* = oracle model. If the underlying f\* complexity is high, but you don’t have enough training points, it’s not worth using.

Potential/capacity of a complex model is higher than a simple model if you get enough training data.

Estimating Eout: Split data into training and evaluation…

**k-fold cross validation:** Train k times on n-(n/k) points each (leave k points as evaluation data out of every n training points).

**Causes of overfitting:**

-Modelling function is inappropriately complex

-insufficient training points

**Solutions to overfitting:**

-Get more training data

-Use a less complex modelling function

-**Regularization:** constrains the model so as to remove some of the noise. Can have side effects. Try to constrain weight coefficients to be as small as possible (within a budget C) without explicitly assigning some of them to 0.

L\* = new training loss model for regularized model.

min L\* = Ltraining(Dtrain, w) + Sum(wt 2) (L2/ridge regression) OR Ltraining(Dtrain, w) + Sum|wt| (sparsity model)

**9/1/22**

**Soft Order Constraint:**

Let the learning choose a weight within a set budget rather than explicitly setting weights to 0. v ----- L2 norm regularization/ridge regression

Equation: L = Ltrain(Dtrain,w) + ||w||22

Ltrain = Sum(wT z – y)2 = ||ZwT – y||22

**Oracle:** The center point in the middle of all contours, has 0 loss. If you use it as a solution your model will likely be overfitted, so you give a budget for your model to choose weights within which constrain it from going all the way to the oracle. We don’t know the oracle before training.

**Contours:** All points on the same contour have the same loss value. Each contour closer to the oracle that you get has a lower loss. Values are usually sourced from the least squares error function.

The vector from the origin towards our approximation of the oracle with the maximum length allowed by our budget constraint is known as w, and any line of other length along w will be λw, where λ is a multiple of the vector w. λ controls overfitting/underfitting, generally λ=0 will result in overfitting, and a large λ will result in underfitting. You can determine which λ you should use by minimizing Eout.

**Posterior probability:** Choose y that maximizes P(y|x1,x2,…,xd).

**Bayes Theorum:** P(y|x1,x2,…,xd) = (P(x1,x2,…,xd|y)P(y))/P(x1,x2,…,xd)

-Assumes conditional independence, hence “naive” Bayes Classification.

-Arm length and reading skills are conditionally independent given age.

-If you assume conditional independence, the data you need to store/compute scales linearly as your data “n” grows, as opposed to exponentially without conditional independence.

-The stored intermediate data on a naive bayes classifier are the individual weights of probabilities for each P(xn|y).

-P(x|y) = Probability of x when given y.

-We want to find a y that maximizes P(x|y) which = P(x1|y) \*…\* P(xn|y) \* P(y)

**Handling scalar variables:**

-Discrete value methods: partition data into tiers, classify each entry based on what bracket/tier it falls into.

-Gausian method: Find the mean value and variance of the scalar variable in the dataset for each classification outcome (Yes or No), construct a normal distribution around that mean for each. These distribution curves will give an output y between 0 and 1 for any x entered in, and we use the models to predict the probability of an input data point being Yes or No using the appropriate of the two models for each calculation. Whichever outcome has the highest probability is chosen as the result.

-If you get 0 a probability in each model, replace the zeroes with a very small non-zero number so both results aren’t 0.

Hyperparameter: an alpha that we don’t know what the optimal value is.

**Decision trees:**

**Internal node:** Non leaf node, attribute value is tested against the decision tree.

**Path:** Traced from root node to any leaf node through any number of internal nodes.

Benefits of decision trees: Fast to train, intuitive, superior accuracy even compared to deep models in some cases.

Types of tree models: ID3, C4.5, CART.

Decision tree training: Greedy (non-backtracking), top down from the root to all possible leaves, recursively partitions the tree into smaller subsets as it is built (divide and conquer).

**Entropy:** Measure of uncertainty associated with a random variable.

-For a set of m values of variable y (y1 ,y2, … ,yi)

-entropy = - Sum(yi log2(yi))

-log of yi will always be negative.

-If you have a perfectly evenly split distribution of values for the random variable (like a conflip), entropy will be equal to the number of bits needed to encode the number of cases. Coinflip would be entropy = 1; 4 sided coin would be entropy = 2; 8 sided coin would be entropy = 3. If any of the possibilities have a probability of 1, then that entropy has an entropy of 0.

Certain outcome = small entropy. Uncertain outcome: high entropy.

Dj = a subset of D where the instances have aj as their value of A

**ID3:**

Info(D)= - Sum(yi log2(yi)). Entropy of the spread of the output/target variable across the dataset as a whole regardless of attributes.

Infoa(D)= Sum from j=1 → j=v (|Dj|/|D| Info(Dj)) where attribute a has v different values/members.

InfoGain: Gain(A) = Info(D) -Infoa(D). This basically says that the entropy before picking an attribute – the entropy when you split on that attribute is the information gained. We want to choose an attribute that has a large info gain as the next deciding attribute.

**9/13/22 Model evaluation metrics:**

**9/15/22 Metrics continued:**

Precision = TP/(TP+FP)

Recall = TP(TP+FN) = TP/P

F1 = F measure = 2\*precision\*recall/(precision+recall)

F2 = Fb measure = (1+b2)\*precision\*recall/b2\*precision+recall **Check this formula**

Specificity = TN/N

Sensitivity = TP/P

Accuracy =

Error rate =

**Deep learning 1: Neurons**

Example of a limitation of a linear models is XOR (checkered pattern on graph)

Neural networks:

A hierarchy of linear models + nonlinear activation functions.

Any target function f that can be decomposed into linear separators can be implemented by a 3 layer MLP.

**9/20/22 Unsupervised learning**

**Basic Concepts:**

Classification: Training a model (parameterized by w) to fit the ground-truth label y given a data instance x. L(x, y, w)

**Partitioning Methods:**

Partitioning: Given a data set D of size N, we want to divide D into K clusters.

**K-means**:

Each cluster Sk has a centroid uk

Let d(x1,x2) be the distance between x1 and x2.

To start, centroids are all randomly assigned a position.

Loop:

Each data point is assigned to the closest centroids cluster.

Centroid positions are updated.

Break the loop when no data points change clusters in an iteration.

**Example:** Ma 1,1 2,1 1,2

Mb -1,-1 -2,-2

ua = 4/3, 4/3

ub = -1.5, -1.5 **Done.**

To minimize having terrible luck assigning initial centroids, we can assign each object from D to a random cluster S and then calculate the intial centroids from there.

**Example:**

Ma = (1,1) (1,2) (-1,-1)

Mb = (2,1) (-2,-1)

ua = (1,2/3)

ub = (0,0)

Ma = (1,1) (1,2) (2,1)

Mb = (-1,-1) (-2,-1)

ua = (4/3, 4/3)

ub = (-3/2,-1) **Done.**

EM/Expectation Maximization Theory: K-means will converge, but might not converge on the global optimal result. (can get different results from different initialization values)

Distances used in K-means are object-centroid, rather than object-object as given in our definition of clustering.

**Proving equivalence between the two objectives of k-means and clustering:**

For each cluster K{

For each point in a given cluster{

||x-uk||22

}

}

The difference between a centroid and the sum of its contributing vectors will always be 0.

**Evaluating your K-means:**

Purity: Evaluate the model’s ability to identify groupings in gold standard data.

Purity of a cluster = percentage of the cluster made up by points from the most common true cluster.

purity(Sk) = 1/nk \* max(nkj)

Rand index: Calculated with a sort of truth table/confusion matrix

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | A | B |
|  | C | D |

RI = (A+D)/(A+B+C+D)

Vanilla k-means creates ball-shaped clusters.

Mahalanobis distance (u,E) can be used with a covariance matrix to create a better distance metric for specific use cases. Each cluster can be given a different covariance to get an appropriate mahalanobis distance metric for it.

Mahalanobis distance: d(x,u) = sqrt( (x-u)T E-1(x-u) )

k-means → Gausian Mixture Model (GMM)

**Hierarchical Clustering:** Clustering will form a tree structure called a dendrogram, which can be cut according to how many clusters you would like to have.

Building the dendrogram by linking most similar sample pairs:

-Single link: Shortest distance

-Complete link: Longest similarity: sim(ci,cj) = min(for x in ci, y in cj) sim(x,y)

-

Agglomerative:

Divisive:

**Convolutional Neural Networks:**

**Overview:**

Main step = convolution = Convolve a filter over the surface of the input layer to create an activation map which will have dimensions of x \* y \* 1. You can have more than one set of activation map/filter layers. Output image will be a new image created by stacking all activation maps created.

You need a non-linear transformation between layers, your model will still be linear no matter how deep you make it.

Hyperparameters needed when creating a CNN model: the width and height of your filter, the number of filters per layer (each filter in a layer corresponds to a different neuron in that layer), and the stride (step size of filter, how far the filter moves)

Stride: if image width = N and filter width = F, (N-F)/Stride +1 must be a whole number. If not, pick a different stride or “zero pad” the border of the image to make it fit.

**Example:** If you have an input of size 32\*32\*3, and run it through 10 5\*5 filters with stride 1, and padding of 2, what is the output volume size?

= H’ \* W’ \* C’

= H’ \* W’ \* 10

H’ = W’ = ((32+ (2\*2)) -5)/1 + 1 = 32

= 32\*32\*10

**Example:** If you have a 32\*32\*3 input volume, and 10 5\*5 filters with stride=1 and pad=2, how many parameters are in this layer?

(5\*5\*3+1)\*10=760

**Pooling layer:** Makes the representations smaller and more manageable. (downsampling), and operates over each activation map independently.

**Training Neural Networks Part 1:**

Neurons in a NN have inputs from axons from previous neurons, with values ranging from w0 – wi respectively.

**Activation functions:**

Why don’t we use sine as an activation function?: it has 0 gradient regions

Why not sigmoig?: It has a saturation problem, despite not having any 0 gradient regions.

Rectified Linear Unit/ReLU: equation is max(0,x), so and negative x will be 0 and positive will be linear.

Leaky ReLU: equation is max(.1x,x)

tanh(x) is used in RNN/LSTM models. Looks roughly sigmoidal.

**Preprocessing Data:**

Normalization:

-Zero-center the data points. (minimizes the effect of the bias term in the model)

-Scale the data so that each dimension’s standard deviation is the same.

Decorrelation:

-Make the data distribution have slope = 0

Whitening data:

-Make the each dimension of the data have the same standard deviation, end result should look like a circle/ball.

In deep learning, we usually only zero-center the data. In image processing, that means subtracting the mean image, and the per-channel mean.

*The variance of activation should be close to the same across every layer.*

**Weight initialization:**

Before training, all weights in a model should have an initial value. Cannot all be 0.

Batch normalization: Usually inserted after Fully Connected or Convolutional layers, and before ?linearization?

-Compute empirical mean and variance independently for each dimension.

**Neural Network Training Part 2:**

Vesterov momentum:

**First order optimization:**

Optimizing the gradient descent implementation you use.

**Second order optimization:**

Using gradient **and Hessian** to form quadratic approximations, then stepping to the minimum of the approximation.

Second order Taylor expansion:

Stop training a model when accuracy on the validation set starts to decrease.

**Model ensembles:** Train multiple independent models, then at test time average their results (Take average of predicted probability distributions, then choose argmax). Improves performance by 2%, despite taking more total training time.

**Regularization dropout:** In each forward pass, randomly set some neurons to zero. The probability of a neuron dropping is a hyperparameter; setting it to .5 is common.

**Regularization Data Augmentation:** Applying transformations to data to increase the quantity of training data (like reversing, blurring, cropping an image).

**Transfer Learning:** Import the basic model with early training results frozen, then redo final training with your usecase-specific training data.

**Outlier Detection: Preliminaries and Techniques**

An outlier is a data point that deviates significantly from the rest of the dataset, almost like it was generated by a different mechanism.

-Outlier detection is related to novelty detection, but is not the same.

-It can be important to justify why a point is an outlier.

**Supervised learning for outlier detection:**

Given a dataset {(xi,yi)}, 1<=i<=N, train a classifier f : X → {0,1}.

For a new instance x, it is classified as an outlier if f(x) = 1, or a normal instance is f(x) = 0.

-Requires a full pre labeled dataset

Potential issues:

-Data imbalance:

-Evaluation gap: Classification maximizes prediction…

**Unsupervised Learning for OD**:

Implicit assumption: The normal data points are clustered

**Semi-supervised learning for OD:**

Given a dataset {xi} and {yi}, 1<=i<=N, j E *J*, |*J*| << N, train a detector f: *X* → {0,1}.

**Outlier detection methods:**

**Idea:** Learn a probability density function p(x), denoting the probability that data point x is generated by the normal generation method.

Parametric methods: Assume p follows certain prior distribution such as gaussian with parameters (theta) to be estimated. Then you can build a distribution representative of that, and measure how many standard deviations from the mean any new data point is and use that to predict whether it is an outlier.

Non-parametric methods: Do not assume prior distribution. Example: Outlier detection using a histogram (sorting data points into bins, in which case choosing a bin size is important so you should try multiple different bin sizes)

-Kernel density: Treat an observed object as an indicator of high probability density in the surrounding region.

-Always true: K(u) = K(-k)

h is used as a bandwidth parameter.

**10-6-22**

Kernel Density Estimation: Adding up all kernel results where each kernel is around a center point.

Used to estimate similarity between two data points.

Kernel functions have to be symmetric functions whose probabilities sum to 1.

h = bandwidth. If bandwidth is large, the kernel estimation will be wide, and if the bandwidth is small, the kernel estimation will be narrow.

As h approaches 0, x/h approaches infinity?

Smaller h corresponds to a smaller bin size in histograms, and a larger h corresponds to a larger bin size in histograms.

**Proximity based outlier detection methods:**

If an instance is far away from other points, it can be considered an outlier. Can be measured using distance based or density based proximity.

Example of a idstance-based outliers system: DB(r,pi):

x is an outlier if |{x’ where dist(x’,x) <= r}|/|D| <= pi

|{x’}| = the number of points within a radius of r from a center point x which we want to know if it’s an outlier.

**Isolation Forests/iForests**

Separating instances from others. If an instance can easily be separated, it is more likely to be an outlier.

Before each tree construction, it will subsample instances.

To build a tree:

1. Pick an attribute to split on

2. Pick a random value to split the data into subsets along

3. Recursively split subsets into more subsets along more values until the number of data points in each subset is <= 1, or you have to split along a new attribute.

(When picking a value to split along, check the max/min values of the data set so you don’t pick a value that doesn’t split. **)**

Normalize the forest with a value C:

C = 2ln(n-1) - 2(n-1)/n

Once you have constructed a forest of trees, you can find the average path length to each node. If a node has a long average path length; it is more likely to be an outlier.

**Midterm review:**

HW 2 problem 6:

-At each level of building a dendrogram, you can only connect one pair of clusters.

-Use the largest distance between any pair of points from different clusters as the distance between the two clusters.

General review:

-Single link measurement between clusters: Use distance between closest pair of points, one from each cluster.

-Complete link between clusters: Use distance between farthest pair of points, one from each cluster.

**Reminder Systems: Preliminaries**

Types of recommendations: Top 10, most popular, recent uploads, etc.

Example model:

Build a matrix, where every column is a rated product (such as a movie), and each row is a user. This is known as a **utility matrix.** For any movie the user has watched/rated, there is a score on a scale of 0.0-5.0 designating how highly they rated the movie.

-Since each user isn’t going to have watched and rated every movie in the model, we need to be able to extrapolate filled in ratings to fill the gaps of unfilled ratings. One method to do this is using **implicit ratings**, which means we learn/guess ratings from user actions. So if a user views an item or purchases it, that carries a positive connotation. Meanwhile if the user returns the item, that has a negative connotation. This method is efficient and convenient for everyone involved, but not very accurate.

**Content-based recommendations:**

-Idea: Recommend items to a user that are similar to previous items {i} rated highly by the user u.

Pros: can recommend to unique users, recommend new/unpopular items, able to provide explanations for recommendations.

Cons: Picking appropriate features can be hard, recommending for new users can be hard, and it doesn’t recommend items outside of a user’s existing content profile (can create an echo chamber).

-These systems rely on user profiles and item profiles.

-We can build a user profile with:

x= (Sum(ru,ii))/(Sum(ru,i))

Given a user profile x and item profile I, estimate: f(x,i) = similarity(x,i)

**Collaborative filtering:** Given a user u, find a set of N similar users and estimate u’s ratings based on ratings of the users in N.

You can find the similarity between two users using cosine similarity on their user profile vectors. sim(u1, u2) = cos(u1, u2) = (u11\*u21 + … + u1n\*u2n)/(sqrt(u11...+...+u1n)\*sqrt(u21...+...u2n))

**Dimensionality Reduction: SVD/CUR**

Dimensionality issues: The more dimensions/features your data has, the harder it is to manipulate it. This causes problems that hurt models like KNN, or any model that uses distance to measure data relations.

“The curse of dimensionality”

More points: less likely to overfit in a linear model. Fewer points: more likely to overfit.

Dimensionality reduction: If data is represented in 3d but can be represented across the main axes of correlation without singificant loss of information, you can project it to a 2d plane along those axes.

We determine how much dimensionality of a matrix can be broken down by looking at the rank of the matrix. The rank of a matrix is the number of linearly independent rows the matrix has. So:

[1 2 1

-2 -3 1

3 5 0]

Has rank=2, since the first row can be recreated by combining the second and third rows. Rank will always be <= the number of features/variables.

**SVD/Singular Value Decomposition:**

A = input matrix, U = left singular vectors, E thing = Singular values (should be an identity matrix), V=Right singular vectors.

A[m\*n] = U[m\*r] E[r\*r] (V[n\*r])T

r = concepts = rank of matrix A

**Local interpretable Model-agnostic Explanations (LIME):**

-Can be used on any machine learning model after it has been trained.

-For any model f, we can locally approximate3e it with a simple, interpretable model g

**Perturbation-based Interpretation:**

Explains an individual prediction and the effect of a specific feature value on the prediction.

Example: occlusion maps.

**Saliency map:**

**Gradients as sensitivity maps:**

For a specific classification model S with input x, the classification result is: class(x) = argmax for c in C of Sc(x)

Integrated gradient: = (x-x0) Sum for each x in path( (dSc(xt)) /dx)\*(1/T)

path = {x10,x11,…,xT)

**Smooth gradient/Smoothgrad:**

Removing noise by adding noise.

**Post-hoc model evaluation:**

Activation maximization:

x\* = argmax fl(x) – R(x)

**Shapely Value:** Takes into account the relational impact of different variables working together, like teammates.

**Text mining and embedding:**

Precision: Fraction of retrieved docs that are relevant to the user’s information needed. High precision = only returns relevant data

Recall: Fraction of relevant docs in collection that are retrieved. High recall: Returns all relevant data, even if it includes some irrelevant data.

Term-document incidence matrix: matrix with each column being a document, each row being a term, and each cell being 0 or 1 indicating whether that document contains that term.

Queries can be vectors of bits, and you can combine query conditions using bit-logic such as AND/NOT. To produce a new combined query vector. (110100 AND 110111 = 100100)

The Inverted Index:

Each document gets a docId, each term gets an array of docIds of all the documents containing it.

AND searches are done by merging the two arrays of docIds and outputting a list of all the ones the terms they both contain.

You can optimize iterating through these arrays by adding hyperlinks that skip some elements in the linkedlist array. You can also optimize by changing the order of the arrays and which pairs you process first. Start with the smallest two sets, then go up in size until all have been processed.

Biword indexes:

Proximity queris: Find where a word occurs within k words if another word.

**Text mining/embedding: Vector Space Model (VSM)**

Ranked retrieval: Instead of boolean query results whether a document should be returned or not, results are presented as a list of ranked documents.

Scoring documents:

Jaccard index: measures overlap between two sets. Index = |A intersect B| / |A U B|

Term frequency:

**TF-IDF (Term frequency-Inverse Document Frequency):**

**dft,d** = log10(1+countt,d), if countt,d > 0 //Term frequency score

**IDF**t = log10(N/dft) //Inverse document frequency score, where N=number of documents.

**wt,d** = log10(1+tft,d) \* log10(N/dft) //Term score

**Score(q,d)** = Sumfor t in q intersect d(tf.idft,d) //Score of a document for a given query is the sum of the scores of all search terms contained in the document

Instead of using similarity(q,d) use similarity(q,d). Using the normalized d will save computational costs by paying the cost in the offline cosine function rather than online?

**Information retrieval-evaluation:**

Precision@K (P@K): Set a rank threshold K, compute % relevant in top K, ignore documents ranked lower than K.

Recall@K (R@K):

Average Precision: average of P@K

MAP = Average Precision across multiple queries.

Precision-recall curve: Graph precision on the y-axis, recall on the x-axis. F1=2\*(P\*R)/P(+R)

DCG: total gain accumulated at a particular rank p:

DCGp = rel1 + Sumfrom i=2 to i=p (reli/log2i )

rel = relevance score?

**Word embedding:**

word2vec: for each position from 1 – T, predict the context words within a window of fixed size m given the center word.

-Given a word w, its context is the set of its nearby words (within a fixed-size window)

-Negative sampling: intended to optimize computation cost for word2vec by reducing the vocabulary V into a subset N, built using random sampling from V.

-word2vec can be implemented using a 2 layer neural network. The first layer contains a weight matrix of dimensions V\*D (Vocabulary \* Documents) for the center embedding, and the second layer contains a weight matrix of identical dimensions for the context embeddings.

u=context word

v=center word

vc = (stepsize \* result of gradient of loss function) + vc

**Graph/network data**

Adjacency matrix: symmetric matrix where each cell is 1 or 0, depending on if the nodes represented by the intersecting column/row connect in the graph. Nodes do not connect to themselves.

Transposed Adjacency Matrix: Adjacency matrix, but nonsymmetric since a 1 is filled in only if the row node points to the column node.

Degree matrix: shaped like an identity matrix, but with the sum of all values in each row in the corresponding cell.

**Node2vec:** Center node is group leaders, search window is necessary node connection hops to reach a node, graph is the sentence?

DeepWalk: Do a gazillion walk starting from random nodes and going through a random path. Then take the string of all nodes traversed and use them as a sentence and put them through word2vec. (Uses a user-predefined length of sentence to cut of walks)

**CNN review:**

H\*W\*C\*I

Can only apply convolution on spatial dimensions (H and W)

**Applications of Machine Learning in Graphs/Networks:**

-Node classification

-Link prediction

-Graph classification

**CNN on Graphs:**

g\_conv(H,A) = AHW

A subset of Rn\*n = adjacency matrix. n = number of nodes.

H subset of Rn\*c = feature matrix. c = for each node how many features does that node have? C = length of that node’s feature vector.

W subset of Rn\*n = weights matrix. d = predefined by user as a latent dimension, usually smaller than c.

W = trainable parameter Rc\*d. c\*d = latent dimension.

For multilayer CNNs, use the output of the first layer (it will be of dimensions n\*d) as the input for the new layer to build a new weight matrix W.

**Time complexity = (n2c)+(nc)+d**

**Optimization techniques:**

GraphSage:

-Node batch

-Don’t collect samples from all nodes, only use partial propagation.

In most cases, A is very sparse, so we do not store the full matrix.

**Eigenvector centrality (measuring importance of a node by its centrality?)**

Degree Centrality: If a node has many links to important nodes, the node is more important. In directed graphs, centrality only passes over outgoing edges; so a node’s centrality is only increased by connections that point to it, not from it.

Node centrality can’t be negative, only 0 or positive.

Perron-Frobenius theorem: A real square matrix with positive entries has a unique largest real eigenvalue and that the corresponding eigenvector can be chosen to have strictly positive components.

**PageRank**: Created by co-founder of Google and helped them become the goto search engine. Pages are ranked as node based on their links to and from hub and authority nodes and those nodes’ ratings.

Hub pages: like Wikipedia. Connect lots of other pages.

Authority: Linked by good hubs.

Hub score of a node = h()

Authority score of a node = a()

**HITS:**

**Convergence theory:**

Hub pages: like Wikipedia. Connect lots of other pages.

Authority: Linked by good hubs.

Hub score of a node = h()

Authority score of a node = a()

**Katz centrality:** Give each node a small amount of centrality to start, that centrality “currency” is passed around by dividing it by the number of nodes it has outbound connections to, and giving a piece to each.

**Markov Chain:** Each markov chain consists of n states, plus an n\*n transition probability matrix P.

-At each step, we are in one of the states.

For 1<= I,j <=n, the matrix entry Pi,j is the probability of j being the next state, given we are currently in state i.

Markov chains are abstractions of random walks.

**Final exam review: Take a double-sided cheat sheet. December 9th, 3:30-5:00 where we have our Monday class.**

Covers sections 4-8

**Outlier detection**: know fundamental methods

-Doesn’t cover One-class SVM or IOF.

**Recommender systems:** High level idea questions, not math questions. Know which model to use for situations and why.

**Interpretation:** Understand the SHAP algorithm as well as the high level idea of what interpretation is.

**Adversarial learning:** What is the objective function (cost function) for ?an attacker?

**FOCUS ON SECTIONS 7 AND 8, NLP AND GRAPHS**

**NLP:** Not asked to compute numbers, know the high level ideas about preprocessing, word embedding, TF-IDF, understand vectors in context and measuring similarity between documents. Understand how to evaluate a search engine (Binary precision, Precision@K, etc.)

**Word embedding:**

Given a cost function, understand it.

**Graphs:**

Graph CNN: Know the formula for graph convolution as well as the intuition behind the formula.

PageRanks: Understand page matrix, identity matrix, ?adjacency matrix?, and degree.

Understand time complexity for the unoptimized pagerank, as well as real life complexity after optimizations.

Final day presenters: Mushroom classification.

Pre-processing: determine features and categories for categorical variables.

-Rows with ? Categorical value (unlabeled) were removed from the table they aren’t labeled for. (Could have used it as an “other” category, or could have classified each based on their neighbors)

-Convert categorical data to numerical data (label encoding)

-split dataset into k-equal parts for training and testing complete dataset

Model implementation: Classified data points using k-neighbors, logistic regression classifier, decision tree classifier, Multi-Layer Perceptron Classifier, Support Vector Classifier, Random Forest Classifier.

Result metrics used:

Precision: What percentage of predicted edible mushrooms are actually edible?

Recall:

Accuracy:

F1-score:

Support:

Macro avg = unweighted mean of metrics

Weighted avg = Weighted mean of metric avgs.

Conclusion: All models worked well for their dataset.