Contents

1 Intro to ML

Problem is to find parts of data that are meaningful. Machine learning is the automatic extraction of valid knowledge from arbitrary sets of data. It focuses more on the maths and theory behind the models, and ignores the time complexity of the implementation.

Tasks in ml

- Classification
- Clustering
- Regression
- Probability estimation (model the p-distr of a random variable)
- Sequence discovery (predict time series)
- Association rule mining (find associations in dataset without labels)
- Model fitting (fitting a function model)

Learning in ML is fitting a function $f(x)$ to training data that maps all inputs to some output. The goal of learning is to generalize, that is to produce outputs to inputs that the model has never seen before.

You dont need to learn if: input size is finite, so there's no need to generalize; if there are no rules that relates inputs to outputs, so we can only memorize.

IRL, memorization is not applicable since the input space is continuous with infinite possible inputs.

2 ML Frameworks

Terminology of ml

- Inputs are called instances, exemplars, observations. These are independent sampled data from the world
- Each instance consists of the attributes/features and some concepts/labels. The features are measurable aspect of each instance, and labels are desired values we want to know about the instance

Goal of ml is to learn a function that maps attributes to concepts. It must be general, in that it will return the concept for every set of attributes, even for ones that the model has not seen before.

Example concepts are

- Labels
- Numbers
- Clusters
- Probabilities
- Order of events

- Sequence of events
- More complex models

Supervised vs unsupervised

- A supervised method has instances that contain the labels, and it learns the association between attributes and labels
- An unsupervised method only has unlabeled instances, it learns only the structure of the attributes like groups, correlation, sequences. It can discover latent variables that explain the patterns in instances. It can reduce dimensions of the dataset.

In supervised

- Learn mapping from attributes to concepts
- Training will have the model see examples of attribute concept pairs, it will attempt to learn the function
- Test will have the model predict concepts on new unknown sets of attributes
- Eval will compare predictions on test to their true labels.

In unsupervised

- For autocomplete, attributes could be a set of words, and concepts are probability distribution of the next word, but the instances are just sentences.
- Training has model see many examples of attributes
- Model will learn a function that produce some useful concepts like a p-dist
- Test will run the model on new sets of unknown attributes and predict concepts
- Eval depends, if predicting p-dist, can check if future data fits the p-dist predicted

Association learning are

- Patterns between attributes or attribute and concepts
- No right answer for association
- Eval is hard
- Can be under supervised or unsupervised learning

Marr's levels of analysis is a framework for underlying any algorithm system

- Computation level, what is the goal of the system
- Algorithmic level, what algorithm/ds is used
- Implementation level, the physical implementation

Applying Marr's level of analysis to ml

- Computation level, what structure does the model expect to see, what rules are there in the data
- Algorithmic level, model fitting, minimizing loss function
- Implementation level, minimize loss function in finite time, gradient descent.

When a model fails, we use the level of analysis to debug where it went wrong.

Note

- Different assumptions about the data will result in different models and results even with the same goal
- Fewer assumptions does not mean a better model. Models that make simplifying assumptions can result in better models.

Places where our model can go wrong

- Computation level, is the model assuming the right structure
- Algorithmic level, is the loss function correct
- Implementation level, did the gradient descent work

3 Probability and Entropy

3.1 Variable Types

Attributes can have many data types

- Nominal (categorical). These are discrete types or categories with no natural ordering. Boolean types are special nominal variables with only two categories.
- Ordinal. Discrete values with a natural ordering. These are not real numeric value since maths relations on numbers don't make sense (no distance, add/subtraction). Threshold on these variables are meaningful
- Continuous (numerical). Real valued number with a zero point and no explicit bound. Intervals are consistent, maths on them make sense. Ints are treated as numerical since they likely represent some continuous real world data.

Different variable types implies different structures in data that should be handled differently, and some models can only work with nominal or continuous data.

3.2 Probability

Probability is used because: the problem is too complex, or there are uncertainties and ambiguities in the dataset. Quickstart

- Probability of event, $P(x)$
- Joint probability, $P(x, y) = P(x \cap y)$
- Conditional probability, $P(x|y) = P(x \cap y)/P(y)$

- Formulas: sum rule, product rule, bayes theorem, chain rule

$$P(x) = \sum_y P(x \cap y)$$
$$P(x, y) = P(x \cap y) = P(x|y)P(y)$$
$$P(y|x) = \frac{P(x|y)P(y)}{p(x)}$$
$$P(\bigcap_i x_i) = P(x_1)P(x_2|x_1)P(x_3|x_2 \cap x_1)\dots$$

- For bayes theorem, the prior probability is $P(x)$, the posterior probability given y is $P(x|y)$.
- Independence between x and y means

$$P(x|y) = P(x)$$
$$P(y|x) = P(y)$$
$$P(x, y) = P(x)P(y)$$

- Conditional independence is when x, y are independent conditioning on z is

$$P(x, y|z) = P(x|z)P(y|z)$$

A probability distribution is a function mapping outcomes of random variables to their probabilities. An empirical probability distribution comes from a sample, a theoretical probability distribution is based on mathematical theory on the population.

Integral over p-dist should be one. Example p-dists

- Binomial distribution, sum of independent bernoulli trials, pmf is

$$P(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

- Multinominal distribution. A series of independent trials where each trial has multiple outcomes. Pmf

$$P(x_1, x_2, \dots, x_n) = \left(\sum_i x_i\right)! \prod_i \frac{p_i^{x_i}}{x_i!}$$

- Normal distribution, represent the distribution of error/noise. Has a mean and standard deviation

Probability model, maths representation of a random event

- The sample space of possible outcomes
- The event space, subset of sample space
- Probability function mapping events to probabilities

3.3 Entropy

Entropy measures information or uncertainty.

- It represents how much information there is to learn in a problem
- Used to quantify how much the model learned

Shannon entropy measures the information required to predict a random variable (or how uncertain the random variable is). It is measured in bits.

- It is defined by
$$H(X) = -\sum_i P(X = x_i) \log_2 P(X = x_i)$$
- Entropy depends on the number of possible states, and the likelihoods of those states. Low entropy implies outcome of rv are predictable, high entropy implies outcomes are unpredictable
- Range of entropy is $[0, \log n]$ where n is the number of outcomes. 0 entropy implies only one outcome, $\log n$ entropy implies uniform distribution

We can use entropy to compress text, namely, we assign longer codes to less frequent characters and shorter codes to more frequent characters. The theoretically minimum number of bits per character is the entropy of the text. Encoding is to pack maximum information into limited bandwidth, and entropy represents the limit of the encoding/compression.

There are no rules to desire for max entropy or min entropy, since it is just an attribute for a rv. But there are exceptions in ML. Suppose we want to compress two attributes by letting the second attribute be determined based on a threshold of the first, we can compute for each threshold the entropy of the true labels that gets assigned to each side of the threshold (and pick the threshold that minimizes average entropy).

4 Naive Bayes Classifier

Concept or probabilistic learning is the method to identify a class concept from limited data, even with few examples, noise features, diverse dataset, and ambiguity about important feature.

Probability models provide a framework for modelling uncertain systems, where we generalize rule from limited observations. It uses the laws of probability. Goal is classification, so supervised. The question is: given an instance T, which class c is the most likely

$$\hat{c} = \arg\max_{c \in C} P(c|T)$$

this is the maximum likelihood classifier.

A simple learner would be to pick the most frequent T given in the training data with label c. But this is not possible since it requires a massive amount of data (need to see every possible combination of attributes, which is $O(Ck^m)$ where k possible values and m attributes).

4.1 Naive Bayes

Naive bayes assume that all attributes are conditionally independent on the class label, namely, $P(x_1, x_2|c) = P(x_1|c)P(x_2|c)$. This DOES NOT imply that the attributes are independent, only

conditionally independent. Naive bayes use bayes rule and conditional independence to estimate $P(c|T)$, suppose $T = (x_1, x_2)$

$$\begin{aligned} P(c|x_1, x_2) &= \frac{P(x_1, x_2|c)P(c)}{P(x_1, x_2)} \\ &\propto P(x_1, x_2|c)P(c) \\ &\propto P(x_1|c)P(x_2|c)P(c) \end{aligned}$$

where we ignore the bottom term since they are equal for all classes. Therefore, the model is

$$\begin{aligned} \arg\max_{c \in C} P(c|t) &= \arg\max_{c \in C} P(x_1|c)P(x_2|c)P(c) \\ &= \arg\max_{c \in C} P(c) \prod_i P(x_i|c) \end{aligned}$$

To get the likelihoods $P(x_i|c)$, we can use the empirical frequency in the training set. We can also get the prior $P(c)$ from the training data.

Assumptions

- the naive bayes treats each class label to generate instances with attributes, where each attribute is independent within each class label (we can then get the conditional probability of the class label via bayes theorem). This is often untrue in the real world, but the classifier works well in practice.
- it also uses the distribution of class labels in the training dataset as bayesian priors, with assumes that the distribution of class labels matches that with the real dataset (not the case if we systematically sampled our dataset)
- the most basic form assumes that the attributes are discrete

4.2 Zero Likelihoods, Probabilistic Smoothing

If any likelihoods are zero, the relative frequency of that class label will be zero. This can happen a lot in small datasets (since we need to see every possible pairing of attributes with class), leading to ties where all conditional likelihoods are zero.

We shouldn't erase zeros since they denote information. Instead, we can replace all zero likelihood with a very small ε, and a relative frequency is higher if: it has the same number of ε but higher coef, or it has fewer ε. Because $\varepsilon < 1/N$, we don't need to scale the other frequencies.

This will tend to penalize smaller class labels if it is missing data.

Laplace smoothing will increase all counts by α. This will change the probabilities to

$$P = \frac{x_i + \alpha}{n + d\alpha}$$

where d is the number of distinct values of the attribute. The specialized case where $\alpha = 1$ is called add-one-smoothing. This will change the probs a lot when there are few instances/samples, but the changes are smaller with larger datasets.

Laplace smoothing will tend to overestimate the likelihoods of unlikely event. ε or lower α will tend to underestimate those events.

Other smoothing methods

- Add-k smoothing
- Good turning estimation, use observed counts of other attributes to estimate the missing attributes
- Regression

4.3 Missing values

If an instance is missing some attribute, we can simply ignore the missing attributes in inference (discarding one or more multiplication) and compute the likelihoods from the non-missing attributes.

We can also ignore the missing attributes in instances in the training dataset by computing the probabilities without those attributes.

Summarizing Naive Bayes

- It usually works, because: we don't need a perfect estimate of $P(c|T)$; ignoring correlations will increase the conditional probabilities, but will not change the ranks.
- Because it is ranked based, it is robust to small errors in $P(c|T)$
- Simple to build, fast
- Computations scale well to high dimensional datasets
- Explainable, can understand why the model makes the decision it makes
- Inaccurate when there are many missing $P(x|c)$ values
- Conditional independence assumption is problematic with complex datasets

4.4 Continuous and Discrete

Naive bayes assumes nominal data. If the attribute is continuous, we will need to bin the values and convert them into ordinal data.

ML Algorithms usually assume that attributes have a particular data type

- Nominal: Naive bayes, decision trees
- Numeric: SVM, Perceptron

If the attributes doesn't fit the algorithm

- Discard the attributes
- Convert the attributes
- Change the model

To convert from nominal to numeric.

- use one-hot-encoding: attributes with m possible values is converted to a length m boolean array. This is the best way to represented nominal data, but it increases the dimensionality.
- Assign separate numbers to each distinct value. This is bad because it implies ordering between nominal attributes and distances are wrong

To convert from numeric to nominal.

- Discretization translates continuous to nominal.
- Usually two steps: decide how many nominal values, and decide where to place the boundaries

Different discretization methods

- Equal width assigns the same width for each bin. Find min and max, each width is $(max - min)/n$.
- Equal frequency assigns the same number of data points in the same bin. Sort values, find breakpoints that produce n bins with equal numbers
- Both equal width and equal frequency have arbitrary group boundaries. Equal width is sensitive to outliers, equal frequency is sensitive to sample bias. The number n is user selected
- Use equal width if the distribution is uniform. Use equal frequency if the distribution is skewed.
- Clustering to discover natural breakpoints.
- Clustering is more complicated, and if the data doesn't have natural groups, k-means is just equal width. K-means also sensitive to outliers
- Supervised discretization, where we group values into class contiguous intervals. Sort values, identify breakpoints in label membership. Reposition breakpoints if numeric values are equal. And set the breakpoints midway between the neighboring values
- This helps with classification, but produces a lot of groups, and can overfit the training set.

Discretization means throwing out information, which can help to simplify data and ease model learning, but can also lose details that would've been helpful.

4.5 Naive Bayes with Continous values

To use continuous values with naive bayes, we can discretize the attributes first and treat it as a nominal value. We can also use the probability density function.

We can model the likelihood $P(x_i|c)$ by a normal distribution.

- Split the dataset instances by their labels
- Estimate the sample mean and sample standard deviation per label per attribute, and assume that $X_i|C \sim N$. (This is done on the training set)
- Substitute $P(x_i|c)$ with the pdf of $f_{X_i|C}(x_i, c)$ during inference (on the testing set).

We can mix the pdf likelihood with the probabilistic likelihood in the naive bayes

If there is only one attribute, we can look at the superimposed pdfs and see where the model will classify the class labels given the range of the attribute (only if equal frequency). We can also estimate the probability that the model will misclassify by looking at the area under the curve of overlaps.

Different continuous models that we can use

- Multivariate (where all the dimensions are in the same attribute)
- Binomial (where all the dimensions are in the same attribute), estimate the parameters using maximum likelihood (means)
- Multinominal, natural numbers with pmf

  $$P(a_i = n) \propto P(a_i = 1)^n / n!$$

  and

  $$P(a_i = n_i, \dots) = n! \prod_i P(a_i = 1)^{n_i} / n_i!$$

  we can estimate the parameters using maximum likelihood (mean) (all the counts are treated as one attribute)
- Normal
- Kernel density estimation, an arbitrary distribution from the sample data points

Note that when modeling using multinominal, we treat the collection of counts as values in a single attribute.

5 Evaluating Models

To measure the goodness of a classification model, we want to know if it generalizes to other datasets or not. So we train on the training dataset, test on the testing dataset (not seen before), and use an evaluation metric on the predictions on the test dataset.

In general, any statement we make about the model based on the test set will highly depend on the test set.

A simple metric is accuracy, the number of correctly identified instances divided by the total number of instances.

5.1 Holdout Split

Use a train test split, where classifiers train on training test and test on testing set. We need the test set so we can check for generalization on unknown data.

Random holdout will randomly partition data into training and testing set. Common split is 80-20. Leave-one-out leaves only one instance in the testing set. Trade-off between having enough data for training set and having a representative test set.

Repeated random subsampling will repeat random holdout multiple sets. The final evaluation is averaged over all iteration. Slower, but result is more reliable than one random holdout.

Cross-validation is a preferred alternative to repeated random subsampling. Data split into partitions, iterativel:, one partition is test set, the other partitions are training data. Evaluation is averaged across all iterations or stored as a list. The crucial thing is that each instance appears in the test set exactly once.

The number of folds in CV has a tradeoff: more folds means fewer test instances. We use $m = 10$ to mimic 90-10 holdout. Best choice if you have time is leave-one-out CV which maximizes training data.

Stratification (vertical sampling) means to split the training and testing set such that the class distribution or feature distributions are identical to the entire dataset. This helps to ensure that the evaluation is not biased towards the populated class.

The data can also be split into train val test. Validation set is a test set for the training data used to choose parameters (and check if the model converged to a good solution on the validation set). Otherwise, we may overfit on the testing set if we use the testing set to evaluate, or it may overfit on the training set if we use the training set to evaluate.

5.2 Theory

The inductive learning hypothesis says that any model which approximates the target function over a large training set will also generalize to unseen examples. Machine learner however suffer from inductive bias, that there are assumptions made about the training data and different assumptions will lead to different predictions.

5.3 Evaluation Metrics

Accuracy is not enough. Consider a two class problem where there are positive and negative classes, there are four possible classification results

- True positive
- False positive
- True negative
- False negative

A confusion matrix displays the counts of such four cases in the inference on a testing test.

Define the accuracy metrics

$$\begin{aligned} \text{accuracy} &= \frac{TP + TN}{ALL} \\ \text{error rate} &= 1 - \text{accuracy} \\ \text{error rate reduction} &= \frac{ER_0 - ER}{ER_0} \end{aligned}$$

Some kinds of errors are more important than other kinds of errors. Sometimes we don't want to miss any positives, sometimes we don't want to identify a negative as positive. Define

$$\begin{aligned} \text{precision} &= \frac{TP}{TP + FP} \\ \text{recall} &= \frac{TP}{TP + FN} \end{aligned}$$

where precision is the accuracy on all identified positive cases, while recall is the accuracy on all true positive cases. The trade off is

- Higher precision means model requires a lot of evidence to say positive
- Higher recall means the model doesn't require a lot of evidence to say positive

A metric that balances between precision and recall is the F-score, which is

$$F_\beta = \frac{(1+\beta^2)PR}{(\beta^2 P) + R}$$

where $\beta = 1$ for F_1 scores.

Sensitivity is another name for recall. Specificity is the proportion of true negatives over the true negatives plus the false positive (true negative over all negatives).

5.4 Multiclass Evaluation

A multiclass confusion matrix denotes the counts in all actual vs predicted class labels over a test set.

For more than one class, if there is a class of interest, true that class as positive and the rest as negatives (evaluate against one vs rest).

If all classes are relevant, define

- class accuracy is the proportion of predicted class instances over all instances with that class
- class precision/recall, treat that class as positive and the rest as negative
- macro-averaging, calculate the precision recall per class and take mean
- micro-averaging, same formula as precision and recall, but on numerator and denomernator sum over all TP etc for all clases.
- weighted averaging, same as macro-averaging but weight each precision recall by the proportion of instances with that class

5.5 Baselines

A baseline is a naive method that we expect a machine learning method to beat. Benchmark is an established rival technique that we are comparing with (current best algorithm). In practice, we use them interchangably.

Common baselines

- Random guessing, uniform or weighted by training class distribution
- Zero-R (zero rule), always guess the most common label
- If regression, guess mean. If object detection, guess the middle of the image

There are models that can't beat the baseline on accuracy, simply due to the fact that the class distribution is very biased. This doesn't mean that it is bad since precision and recall also exists.

In ML, there isn't a mathematical way or knowing which model generalizes the best. The only way to guarantee optimal performance on the test set is to know some priors of the test set (no free lunch theorem). We also can't know if a model is solving a problem correctly.

6 Decision Tree

One R (One Rule) baseline is a baseline for discrete class classification

- For each attribute, assign each level of that attribute to the most likely class, compute the error rate using this method
- Pick the attribute that has the lowest error rate and use that attribute to do inference

The One R model is a decision stump (splitting on one only attribute). A decision tree is a hierarchy of decision stumps and can split on all attributes.

ID3 is an algorithm to construct decision trees. At each call, we consider

- If all instances at node has the same labels (or some purity), done
- Otherwise, select an attribute to partition the node on.
- Branch out to nodes split by the attribute, call ID3 on those nodes.

Other stopping criteria are

- All attributes have been used (only if continuous trees since in discrete trees using a attribute twice doesn't do anything)
- The IG dropped below a threshold
- The purity/entropy is below a threshold

To classify a new instance, simply traverse down the tree to a leaf node, and pick the most frequency label in that node. If an attribute doesn't exist

- Pick a random child
- Pick the child node with the most child classes
- Random but weighted by the child node size
- Average across all going down all child nodes weighted by size

Decision trees can naturally be read as disjunctive descriptions, where a label is identified by ORing all paths to true cases leafs.

6.1 Splitting via Information Gain

Entropy measures the level of uncertainty.

When splitting, we want branched nodes to have minimal entropy to maximize purity.

Information gain is the reduction in entropy caused by knowing the attribute. To compute it, we calculate the entropy before splitting and the weighted average of entropy of the children after the split (mean information). If the entropy decreases, it is a good split (this will happen in general).

Mean information formula for splitting by attribute A with levels x_i

$$MI(A) = \sum_i P(A = x_i)H(A = x_i)$$

The information gain for splitting on the attribute A on node N is

$$IG(A|N) = H(N) - MI(A)$$

The splitting criterion using information gain picks the attribute that maximizes information gain.

Note that the information gain is always possible since the weighted average of the entropies on splitting is always less than or equal to the entropy before.

One issue with information gain is that it rewards attributes with more labels and more branches, since the subsets are likely more pure. This will result in overfitting.

6.2 Gain Ratio

Gain Ratio reduces the biases of information gain by dividing it against the split information.

$$GR = \frac{IG}{SI}$$

Split information is the entropy the child node sizes of a given split. It is higher if the split is more uniform and therefore reduces the gain ratio on highly branching splits

$$SI = -\sum_i X_i \log X_i$$

Gain ratio makes the decision tree less likely to overfit

6.3 ID3 analysis

ID3 is an inductive learning algorithm. It searches a space of hypothesis (all decision trees) for one that fits the training examples. It uses a hill climbing, greedy method to select the decision tree, with no backtracking.

We generally want the smallest tree due to Occam's Razor and generalizability. A shorter hypothesis that fits the data is less likely to overfit, while a longer hypothesis may just be a coincidence for the training set only.

Practical properties

- Pretty good
- Fast to train and classify
- Suspectible to noise and irrelevant features
- Need to deal with missing features

Variants

- Random trees, use a sample of the possible attributes to split at each node. Helps irrelevant features
- Random forests uses multiple random trees and a voting system
- C4.5 is an extension to continuous variable attributes. It can also prune branches to prevent overfitting

7 Instance Based Learning

Mostly about geometric style models.

Instance based learning stores labeled examples in memory, and it learns directly from the examples. This is in contrast to supervised learning where we find parameters of a function.

7.1 Distance and Similarity

A feature is an aspect, characteristic of an instance. A feature vector is an n-dimensional vector of features that represents that instance. Features can be of any datatype.

Similarity is a numeric measure of how alike two objects are, higher when they are more alike, often in $[0, 1]$. Dissimilarity is a numeric measure of how different two objects are, lower when two objects are more alike, minimum of zero, but likely no upper bound.

Typical similarity and dissimilarity metrics between two objects with attribute p and q, over the data types of nominal, ordinal, and numerical

$$d = \begin{cases} 0 & p = q \\ 1 & p \neq q \end{cases} s = 1 - d$$

$$d = \frac{|p - q|}{n - 1} \qquad s = 1 - d$$

$$d = |p - q| \qquad s = \frac{1}{1 + d}$$

A distance measure is a function that takes two points in space, and has the properties

- No negative distance
- Zero between the same point
- Associative
- Triangle Inequality

Dissimilarity measures don't have to be distance metrics. For instance, the set dissimilarity defined as

$$d(A, B) = |A - B|$$

since it is not associative.

Euclidean distance is

$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

Manhattan distance is

$$d(x, y) = \sum_i |x_i - y_i|$$

Cosine similarity ignores the length but measures the angle

$$s(a, b) = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2} \sqrt{\sum_i b_i^2}}$$

NOTE that the higher the cosine similarity the closer the vectors.

For nominal variables

- Convert categories to arbitrary numbers, but this is bad since it implies an ordering with different distances
- One hot encoding into an n-dimensional space so that distances between categories are identical. But this increases dimensionality
- With either option, we can use Euclidean distance or hamming distance (number of dimensions different) on the encoded vectors.

7.2 KNN

To classify an instance, find the most similar instances (neighbors) and use them to choose the test label. This can be for classification and regression where we take the majority vote or an average (or weighted).

We define the nearest neighbor by max similarity or min distance. KNN will consider the k nearest neighbors given a test instance in the entire training set.

A decision boundary outlines the boundary in feature space where the classification changes.

Design decision for KNN

- Data scaling
- Distance metric
- Choosing K
- Aggregation, voting
- Tie breaking

Since attributes may have different ranges and our distance metric depends on the scale of the features, we need to standardize or normalize the features.

- Min-max scale scales all between zero and one
- Standardization scales all so that it has zero mean and one sd.

To choose K

- Smaller K tends to lower performance due to noise, can overfit on training set. It creates decision boundaries over outliers
- Larger K drives the classifier to 0R baseline
- K depends on the density of points, generally trial and error over the training data to get K

Aggregation

- Majority class and weighted Knn, weighted KNN uses the class with the highest weight
- Equal weight implies majority class
- Inverse linear distance uses the normalized linear distance (against all distances in the k nearest neighbor) as a weight

- Inverse distance uses
$$w_i = \frac{1}{d_i + \epsilon}$$
where $\epsilon > 0$.

Breaking ties, use weights, use another neighbor, randomly choose one class, use the class with the highest prior. Alternatively, use an odd number for K.

Implementation

- Typical impl uses brute-force of distances against every training instances. This is linear, which can be good on small datasets, but it becomes infeasible with larger training set sizes

Properties of KNN

- A lazy learner, since it doesn't take any time to train, but it does take time to classify — the model is the dataset itself
- Compared to classical models where the models are generally smaller than the dataset, KNN will have equal sized models with its dataset. Note that modern models are about the same size as their datasets

Strength

- Simple, instance based, model free
- Flexible decision boundaries (non-linear)
- Incremental, can add extra data on the fly

Weaknesses

- Requires a good distance function
- Arb K vlue
- Lazy learner
- Prone to noise and curse of high dimensionality if the dataset has a lot of dimensions

8 SVM

Nearest prototype classification

- Parameteric variant of KNN, since it is now fitting a model with C centroids
- Define the centroid for a class to be the average value for each attribute over all instances with that class label
- Also need a distance metric
- Classify a new instance by finding the centroid closest to the new instance, and use its label.
- Benefit of only keeping the centroid values into the model
- Problems of only able to classify linearly separable classes, and sensitive to outliers or extreme points close to the other class centroids

Support Vector machine

- Goal is to find a straight line or hyperplane that separates the two classes
- For now, only consider perfectly linearly separable (two classes can theoretically be separated by a hyperplane) binary classification tasks

Define a hyperplane in D dimensions as a normal vector w and an intercept number b, namely, it passes all points x where

$$w^T x + b = 0$$

In 2d this is a line, in 3d this is a plane.

A linear classifier like svm has the classification function

$$f(x) = w^T x + b$$

The points on the normal vector side will have $f(x) > 0$, and the points on the other side have $f(x) < 0$.

There are infinitely many hyperplanes that can separate the classes. However, we want a hyperplane that maximizes the separation. Namely

- A point far from the line has high confidence; a point closer to the line has low confidence since it is sensitive to small changes in the decision boundary
- SVM finds an optimal solution that gets a decision boundary that maximizes the correctness and confidence of predictions in a training set, precisely, it maximizes the distances between the plane and difficult points around the decision boundary.
- Define the margin (margin width) as double the distance between the plane and the closest point. The width of this margin is therefore defined by a few points closest to the line called support vectors (there should be at least 2 support vectors)
- Note that we don't select the support vectors, and the support vectors depend on the hyperplane chosen

SVM training and classification

- Let one class as positive and one class as negative
- To train, find the best hyperplane w and b that completely separates the two classes while maximizing the margin against the support vectors. Naive method is to use hill climbing.
- Actual method makes this an optimization problem that is convex and can be solved using quadratic programming. This will net a global optimal.
- To do inference on a test instance t, find the sign of $f(t)$. Can also convert the magnitude of $f(t)$ into a prob that shows the confidence. We can also assign a question mark to instances within the margin since we've never seen a point in the margin

Comparison against KNN

- SVM uses the training data to learn the weight vectors and intercept, while discarding the training data
- KNN must remember all training data

Outliers

- Sometimes violating constraints and using a larger margin can net an overall better separation
- Soft margins allows us to ignore some support vectors when computing the margin and determining the constraints
- We define the penalty of allowing such violations using a hyperparameter C, larger C allows for more violations

Non-linearly separable classes (non-linear SVM)

- Can still make it linearly separable
- Transform data using a mapping function to a higher dimension, then apply a linear classifier to the new feature vectors and hope that the higher dimension space is linearly separable
- In practice, use a kernel trick that performs the dot product on the transformed space

Multiclass KVM

- Commonly convert it into multiple two-class problems
- One-versus-all, one KVM to separate one class from the rest. Choose the class (that is not a combination of other classes) which classifies the test data point with the greatest confidence/margin/distance
- One-versus-one, one KVM for each per of class, choose the class selected by the most KVMs
- Training time for all these individual KVM can be high
- The decision boundary will be a combination of hyperplanes

8.1 Maths of SVM

Quadratic optimization problem formulation. NOT EXMAINABLE.

9 Model Interpretation

Methods of interpreting models

- Error analysis, why a model has mis-classified an instance
- Model interpretability, why a model has classified the instance in the way it has

Steps of error analysis

- Identify difference classes of errors the model makes. Usually we look at the confusion matrix, namely high values in the off-diagonal entries. Typically, each cell in the confusion matrix forms their own error class, but it is also possible that different errors exist in the same cell or the same error exists among multiple cells.
- Hypothesis to what caused the errors, testing them against data
- Quantify whether it is a problem with the data quantity or sparsity/quantity, or something more fundamental
- Feeding hypothesis back into model to improve it

Try to use the model assumptions to guide the error analysis, look for instance that break the assumptions and gets misclassified.

Note that this technically can leak test data into the model, since we are doing error analysis on the test set and feeding it back into the training process.

Error analysis details

- Evaluate the model beyond accuracy
- What items are correctly/incorrectly classified
- What items are the most/least confident classified/misclassified
- Performance among classes

Confidence

- Model's internal score on how well an item fits into a class, often as a probability
- NOT the probability that the model is correct, but it may be correlated
- Different definition for different models, SVM is distance to hyperplane, Naive bayes is the posterior probability ratios. KNN, weighted distance to selected neighbor

Model interpretability

- Understanding why the model classified the instance like it did
- Hyperparameters, parameters which define and control the learning process
- Parameters, what is learned by a learner given a set of hyperparameters when applied to a particular training set. Used for inference
- A model classification can be interpreted relative to the parameters associated with a given test instance (how the test instance features interact against the parameter values)

KNN

- Hyperparameters, K, distance metric, weighting strategy
- Parameters, the entire training set, model is lazy and doesn't abstract the training instances
- Interpretation, the training instances that give rise to a classification (nearest neighbors), how training instances are distributed in feature space

Nearest prototype classifier

- Hyperparameters, distance metric, feature weighting
- Parameters, class centroid/prototype locations. Size $O(CF)$ where C is the number of classes, F is the number of features
- Interpretation, the distribution of prototypes in space, and distance to each prototype given a test instance

Naive Bayes

- Hyperparameters, smoothing, distribution model to model a feature
- Parameters, conditional probability matrix, class priors, $O(C + CFV)$, where FV is the number of feature-value pairs

- Interpretation, find the most positively relative weighted feature given a test instance, the conditional probability matrix, the confidence/posterior probabilities

Decision tree

- Hyperparameters, splitting method, stopping criterion (max depth)
- Parameters, the decision tree, $O(FV)$.
- Interpretation, the path taken through the decision tree given an instance

SVM

- Hyperparameters, C, kernel, multiclass methods
- Parameters, hyperplane weight vector and bias term, constraint entries. $O(CF)$ assuming one vs all SVM.
- Interpretation, the absolute value of weights against each non-zero feature, since higher weights indicates that it is more important

Hyperparameter tuning

- Understand the hyperparameter meanings
- Manual tuning, grid search
- Compare performance on validation set

10 Visualization

Visualizing data can help you to understand it, also helps to detect anomalies. Methods for visualization are

- Data distribution along a feature (histogram) or along multiple features (scatter plot)
- Check model decision boundary.

Dimensionality reduction

- If more than 3 attributes, can use dimensionality reduction to reduce feature space down
- Either removing features, or dimensionality reduction.
- All reduction methods are lossy and cannot capture everyone in dataset

PCA, find principal components (new features) that capture maximum variation while being linear combinations of features and each orthogonal to each other. Popular dimensionality reduction method.

11 Regression

Linear regression

- Use continuous features to predict continuous concepts
- Assume independent linear relationships between features attributes and concepts, with the gradient as a weights vector β

- The concept is the outcome variable, response, dependent. The features are predictors, attributes.
- We can visualize the model as a hyperplane mapping the x coordinates to the $\hat{y}$ predictions

Assumptions

- Linear functions captures changes in one variable that correlate linearly against changes in another
- Simple assumption, less descriptive than non-linear functions, but allows simpler mathematically strategies
- Makes sense for some variables for small changes

Training

- Training set has N examples of features concept pairs
- Goal is the find the optimal weights β through a linear algorithm so that the prediction $\hat{y}_i = \beta \cdot x_i$ on the training set is the closest to the true y_i.
- To find the best line, we consider the line that minimizes the total distance between all points and the line via some distance metric.
- The least squares method finds a set of weights that minimizes the sum of the squared errors, namely minimizing MSE

$$MSE = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$

  on the training set

11.1 Parameter optimization

In general, learning is

- Given an evaluation metric M, a dataset T, a learner L with parameters θ
- Optimization will maximize

$$M(\theta; L, T)$$

  where L and T are fixed by changing θ
- Typically M is an error metric, and we are finding parameters that minimize the error metric
- Learning is finding the parameters that optimizes some criterion M

Analytic solution

- Closed form, can be computed exactly
- Requires solving a system of equations — set of equations setting error's partial derivative against a parameter to be zero. There will be $D+1$ equations with $D+1$ unknowns, where D is the number of attributes.

- For linear regression

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i} (y_i - \theta \cdot x_i)^2 = (X^T X)^{-1} X^T y$$

- Challenges are that derivatives can be hard to calculate for complex models

MSE is a convex loss function, so a local minimum is also a global minimum. An analytic solution will also be a global optima.

Exhaustive search

- For each set of possible parameter values, calculate the error. Pick the set with the minimum error
- For N parameters with M unique values each, requires N^M evaluations
- Works well in scenarios with a small set of hyperparameters, each having small number of values. Examples: K in KNN, similarity measure, voting strategy
- If features are continuous, use grid search: find boundaries of the parameter, divide range into equal width samples, do exhaustive search on the equal width samples
- Grid search has a tradeoff between more samples creating better estimate but also more training time

Iterative approximation

- Initialize initial guess, θ^0
- Evaluate guess
- Check stopping criterion
- Update the guess
- Repeat from second step

Gradient descent

- Update the guess by

$$\theta^1 = \theta^0 - \alpha \nabla M(\theta^0; L, T)$$

  where the gradient is the gradient of the loss function against the parameters, $\alpha > 0$ is the learning rate.
- Since the gradient points to the steepest slope, this is a greedy approach that climbs down the hill
- α defines how big of a step to take. Smaller α can increase training time. Larger α might miss the minimum
- If the loss is convex, the iteration will find the global minimum. Otherwise, it might get stuck in a local minimum

We can use gradient descent to optimize linear regression weights, but this is not necessary since we have an analytical solution

$$\beta_k^1 = \beta_k^0 + \frac{2\alpha}{N} \sum_{i} x_{ik}(y_i - \hat{y}_i)$$

11.2 Evaluation

Score metrics for linear regression are usually distances between predictions and true values

- MSE
$$\frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$
- RMSE
$$\sqrt{MSE}$$
- Root relative squared error, relative to the baseline that is the mean
$$\sqrt{\frac{\sum_i (y - \hat{y}_i)^2}{\sum_i (y - \bar{y})^2}}$$
- Correlation coefficient, pearson's correlation
- In general, the relative rankings of models across metrics are stable, so it doesn't really matter which metric to use

Feature engineering

- Add polynomial features like $(1, x, x^2, \dots)$ and add weights to new features to fit a polynomial relationship

Other regression models

- SVR
- Regression decision tree
- KNN with regression

12 Logistic Regression

Logistic regression is a classification model. Note that however, regression and classifications are interchangable.

To translate regression tasks into classification tasks

- Discretize the continuous labels onto discrete labels each containing a subset of the continuous range
- Build a classification model predicting the discrete labels

To translate classification taks into regression tasks

- If binary, assign one label as 1 and the other as 0. Build a regression model on the numeric labels, the classified class is determined based on the distance between the prediction value against 1 and 0
- If multi-class, convert it into a set of individual regression tasks that creates an overall model.

- Uses the concept of a numeric membership for classes: the group of numbers that belongs/infers to a class. This is a hyperparameter.

Logistic regression uses regression to solve classification tasks. The idea follows naive bayes by selecting the class with the highest posterior probability $P(c|x)$. Instead, we model $P(c|x)$ directly. Consider a binary problem

- Define $P(c = Y|x)$ to be the probability that the instance x is Y
- If label is Y, $P(c = Y|x) = 1$. Otherwise, $P(c = Y|x) = 0$.
- Treat attributes as predictors, and $P(c = Y|x)$ as an continuous outcome to fit using regression.

Types of models

- If $P(c|x) = \beta x$, the output is unbounded and the meaning of those probability doesn't make sense
- If $\log P(c|x) = \beta x$, the curve is unbalanced: values close to zero are fine with low sensitivity, values close to 1 becomes corse with high gradient. Also unbounded
- Logistic regression fits a linear plane to the logit of the posterior probability, $\text{logit}(P) = \log \frac{P}{1-P} = \beta x$. Rearranging we get

  $$P(c|x) = \frac{1}{1 + e^{-\beta x}}$$

  , which has the nice property of being symmetric with values close to zero/one being fine/low-sensitivity.

Training and inference

- Given N examples, we find the optimal vector β that maximizes log likelihood. The model consists only of the weight vector.
- To predict an output, compute $P(c|x)$ using the trained weights. Select the class with the highest posterior probability
- Most values lead to confident outputs $P \approx 1$ and $P \approx 0$. For binary classification, set the cutoff boundary to be 0.5. The closer the inference is to the cutoff boundary, the more uncertain the model is
- The decision boundary still comes from a hyperplane βx albeit with a logit, hence it is also linear. Namely for binary classifications, $\beta x = 0$ forms the boundary.

Parameter estimation

- The objective function to optimize is the likelihood function, which is likely concave
- Can use gradient ascent

- For binary classification where y is zero or one, define the likelihoods as

$$\begin{aligned} h_\beta(x) &= \frac{1}{1+e^{-\beta x}} \\ P(y=1|x) &= h_\beta(x) \\ P(y=0|x) &= 1-h_\beta(x) \\ L(\beta;x,y) = P(y|x) &= (h_\beta(x))^y(1-h_\beta(x))^{1-y} \end{aligned}$$

  If we have N independent samples, the total likelihood is

$$L(\beta;x,y) = \prod_i (h_\beta(x_i))^{y_i}(1-h_\beta(x_i))^{1-y_i}$$

  For simplicity, we use the log likelihood instead

$$\log L = \sum_i y_i \log h_\beta(x_i) + (1-y_i)\log(1-h_\beta(x_i))$$

  Note that this is concave, so we can use gradient ascent to find the global maxima and thus the optimal weights β.

- The gradient ascent is

$$\beta_i = \beta_i + \alpha \sum_i (y_i - h(X_i;\beta))x_i$$

- For multiclass, select one class as the pivot class. Build $|C|-1$ regression models for the other class assigned to Y and the pivot class assigned to N

$$\beta_j x = \log \frac{P(c=c_j|x)}{P(c=c_0|x)}$$

  Combine the regression models with the normalizing condition that $\sum_j P(c=c_j|x) = 1$ to get $|C|$ equations. When simplified, the posterior probabilities are

$$\begin{aligned} P(y=c_j|x) &= \frac{e^{b_j x}}{1+\sum_i e^{b_i x}} \\ P(y=pivot|x) &= \frac{1}{1+\sum_i e^{b_i x}} \end{aligned}$$

  and we maximize likelihood the same way on all $(|C|-1)F$ weights at once. The inference requires computing $|C|$ posterior probabilies and take the maximum

Comparison against naive bayes

- naive bayes models $P(c|x)$ using bayes rule on attributes
- logistic regression directly regresses on $P(c|x)$ using the weights β

Summary

- Forms a linear decision boundary
- Provides class probabilities
- Easy to interpret

13 Ensemble Methods

Ensemble learning

- Construct a set of base classifiers from training data and perform classification by considering the aggregate outputs made by all base classifiers (majority voting for example)

Intuition

- Accounts for the opinions of multiple experts rather than a single one
- The combination of lots of weak classifiers should be at least as good as one strong classifier
- The combination of a set of strong classifiers is usually at least as good as the best of the base classifier

Does combination methods work

- The base classifiers do not make the same mistakes
- Each base classifier is reasonably accurate
- This means that the classifiers should be relatively independent and be better than random prediction

Base classifier construction

- Instance manipulation, generate multiple training datasets through sampling, and train base classifiers over each dataset
- Feature manipulation, generate multiple training datasets through different feature subsets, and train base classifiers over each dataset
- Algorithm manipulation, semi-randomly tweak internal hyperparameters within an algorithm, and train a base classifier with each set of hyperparameters

Classifier combination methods

- For discrete, majority voting, weighted voting
- For continuous outputs, use an average (maybe weighted) over outputs

Bias and variance

- Metrics about the generalization error of a prediction model
- Bias is the tendency of a classifier to make systematic wrong predictions. Essentially this is how biased a trained model is
- Variance is the tendency of producing different models or predictions for different training sets under the same learner. Essentially this is how variable the model is under different training sets
- We want low bias and variance to reduce generalization errors

13.1 Bagging

Bagging is aggregating bootstrap samples

- The idea is to generate more data from a fixed dataset, because more data implies better performance
- Method is to construct new datasets through repeated bootstrap samples, say randomly sample the original dataset N times, and get a new dataset of the same size.
- Repeat this bootstrap sampling k times and train k base classifiers on each fo the bootstrap samples
- The final prediction is derived from simple majority voting of the k base classifiers

Analysis

- Same base classification algorithm for each bootstrap sample
- Reduces variances of predictions
- Good for unstable classifiers (unstable as in high variance, DT).
- Can reduce the performance of stable classifiers (like KNN)
- Effective over noisy datasets, since the outliers vanishes over sampling
- Simple sampling method through instance manipulation
- Can be parallelized training individual base classifiers
- Performance is generally significantly better than the base classifiers, and only sometimes worse

13.2 Random Tree

Random trees

- A decision tree, but the splitting criterion at each node only considers a subset of the possible attributes
- Can use a fixed proportion of all the attributes τ
- Faster to build compared to DT, but massively increase the variance

Random forest

- An ensemble of random trees
- Each random tree is trained using a different bagged/bootstraped dataset
- Combined classification using voting
- Interpretable since we can see the logic in predictions by following each random tree

Hyperparameter

- Number of trees B (can be tuned using out of bag dataset errors)
- Feature sub-sample size, the subset of features for each random tree node Increasing the features will increase the strength of each base classifier but can also increase the correlation between base classifiers. Usually use $\log_2 |F| + 1$ features per node

Properties

- A strong learner, fast to construct

- Parallelizable
- Robust to overfitting
- Sacrificing interpretability over DT

13.3 Boosting

Boosting

- Idea is to train base classifier against the hard to classify instances during training
- Method is to make a weighted array on the instances that is updated after each base classifier training to reflect the performance of the previous classifiers, and train the next classifiers based on the weighted array

Algorithm

- Initialize a weight vector with uniform weights
- Repeat: apply a weak learner (a learner that is slightly better than random) to weighted training instances; increase weight of misclassified examples and decrease weight of classified examples
- The weights represent the probability that the instance is included in the next sample dataset to train the base classifier one
- Use weighted voting for final classification

AdaBoost algorithm

- Have a set of base classifiers C_i (weak classifier), training instances (x_i, y_i), and weights $w_i^{(j)}$
- Create a bootstrap weighted sample using w_j, use it to train C_i, compute the error rate of C_i by
$$\epsilon_i = \sum_j w_j^{(i)} I(C_i(x_j) \neq y_j)$$
where I is the indicator function
- Compute the weight associated with C_i used for final voting
$$\alpha_i = \frac{1}{2} \ln(\frac{1-\epsilon_i}{\epsilon_i})$$
- Update the instance weight for the next iteration
$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z} \begin{cases} e^{-\alpha_i} & C_i(x_j) = y_j \\ e^{\alpha_i} & C_i(x_j) \neq y_j \end{cases}$$
where Z is a normalization term. The idea is to increase the weights for misclassified instances
- Iterate this training for T iterations, reinitialize the weight vectors to $1/N$ and remake C_i if $\epsilon > 0.5$

- To classify use the weighted (using α_i) base classifier votes

$$C(x) = \arg\max_y \sum_i \alpha_i I(C_i(x) = y)$$

  This final learner is a strong learner

Boosting analysis

- Base classifiers are decision stumps or DT
- Mathematically complicated but computationally cheap to train
- Always able to improve training dataset performance by making more weak learners
- More computationally complicated over bagging
- Practically often overfits

Compared to bagging

- Random forest can do parallel samples, while boosting uses iterative samples
- Bagging uses simple voting, boosting uses weighted votes
- Both uses homogeneous classifiers
- Bagging reduces the variance of base classifiers by repeated sampling, boosting reduces instance biases by focuses on hard to classify instances
- Bagging is not prone to overfitting, but boosting is prone to overfitting

13.4 Stacking

Stacking

- Idea is to smooth errors over a range of different algorithms each with different biases
- Train many different base classifiers, and use a meta-classifier over the base classifier outputs, which learns which base classifiers are reliable
- Can be trained using nested cross validation

Terminology

- Level-0 are base classifiers like SVM, Naive bayes, DT, trained on the training dataset
- Level-1 is a combination meta-classifier. Its features are created based on level-0 classifiers outputs as well as other classifier output features like confidences. Level-1 classifier creates the final prediction

Stack analysis

- Combining heterogeneous classifiers with varying performance
- Mathematically simple but computationally expensive due to training various classifiers
- Practically results in better predictions than the best of the base classifiers
- Reduces both bias and variance

- We prefer stacking over bagging or boosting when: data is complex and no single model performs consistently well, want to leverage diverse model types. The problems are that it requires more computational resources and tuning

14 Feature Selection

The goals

- Main goal is to identify good features and remove the redundant irrelevant features, all to improve the performance and generalizability
- Side goals is that seeing important features can suggest other featuers to engineer, and that fewer features will lead to smaller models that are faster to run (and potentially better generalizable models due to occam's razor)

14.1 Wrapper

Choose the subset of attributes that gives the best performance on the validation data.

A full wrapper approach will check every subset

- Pros is that it can find the feature subset with optimal performance on the validation set
- Cons is that it will take a long time

The time taken for a full wrapper approach is $2^m - 1$ if m is the number of features. This means that it is only practical for small datasets or feature sets.

Greedy approach, sequential forward selection. The algorithm repeats: evaluation on the selected attributes plus another feature, choose the best additional single attribute given the evaluations and add it to the selected attributes. We terminate if the performance (accuracy) stops improving.

- Running time of sequential forward selection is $m(m+1)/2$ due to $1 + 2 + \cdots + m$. In practice, the convergences and termination stops much earlier than this.
- However, can converge to a sub-optimal solution since it is greedy.
- Also assumes independence of features in that each feature contribute to the inference individually (and that there are no dependences between features like XOR).

Ablation approach, sequential backward selection. Reversing the greedy forward selection approach by starting with m features, and evaluate the model by dropping one feature, repeat. Termination condition is when the performance degrades by more than ϵ against the previous iteration.

- Advantage is that it removes the irrelevant features at the start, and performs the best when the optimal subset is large
- Disadvantage, running time for each cycle is longer since we start with more features (compared to forward selection). Not feasible for large dataset as well due to the quadratic time complexity
- The same worse case running time as sequential forward selection $m(m+1)/2$.

We use forward selection if the subset sizes interested is small, and backwards selection if the subset sizes interested is large and close to full.

14.2 Embedded methods

The model itself performs feature selection as part as its algorithm. For example, decision trees and regression models with regularization (L1 LASSO). Direct feature selection can still benefit these types of models due to the side goals.

14.3 Filtering methods

Evaluate the goodness of each feature by considering them separately, and drop the bad features

- Runs in linear time
- Most popular strategy
- Requires a way to determine if a feature is "good". We often define the goodness of a feature by its correlation metric with the class label

Features are good if

- They are well correlated with the interested class label
- Inversely correlated with the interesting class
- Well correlated or inversely correlated with the uninteresting class. Not as good as the other two

14.3.1 PMI

Two attributes are independent iff $P(A, B) = P(A)P(B)$. If the ratio $R = \frac{P(A,B)}{P(A)P(B)}$ is larger than one, they are positively correlated; if it is close to one, they are close to independent, and if it is smaller than one, they are negatively correlated.

Pointwise mutual information normalizes this notation of independence, define it as

$$PMI(A = a, B = b) = \log_2 \frac{P(A = a, B = b)}{P(A = a)P(B = b)}$$

filtering via PMI means to select the features with the highest PMI. The particular domain values we choose (a, b) to compute the PMI are the most interesting attribute value or label. (Can also choose some other attributes, see features are good if)

Note that

- ∞ PMI implies perfect positive correlation
- 0 PMI implies independence
- $-\infty$ PMI implies perfect negative correlation

14.3.2 MI

Mututal information combines the PMI for all combinations of interesting and uninteresting values.

$$MI(A, B) = \sum_i \sum_j P(A = i, B = j) PMI(A = i, B = j)$$

note that both i and j contain only two values, the interesting value and teh uninteresting values. Pick attributes with the highest MI.

Can build a contingency table with counts between the two attributes, this is a joint probability table. Use the contingency table to compute the MI between two attributes.

Alternatively, we can compute MI using multiple values where i and j can have multiple values. This will yield the same result as the regular entropy mutual information formula/calculations.

14.3.3 Chi squared

Use a statistical chi squared test to check if the feature and label are independent. Also uses a contingency table. The algorithm

- Assume that the feature and label are independent, compute the expected counts E for each cell based on marginal probabilities
- Compare E with the observed count O. If $O > E$ or $O < E$, it is likely predictive since the cell occurs more or less frequently than we expected under the independence assumption
- Compute the test statistic

  $$\chi^2 = \sum_i \sum_j \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

  which follows a chi squared distribution. Higher values indicates dependency and a good feature

14.4 Common Issues

The attributes are nominal with multiple values

- One strategy is to treat it as multiple binary attributes (one hot encoding), and do feature selection as normal.
- However, this can increase the dimensionality and can be difficult to interpret if a feature is only predictive when it equals to a subset of its possible values (is that feature predictive?)
- Another strategy is to expand the formula and contigency table.

The attributes are continuous

- Can discretize values
- Fit a distribution like gaussian in order to compute the marginal/joint probability densities

Multiclass classification

- More difficult than binary classification, since predictive features for a given class label may not be predictive for another class label

- Can do feature selection m times, each time treating one class label as the interested label and the others as uninterested. This allows our classifier to select features that is predictive of each class label

Importantance

- Necessary for distance based classifiers, due to it being sensitive to input features
- Naive bayes and decision trees are less necessary since they are either embedded or they assume conditional independence
- SVM works well due to regularization

15 Model Evaluation 2

A typical model train and eval consist of

- Split datasest into train and set
- Train model on train dataset
- Use the model to predict the test dataset
- Measure and eval the test predictions against their labels
- The evaluation metric assess the effectiveness of the classifier under generalization

Inductive learning hypothesis (again)

- Any hypothesis that approximates the target function well over a large training data set will also approximate the target function well over a held-out test data set.
- We need to test our hypothesis on the held-out test data set to prevent overfitting and determine generalizability
- Large training dataset implies that the dataset contains sufficient instances with properties that are important for generalization
- The size of the test set determines how confident we are about the evaluation

15.1 Overfitting

Tensions in learning

- Generalization, how well does the model generalize from training to testing dataset
- Consistency, how accurate the model is under repeated evaluation
- Overfitting, has the model tuned itself to the training data instead of the generalizable properties
- Tradeoff between consistency and overfitting

Learning curve plots learning performance over dataset size or time

- Y-axis is the evaluation metric
- X-axis is the size of training set, model complexity, or epochs

- Training learning curve computes the evaluation over the training set, shows how well the model is learning
- Testing learning curve computes the evaluation over the testing set, shows how well the model is generalizing

We can use the learning curve to select the optimal hyperparameters, by looking at where the training testing performance gap increases. This can be: the train test set proportions, polynomial degree, the svm regularization coefficient, or knn neighbor size.

An overly complex model that captures specific patterns in the training data but fails to learn the true relationship in the dataset will see a large train test performance gap, indicating overfitting.

Overfitting reasons

- Noisy dataset that makes the decision boundary noisy
- Limited training set, does not fully represent the patterns in the population. Either due to small amounts of examples or sampling bias in the training sample

Overfitting solutions

- Regularization techniques, punishing large weights in models. Examples are: soft margin SVM slack variables, CNN dropout or max pool layers.
- The norm of the parameters in linear regression can be used as a regularizer, this changes the objective function to
$$E(\beta) + \lambda\psi(\beta)$$
where λ is the strength of regularization, and $\psi(\beta) = L_p(\beta)$ is the norm function.
- Define the L_p norm as
$$L_p = \left(\sum_i |x_i|^p\right)^{1/p}$$
- L2 norm regularization is called ridge regression. It encourages solutions where most parameters are small
- L1 norm regularization is called lasso regression. It encourages solutions where only a few parameters are non-zero.

15.2 Model Bias and Variances

Bias

- Model bias, the tendency of our model to systematically make wrong predictions
- Evaluation bias, the tendency of our evaluation strategy to over or under estimate the model accuracy
- Sampling bias, that our training or test dataset is not representative of the population due to biased samples

Model bias

- To compute, compute the mean signed errors on the testing dataset (average error). If this is large, the model is biased.

- A model is biased if its predictions are systematically higher or lower than the true value. It is unbiased if the predictions are on average correct.
- For classification, model bias implies high/low bias towards the majority class. Typically a biased classifier produces labels with a different distribution than the actual distribution.
- Typically caused by incorrect model assumptions like: wrong relationships, irrelevant features, interdependence between features. This will cause underfitting since it will have a poor fit on both the training data and also the testing data.

Variance

- Model variance, tendency of different training set to produce different models/predictions with the same learner. High model variance if different training sets leads to very different predictions under the same learner.
- Evaluation variance, tendency of different testing sets to produce different evaluation metrics on the same fitted learner

Evaluation method

- The evaluation metric is also an estimator. The target is the true error rate of the model, and the sample is the measured error on the test dataset
- The training error is an ok evaluation metric. With unlimited samples, the training error will approach the true error rate

Evaluation bias and variance

- Bias is the bias of the model effectiveness metric
- Variance is the variance in the model metric under different testing sets. Can be hard to tell apart from model variance

To control evaluation bias and variances

- Tradeoff in the holdout partition sizes between model and evaluation variances
- Repeated k fold cv reduces evaluation variances
- Stratification has less model and evaluation bias
- Leave one out cv has no sampling bias, minimizes the evaluation variance

16 Sequential Model

For many tasks, there is structure between instances. These sequence structures can be: time series autocorrelation, speech recognition, genomic data between people.

Structure types

- Hierarhical structure, tree like structure like webpages in a website
- Graph structure, social media networks

To capture these interactions between instances, we need structured classification models.

16.1 Markov Chain

Markov chain

- Contains a set of states S_i, an initial state distribution π_i, and a transition probability matrix P_{ij}.
- Describes a system that transits from one state to another completely determined in probability by the previous state only, via the transition matrix
- Given a sequence of states, $q_1 \dots q_{t-1}$, the state q_t only depends on the immediately preceding state q_{t-1}, namely
$$P(q_t|q_1 \dots q_{t-1}) = P(q_t|q_{t-1})$$

To compute a joint probability, we do

$$P(q_3, q_2, q_1) = P(q_1)P(q_2|q_1)P(q_3|q_2)$$

16.2 Hidden Markov model

Definition

- Contains an underlying markov chain
- See a sequence of observations, but the sequence of states is hidden
- Defined by: states s_i, observations y_k, initial state distribution π_i, transition matrix a_{ij}, and output probability matrix $b_{ik} = P(y_k|s_i)$.
- The observations's probability distribution given the state is provided by the output probability matrix

Assumptions

- The markov chain conditional independence assumption
- The observation is dependent only on the current state

$$P(y_t|q_1 \dots q_{t-1}, y_1 \dots y_{t-1}) = P(y_t|q_t)$$

Tasks

- Evaluation, estimate the likelihood of an observation sequence
- Decoding, find the most probable state sequence given the observation sequence
- Learning, estimate parameters of the HMM given the observation sequence and state sequence

Evaluation

- If we know the hidden states, we can compute the likelihood in $O(T)$ where T is the sequence length
- Otherwise, requires $O(TN^T)$, where N is the number of states

- The naive formula

$$\begin{aligned} P(Q|\mu) &= \pi_{q_1} a_{q_1 q_2} \dots \\ P(\Omega|Q,\mu) &= \prod_t P(o_t|q_t,\mu) \\ P(\Omega|\mu) &= \sum_Q P(o_t|Q,\mu) P(Q|\mu) \end{aligned}$$

- Use dynamic programming (forward algorithm) to speedup the computation. Define $\alpha_t(i)$ as conditional probability (forward probability) of the observation sequence ending at t at state s_i given the model μ, use the following algorithm

$$\begin{aligned} \alpha_1(i) &= \pi_i b_i(o_1) \\ \alpha_{t+1}(i) &= \left(\sum_j \alpha_i(j) a_{ji} \right) b_i(o_{t+1}) \\ P(\Omega|\mu) &= \sum_i \alpha_T(i) \end{aligned}$$

- DP algorithm time complexity is $O(TN^2)$.

Decoding

- Find the most probable state sequence
- A bruteforce algorithm enumerates probabilities for all hidden state sequences, $O(TN^T + N^T \log N^T)$
- Cannot guarentee the most probable sequence, however, since the transition probabilities can be small and floating point errors
- Viterbi algorithm is a more efficient version. Define the $\delta_t(i)$ as the max probability of the observations ending at t in state s_i. To keep track of the path, let $\psi_t(i)$ be the last state in the most probable partial sequence in $\delta_t(i)$.
- The formulas are

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(o_1) \\ \psi_1(i) &= -1 \\ \delta_{t+1}(i) &= \max_j (\delta_t(j) a_{ji}) b_i(o_{t+1}) \\ \psi_{t+1}(i) &= \arg\max_j (\delta_t(j) a_{ji}) \end{aligned}$$

  It is just dp on the $\alpha_t(i)$ and keep the max in $\delta_t(i)$ and keep track of the previous optimal state in $\psi_t(i)$.

- Termination

$$\begin{aligned} P^* &= \max_i \delta_T(i) \\ q_t^* &= \psi_{t+1}(q_{t+1}^*) \end{aligned}$$

  where we use backtrack on ψ to find the optimal state. The last state is the maximum δ_T.

- Time complexit is $O(TN^2)$.

Learning

- Given a set of labeled dataset, supervised, (state sequence estimated from observation sequences), it is possible to use maximum likelihood estimation on the markov chain
- Formulas are

$$a_{ij} = P(s_j|s_i) = \frac{\#(s_i, s_j)}{\#s_i}$$
$$b_{ik} = P(o_k|s_i) = \frac{\#(y_k, s_i)}{\#s_i}$$
$$\pi_i = P(q_1 = s_i) = \alpha\#(q_1 = s_i)$$

- If no state labels, unsupervised, use forward-backward algorithm to label them, not examined

Analysis

- Highly efficient approach to structured classification, built with limit context consist only of observation sequences
- Suffers from floating point underflow. Can use scaling coef in forward algorithms, use logs in viterbi algorithm

16.3 Applications

Parts-of-speech tagging

- Given a series of text, tag each word by their english tag. The states are their tags, while observations are words

OCR

- Given an image
- The states are the letter true values, and observations are pixels

Other cases

- To handle vectorized outputs, we can model them as multiple separate HMM insteads by assuming that the outputs are conditionally independent given the state. Otherwise, additional output values will increase the HMM complexity
- Can also relax the condition where the state depends on the previous states, and output depends on the previous output. This makes the algorithm take exponential time, however, and making everything more complex

17 Neural Network

Representation learning is a way to learn the main idea of text/image.

- On text, use bag of words to generator word frequency features.

- Problem is that this ignores word order and context, and faces curse of dimensionality where the word space is very large and sparse
- On images, use a bag of pixels method on each color channel (color histogram).
- This can work for constrainted tasks. But complex tasks that requires object and shape detection will required custom features. Also faces curse of dimenionality in the pixel color space.

NN respresentation learning

- Common application with neural network
- Networks learn a hierarchy of features, from simple combinations of the inputs to more complex features. These inner computed complex features are called embeddings
- The embeddings are low dimensional representations/summary of the input. They can be useful for a range of other tasks (like summaries) other than the task that the neural network was trained for

A biological neuron

- Idea is that neurons that fires together will wire together. In that overtime, there will be more weights on features associated with the label, and less weights on features not associated with the label.
- Signals goes into a neuron, which combines the signals
- Outputs an activation if the sum reaches a certain limit

An artifical neuron

- Inputs are summed in a weighted way
- Added a bias
- Passed through an activation function
- A basic unit of an NN. Input is a vector x, output is a scalar y, weights w, and hyperparameter f. The function is $y_i = f(w \cdot x_i + b)$

Perceptron

- A neural network with just a single neuron
- A binary classifier defined by

$$f(w \cdot x + b) = \begin{cases} 1 & w \cdot x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

  if the result is above 0, output 1. Otherwise, output 0. This is a step function activation function.

Training perceptron

- Finding the weights that minimizes errors on the training data
- Iterate over samples in the training dataset, compute prediction, update weights against actual label. Each iter is called an epoch

- The weights updating formula is

$$w_j = w_j + \lambda(y_i - \hat{y}_i)x_{ij}$$

  where $\hat{y}_i = f(w \cdot x_i + b)$. Where $\lambda > 0$ is the learning rate. This causes weights for inputs that underestimate the prediction to increase, and vice versa.

- Convergence is when there were no updates in the epoch. We can end training when convergence happens

Activation function

- In perceptrons, maps the linear response to the desired range
- In multilayer perceptron, added non-linearlity to the network.
- Common choices are, sigmoid, hyperbolic tan, and relu

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$
$$f(x) = \tanh(x) \tag{2}$$
$$f(x) = \max(0, x) \tag{3}$$

Perceptron properties

- Guaranteed to converge for linearly separable data.
- Convergence point (class boundary) depends on the initial weights and bias values and the learning rate
- Does not guarantee to maximize margins between classes, nor to converge over non-linearly separable datas
- With sigmoid acitvation (plus a binary step to convert output to 0 or 1 against 0.5), it is equivalent to logistic regression

17.1 Multilayer perceptron

Extension

- Can extend perceptrons to a matrix of weights mapping to multiple outputs, with each inputs output pair as a seperate neuron.
- Can extend perceptrons in the number of layers, each layer will contain a set of neurons

Definition of MLP

- A layer is a group of neurons (size n) working in parallel on the same input, to produce n outputs. Each layer will receive input from the previous layer.
- Has an input layer, at least one hidden layer, and one output layer that outputs the class label.

Hidden layer

- In representation learning, these layers learn features that are useful for the final output layer

- They often represent abstract concepts like shapes or edges
- Problem is that we don't know how many features the input has
- In theory, the number of features depends on the complexity of the decision boundary. In practice, pick an arbitrary value between the input and output size

Learning MLP

- Not just one correct output for the neurons in the hidden layer
- Use stochastic gradient descent to update the weights and biases on the entire MLP using backpropagation. This will minimize the entire MLP loss function
- Backpropagation requires a learning rate. The idea is to look at the derivative of error against each parameter, and update these parameters. It first compute error gradient at the output layer using partial diff, and propagate gradients back to earlier layers using chain rule

Output layer

- Structure depends on task.
- If binary, use step activation function with 1 neuron
- For n way classification, use n neurons with softmax activation
- For regression, use one neuron with linear activation

MLP properties

- Universal approximation theorem, a MLP with a single hidden layer and finite neurons can approximate any function in R^n.
- This implies that the network can learn any function dynamically, without requiring hyperparameter tuning a kernel function
- This requires a non-linear activation function. If the activation function is linear, the layers will reduce to a single linear layer, and hence doesn't fit the universal approximation theorem
- NN still requires feature engineering sometimes. They also don't guarantee to generalize better than other learner since they can still overfit.

MLP pros and cons

- Can be adapted to many types of problems: classification and regresseion
- Universal approximation that can approximate arbitrary functions
- Representation learning in the hidden layers
- Has a lot of parameteers, so slow to train, prone to overfitting, and requires lots of computing resources
- Stochastic gradient descent is random and does not guarantee to converge to the same solution everytime

18 Deep Learning

Deep Learning

- Defined by neural networks with a large number of hidden layers
- Network depth is the number of layers. Network width is the number of neurons per layer
- Focuses on representation learning. Transformation of raw inputs into a smaller latent space that is easier to learn

History

- Started in 2010 where Neural networks became good
- But the concepts were introduced in 1950s
- What changed were GPU, algorithm improvements, and the internet.
- The introduction of GPUs for fast parallel computation (CUDA). Improvements in algorithms like pretraining, ReLU, regularizaton. The internet in providing large amount of training data in the forms of text and images

Large image datasets

- ImageNet
- Open images dataset
- Youtube-8m

18.1 Architectures

CNN layers

- Convolution layers with maxpooling and batchnorm
- Fully connected layers with dropout

Convolution

- Focus on repetition of locally similar features
- The goal is to connect output neurons to only a local subset of the input features
- Defined by: the kernel matrix overlaid on the input and computes an element wise product with the input subset (which are summed together), a stride that dictates how many positions in the input to advance the kernel for each output neuron

Convolution Kernel

- Kernel is a matrix with trainable weights that is moved across the image. For each position, compute the elementwise product and sum to get the convolution output
- Essentially, the kernel captures input features similar to its weights
- The first layer kernels can be used to detect horizontal/vertical edges on the input
- The second and later layer kernels are computed on the outputs of the first layer kernels. They are typically hard to interpret.

- The layers learn a hierarchy of image features, where each layer learns some features of the image at a different scale: edges, shapes, complex objects.

A convolution layer is defined by: kernel size, stride, number of output layers. Each output layer has a separate trainable kernel.

Fully connected layers vs Convolutional layers

- Fully connected has each neuron connected to every neuron in the last layer. Conv has each neuron connected to a small patch of the last layer's outputs
- Fully connected learns the weights of a linear combination of the last layer. Conv learns a convolution kernel
- Fully connected outputs are neuron responses. Conv outputs are the inputs convoluted with the neuron's kernel

Convolution layers pros and cons

- Efficient, since each neuron learns to recognize a single feature everywhere in the input
- Preserves spatial relationships since location information are preserved in convolutions
- Limited kernel size means that a single convolution layer can only learn local features

Max pooling

- Within a small window in the kernel's output, take the highest output and discard the rest
- The dimension is the size of the window. The stride is the number of positions to move the window
- Improves translation invariance, since network response is similar even if features are slightly shifted in the image
- Reduces the amount of data to add regularization and reduce computation time
- May discard important information

Dropout

- Randomly discard some neurons with a fixed probability
- A regularization method that forces each neuron to find independent useful features from the previous layer
- Essentially just training multiple neural networks in parallel

18.2 Training

For n-way classification

- Contains N output neurons
- Processed with a softmax function that maps the outputs to a probability distribution
- The loss function is the cross entropy loss, defined by

$$E = -\frac{1}{N}\sum_i y_i \log \hat{y}_i$$

where y_i is the ground truth probability for the instance at class i, and $\hat{y}_i$ is the softmax output.

- This essentially moves the softmax output closer to a probability distribution where the correct class is one

For regression

- One output neuron
- Identity activation function
- MSE loss

Training steps

- Initialize weights and biases. Small random values for weights and zeros to biases
- Set hyperparameters of batch size and learning rate
- For each epoch, split training data into batches. For each batch, classify batch, compute loss, update model parameters via backpropagation
- Monitor validation loss and stop when that is not improving

Regularization

- Due to the high number of parameters, CNNs are prone to overfitting even on large datasets
- Regularizations are used to reduce overfitting: L1,L2 regularization, dropout, early stopping
- L1, L2 regularization adds an extra term to the loss.
- Early stopping keeps parameters close to their original values which is near 0
- Dropout essentially creates multiple networks bagged with a subset of features

18.3 Generalization performance

CNN achieves human like performance.

Embedding

- The embedding of an input for a trained neural network is the network response to the input at some layer, typically at some fully connected layer. These embeddings are good informative representations of inputs for a range of other task.
- Can get improved performance by adding another layer after the embedding layer and training it on the specific dataset fixing the original network weights, essentially treating the embedding as features

Visualization

- We can visualize the classes by looking at the embedding layer values given an image (typically high confidence and low confidence images).
- We can also look at informative patches.

However, models train on one task may not generalize well to a similar related task. Adversial images are

- CNN models are very sensitive to some type of noises, like appplying a kernel on the original image
- Can delibrately add some amount of noise to image to trick the CNN outputs

19 Generative Models

Discriminative vs generative

- Discriminative models learns the conditional probability of class Y given attributes X
- Generative models learns the joint probability of class Y and attributes X. Note that this can be rearranged to get a conditional probability (discriminative models). It can also due to reverse, the prob dist of attributes given class.

Generative models

- In theory, provides a better reasoning since it captures a complete model of the world.
- In practice, we often only care about discriminating between the classes, so a discriminative model performs better
- Can compute model uncertainty, and detect out-of-distribution data
- Can generate new samples from the distribution

Simple cases

- If we know the generating process, or when we can approximate it with a simple model, we can easily generate the textures
- Procedural texture generation
- Gassuian mixture models
- Or more complex generative algorithm
- But outside of these special textures, image generation cannot be approximated by a simple model

Complex cases

- Model the pdf that we want to generate
- Often hard to model it direction, instead, three approaches
- Map to low dimensional latent space. Autoencoders and variational autoencoders
- Learn a function that convert samples from a simple pdf to the target pdf. Generative adversial networks (GAN)
- Learn the gradient of the pdf and move along the gradient to generate more probable samplse. Score based models, diffusion models

Training

- Requires a lot of data in the tb
- Often unsupervised requiring no human labels

- For images and text, hide part of input and ask the model to predict it. Masked language modelling (predict missing words). Image denoising (removing gaussian noise from image)

19.1 Autoencoders

Definition

- A neural network that outputs the same image as the input
- Hidden layers learns a low dimensional latent representational representation of input
- Variational autoencoder adds constraints to the autoencoder so we can use the latent layers to generate new samples

Topology

- Encoder layers from input to hidden latent layer
- Decoder layers from hiddent latent layer to output layer
- Often symmetrical between encoder and decoder with identical weights
- The hidden layer should have smaller size than input/output (fewer neurons), since we want to project to a lower dimension
- The hidden layer is called the bottleneck layer. It represents the input in terms of latent variables. If one hidden layer with linear activation function, it produces PCA
- If image is binary, use tanh and sigmoid for last layer activation. For grayscale or colored images, use linear activation function for last layer.

Summary

- Learns a smaller, latent variable representation of input
- Uses complex features of inputs to compute latent variables
- Variational autoencoders used to generate new instances
- Deeper versions can be difficult to train

19.2 GAN

GAN

- Uses neural networks that learns to generate instances from a given distribution
- Consists of two network, a generator and discriminator
- Training involves a competition between the networks. The generator aims to generate a fake image that looks real. The discriminator aims to differentiate between fake and real images (binary classifier)

Generator

- Learns a probability distribution over the images by having its input sampled from random variables (random n dimensional values) and outputs as the image.
- Essentially inverting the CNN model

Discriminator

- Aims to identify between real and fake inputs
- Two classes, real or fake
- Architecture depends on tasks, usually CNN for images

Training method

- Given generator G and discriminator D, each mappng input to outputs
- Real data/images is X and generator input is z (random valuess).
- Generator generates $G(z)$, dscriminator generates $D(G(z))$ or $D(x)$.
- Discriminator goal is to maximize $D(x)$ and minimize $D(G(z))$.
- Generator aims to maximize $D(G(z))$.
- Loss from D is feed back into generator and discriminator
- Treat this as a zero sum game with the goal of finding an equalibrium between D and G.

Training issues

- If discirminator is too good. Not much information to train the generator since all fake inputs are rejected
- If discriminator is too bad. It can easily be confused by fake inputs. Generator will learn a poor solution
- Difficult to train by finding a balance between generator and discriminator

Evaluation

- The generator outputs should look like inputs.
- The output should not be identical to inputs (from memorization)
- The output should be ass diverse as real data. Avoid mode collapse, where the generator only creates one of few output classes
- To identify memoziation look at the nearest neighbours in the training set of a generated image by latent space see if they look similar
- Hard to tell if it captured diversity in the training set
- Often ignores physics in the generated output

Summary

- Can model and generate samples from complex probability distributions
- Unstable and hard to train
- Difficult to evaluate. Even if the performance looks good, the learnt probability distribution may not be correct due to biases

20 Unsupervised learning

Take the embedding of a NN on a set of images without labels, we can use clustering to make distinct clusters that groups similar images together. The size and distant of the clusters defines how well the neural network has learned.

The nearest neighbors at each embedding layer demonstrates what the network has learnt so far — shape identifying layers will net images with the same shape, color identifying layers will create similar colored images.

20.1 Clustering

Supervised learning separate datapoints into classes given labels. Unsupervised learning has no labels, hence requires assigning the datapoints into clusters given their features.

Clustering

- Unsupervised learning, no explicit definition of classes
- Learnt structure from data
- Requires some assumptions of the expected data structure: exclusive or overlapping clusters, hierarchical clusters, evaluation of a good cluster

Deterministic vs probabilistic clusters

- Deterministic means each instance is a member of one cluster only. No overlapping of clusters. Creates hard boundaries between clusters
- Probabilistic means each instance has a weight for belonging in each class. Allows for overlapping in clusters. Creates probability distribution per cluster in space
- Use deterministic clustering if hard boundaries are required or likely. It is also fast and efficient. Use probabilistic clusters if clusters are likely to overlap with complex cluster shapes

K-means clustering

- Select K points at random as intiail centroids
- For each instance, assign it to the cluster with nearest centroid
- Compute new cnetroid for each cluster as mean of all instances in cluster
- Repeat until centroids converge (no instance assignment changes)

Soft k-means clustering

- K-means creates determininstic hard clusters. Can modify to allow probability-clusters.
- Use a softmax function per instance on its distance to each centroid

  $$\sigma(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

  where x is a specific instance and x_i is the distance to centroid i. The goal is to scale x such that it is a probability distribution

- Softmax produce a vector with a probability distribution: values within 0 to 1, sums up to 1. Commonly used to normalize a multiclass classifier
- Process for soft k-means. Select random centroids μ_j, compute softmax for all instance cluster pairs where
$$z_{ij} = \frac{e^{-\beta|x_i - \mu_j|}}{\sum_k e^{-\beta|x_i - \mu_k|}}$$
is the probability that instance i is in cluster j given centroids μ_j. $\beta > 0$ and is a stiffness parameter. Update centroids using a weighted average of its members
$$\mu_j = \frac{\sum_i z_{ij} x_i}{\sum_i z_{ij}}$$
Repeat til converge
- Produces overlapping probabilistic clusters

20.2 Mixture models

Mixture

- Finite mixture is a distribution with k component distributions. Used to represent latent factors in a dataset/distribution
- Gaussian mixture model represents a distribution as k Gaussian distributions

Expectation maximization

- Parameter estimation method with guaranteed positive hill-climbing characteristic relative to the log likelihood gradient. Means that we can always improve log likelihood using it
- Used to estimate hidden parameter values like cluster probability distributions when the isntances are unlabeled

GMM with EM

- A generization of soft kmeans
- Expectation step, given current parameter estimates, assign clusters to soft labels.
- Maximization step, find parameter values that increases overall log-likelihood given the soft labels from Expectation
- Repeat expectation and maximization step until convergence. Convergence is defined when the log-likelihood stops improving or reaches a threshold. The resultant parameters will maximize log-likelihood

Log likelihood of a given finite mixture is
$$L = \sum_i \log \left(\sum_j P(c_j) P(x_i | c_j) \right)$$
where x_i is an instance and c_j is a cluster/class. The EM algorithm aims to improve this loss each iteration as it estimates the goodness of the clusters. We consider convergence when the change in L falls below a certain threshold.

GMM example

- Two normal distribution clusters. Probability in dist 1 is p and 2 is $1-p$. Total probability density is
$$g(x) = p\phi_1(x) + (1-p)\phi_2(x)$$
where ϕ is the normal pdf. Log likelihood is
$$\sum_i \log(p\phi_1(x_i) + (1-p)\phi_2(x_i))$$

- Minimizing L directly is hard, but if we condition on the cluster/distributioin that generates the instance, we can set one of the ϕ to zero leading to
$$\sum_{j \in D_1} \log(\phi_1(x_j)) + \log(p)$$
and equivalent in $j \in D_2$

- We don't know which distribution generated each instance. But given some parameters, we can estimate how likely each distribution generated each instance.

- Iterative EM algorithm will perform: soft assignment of instance to distributions given current parameters in expectation step, update parameters based on assignment to maximize L in maximization step

EM iteration

- Initial means are random instances, initial variances are sample variance, initial class probability are uniformlly assigned

- Expectation step. Compute responsibility
$$\hat{y} = p\phi_1(x)/(p\phi_1(x) + (1-p)\phi_2(x))$$
which is likelihood that x is generated from distribution 1.

- Maximization step. Compute responsibility-weighted mean and std of all instances per distribution, and use that to update distribution parameters.

- Repeat expectation and maximization step by recomputing mixing/class-probabilites using the updated parameters.

- After convergence, parameters are locally maximized for log-likelihood

EM algorithm pros and cons

- Guaranteed positive hill climbs

- Fast to converge

- Creates probabilistic clusters

- Has an element of randomness as final model may depend on initial parameters

- Can get stuck in local maxima, not guaranteed to be global maximum-likelihood solution

- Requires hyperparameter of the number of clusters

20.3 KDE

Modelling distributions. Given data points, how do we represent the distribution that this data originated from

- Can discrete into k bins
- Model as a single gaussian distribution
- Model as a mixture of k Gaussians (or other mixture distributions). This is a GM model
- KDE

Kernel density estimation

- Formula for normal KDE is
$$f(x) = \frac{1}{N} \sum_i \phi_\sigma(x - x_i)$$
where σ is the std of the normal kernel function. Equivalent to applying a guassian distribution per datapoint and summing them up
- The std here is the kernel bandwidth (how much local region each point affects). Lower std will be more jagged. Higher std will be smoother

KDE pros and cons

- Can model arbitrary probability distributions given kernel
- No assumptions about the distribution shapes
- Needs to choose a kernel and bandwidth
- Need lots of parameters to represent final PDF (linear to data points). Slow to compute PDF

20.4 Unsupervised evaluation

Goal of unsupervised learning

- Mapping high dimensional data to a smaller set of clusters with latent factors
- Discover relationship or trends in data
- Can model probability distributions for some models. For instance, use KDE for the conditional likelihood in Naive bayes
- Generate new samples from the modelled probability distribution

Evaluation problems

- No ground truth labels. Cannot tell if a model is more correct than another
- Requires subjective evaluation like checking similarity of clusters over separate iterations or doing cross-validation using an unsupervised metric. If labels are available, do cross-validation on clustering to see how well the clusters match labels.

Unsupervised Evaluation metrics

- Cluster cohesion, how close instances are within each cluster

$$Co(C_i) = \frac{1}{\sum_{x,y \in C_i} dist(x,y)}$$

- Cluster separation, how far instances in each cluster is from each other

$$Sep(C_i, C_j) = \sum_{x \in C_i, y \in C_j} dist(x,y)$$

- Cluster compactness (especially for k-means) is the sum of squared errors against the centroid

$$SSE = \sum_i \sum_{x \in C_i} dist(x, c_i)^2$$

  where c_i is the centroid of cluster C_i.

- Distribution metric is euclidean for numeric, and manhantten for nominal
- Good clusters have high cohesion and separation, with low SSE.

Supervised cluster evaluation, assumes that labels are available

- Purity is the average proportions of the most frequency label within each clusters

$$\sum_i \frac{|C_i|}{N} \max_k Prop(k, i)$$

  where $Prop$ is the proportion of k labels in cluster i

- Entropy is the weighted entropy of each cluster

$$\sum_i \frac{|C_i|}{N} H(x_i)$$

- Good clusters have high purity and low entropy

21 Big data

Simple models with lots of data trumps complex models with fewer data. Typically, the bottlenecks are the labeled data, with models and computing power increases overtime but labeled datasets staying constant (model depth increases but size of large-scaled datasets has not kept pace).

Data importance

- Models trained using more data often beats models trained using fewer. Adding data is nearly as effective as adding layers
- Similarly, increasing model complexity will also yield better performance, but will cap out if the dataset size is fixed. More parameters are not helpful unless there are more data to train them

Labelling bottleneck

- Abundent data but expensive labelling
- 400 hours of annotation time per hour of speech
- 30-60 minutes per image for complete segmentation

21.1 Data augmentation

Augmentation is generating new labeled datasets with the existing dataset. Methods like: resampling, manipulation, synthesis

Resampling (bootstrap sampling)

- Creates new datasets by resampling existing data with or without replacement. Common in mini-batch. Benefit of each batch dataset having different instance distribution in features, forcing models to use different features and not get stuck in local minima

Data manipulation

- Adds dataset specific artifical variations to each instance, without change label.
- For images, adjust brightness, flip direction, shift, resize, rotate
- For audio, volume, time shift, frequency shift
- For text, synonym substituion
- the idea is that the manipulation should not change the instance label. These variations should reflect those that appear in real-world data.
- Need to be careful about manipulations given the dataset: flipping a speedlimit sign can generate incorrectly labeled data

Data synthesis

- Create data using another ml method
- Train probability distribution on labelled data, then sample this dist to generate new instances
- Use GAN or autoencoders
- Often this exploit/uses algorithms designed for other task: use computer generated images (CGI) to train image classification, or translation output to text generation

Data augmentation pros and cons

- More data typically improves learning
- Most algorithms are robust to noise in augmented data
- Can create biased training data
- Can introduce features that don't exist in the real world
- Can propagate errors in existing learners
- Makes interpretability and transparency of models harder, as the generated instances has no origin and we cannot interpet model errors made to instances generated at runtime that are not saved.

21.2 Semi-supervised learning

Semi-supervised learning is learning from both labelled and unlabelled data

- Training data has L labelled instances and U unlabelled instances. Often U is much greater than L
- Goal is to learn a better classifier from L and U combined compared to just L

21.2.1 No labelling

Combination approach

- Combine a supervised model and unsupervised model
- Find clusters in combined dataset, choose label for each cluster by labelled instances and apply to unlabelled instances

Self-training (bootstrapping)

- A form of combination approach. The unsupervised model is clustering based on current model
- Train model f_i on L using supervised learning
- Apply f_i to predict labels in U
- Identify subset U' with high-confidence labels
- Move U' into L with its predictions as ground truth
- Repeat until L does not change

Self-training details

- Need to assume that points that are nearby likely has the same label
- Not really creating data from nothing, as we are labelling existing data
- Errors are also propagated. May reduce this by moving points back to unlabelled if their classification confidences falls below threshold.

21.2.2 Active Learning

Active learning

- Hypothesis is that a classifier will achieve higher accuracy with fewer overall training instances if it can select the training instances. Labelling is expensive, so we should label instances that maximize learning
- Active learners pose queries (unlabelled instances) that are labelled by oracles (humans)
- Typically choose instances that have low confidence — either far away from existing points or have low confidence near boundaries.

Query strategies, uncertainty

- We want instances that are most effective in improving models. But to do this, we need to know the model likelihood assignment to points or how labels are distributed over labelled instances.

- Therefore we should query instances with high uncertainty in classification. This can be done by selecting the instances that the classifier is the least confident on (single degree)

$$x = \arg\max_x (1 - P_\theta(\hat{y}|x))$$

Or when the margin between the top two predictions are low (two degrees)

$$x = \arg\min_x (P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x))$$

Or use entropy on the likelihoods of classification (all degrees)

$$x = \arg\max_x -\sum_i P_\theta(\hat{y}_i|x) \log_2 P_\theta(\dots)$$

where we focus on classification with high entropy

Query by committe

- When there are multiple diverse classifiers
- Train multiple classifiers on labeled dataset. Predict on unlabelled data, select instances with highest disagreement between classifiers (measured by entropy or simple counts).
- Ideally all classifiers learns different features, hence disagreement provides information in finding hard to classify (uncertain) instances that are valuable to label
- Vote entropy formula

$$x^* = \arg\max_x -\sum_{y_i} \frac{V(y_i)}{n} \log_2 \left(\frac{V(y_i)}{n} \right)$$

where x are the instances, n is the number of classifiers, and y_i are each classifier's output for the given instance x. Select the instances with the highest vote entropy.

Active learning pros and cons

- Empirically is a more robust strategy to increase accuracy
- Often hard to justify these strategies theoretically
- Introduces bias and that could result in a dataset not suitable for ml tasks not related to this one
- Can be sensitive to label noise in new labels, with their errors propagating to future queries

22 Data considerations

Questions

- Does the training set reflects the real world? We have a tendency to use data that is convenient but not necessarily representiative (twitter)
- Do we want to more accurately reflect the real world (or a scenario)? To make a model for the past, we can undersample minority groups or replicate historical biases
- Do we have right to use this data? Data ownership, copyright, privacy

Examples

- Imagenet dataset geographically are all distributed from US, which may not be representative of the real world depending on where you live. Different objects due to culture that are the same can have differing prediction accuracies
- Imagenet model classification accuracies are worse on nations not US

Dataset bias

- Less represented groups: will contribute less to loss function, potentially having more errors on data from this group; having poorer model fit on this group due to limited training data; having poorer generalizability in general due to lack of diversity
- Continuous learning methods that learns from data generated from models trained on the biased dataset will become more biased. (underpresentedd groups becomes more underpresented due to poorer accuracy and thus less data)
- Typically leads to SIMILAR performance on the test set since it is also biased.

Dataset bias mitigation

- Treat it as an imbalanced dataset problem: use data augmentation or oversample to increase underrepresented group samples; adjust loss to put more penalty on errors in underrepresented groups (F1)
- Can force model output to be independent on variables that is less diverse
- Drawbacks are that they requires knowing the bias in the dataset, which is hard since the test set doesn't show the bias. They also don't remove the problem of low diversity of underrepresneted groups
- Recommendation is to check dataset for biases that you can think of. Be aware that they can be biased in ways you haven't thought of

Just because data is on the internet doesn't necessarily mean that you can use it to train AI. There may be legal issues around using data this way even if they are publicly available.