Machine Learning

Supervised Learning
Supervised Learning

- Supervised learning is learning where the training data contains the target output of the learning system.
  - Training data: $D = \{(x^{(i)}, y^{(i)}): i \in \{1..n\}\}$
  - The type of output data determines the type of supervised learning problem
    - If $y$ is a real number it is a regression problem
    - If $y$ is an enumeration type it is a classification problem
Supervised Learning

- The task of supervised learning is to learn a function (hypothesis), $h$, such that $h(x^{(i)})$ is a “good” predictor for $y^{(i)}$: $h : X \rightarrow Y$
  - The representation used is an encoding of $h$
  - The hypothesis space is the space of all functions that can be represented with the representation
  - The evaluation/performance function determines what “good” means
  - When probabilistic criteria are used, $h$ can also be chosen to represent $P(y)$ followed by either a MLE, MAP, or expected value pick of $y$
Supervised Learning Representations

- The space for all possible supervised learning solutions is the space of all functions
  - Infinitely large, requires infinite representation
    - It is intractable to design an algorithm with a universal and precise hypothesis space
      - Can limit the type of function
      - Can limit how precisely to represent the function
    - Parametric representations limit the type of function and encode the function using a set of parameters
    - Non-parametric functions limit the precision to which they model the function and represent it in terms of weighted data points
Supervised Learning

Representations

- Example representations include
  - Parametric
    - Polynomials of fixed degree
    - Parametric exponential functions
    - Neural networks
    - ...
  - Non-parametric
    - Point-based density estimators
    - Point-based mixture distributions
    - Tile coding
    - ...

© Manfred Huber 2019
Point-Based Estimators

- Point-based estimators are a non-parametric way to represent arbitrary functions with the precision limited by the number of points.
  - Each data point represents the location of a specific (kernel) function and its weight represents the magnitude of its contribution.
    - Kernel functions define the change in function value contribution when moving away from the point in terms of a similarity measure.
  - The function estimate at another point is the sum or average of all the kernel functions' values at that point.
Point-Based Probability Density Estimators

- To (approximately) represent an arbitrary probability density function we can use a point-based estimator where the kernel function is a valid probability density function and the weights are the probability of the point contributing (i.e. weights are a probability distribution over points)
  - At any point, the probability density is the weighted sum of the probability densities of the kernel functions at that point
    - The more points used, the more precise a function can be represented
Point-Based Probability Density Estimators

- Computation can be reduced by limiting how many points are used for any evaluation.
  - If the kernel function has limited extent (is 0 everywhere outside a certain region), we only need to use a subset of the points to estimate the density.
    - E.g. Parzen window kernel:
      \[
      k(d/h) = \begin{cases} 
      1 & |d_i| \leq 0.5 \times h \\ 
      0 & \text{otherwise} 
      \end{cases}
      \]
    - The expected number of points, \( n_v \), contributing to any estimate is related to the probability density at that point and the size of the window, \( V = h^D \):
      \[
      p(x) \approx \frac{n_v}{nV} = \frac{\sum_i k \left( \frac{x - x_i}{h} \right)}{n \ h^D}
      \]
Point-Based Probability Density Estimators

- Alternatively, the size of the kernel function can be regulated such that a constant number of points contribute to the estimate.
  - Easiest using a hyperspherical window rather than a Parzen window:
    \[
    k(d/h) = \begin{cases} 
    \frac{1}{V_h} & |d| \leq h \\
    0 & \text{otherwise}
    \end{cases}
    \]
  - The value for \( d \) (and thus \( V \)) at a point is set such that exactly \( k \) data points fall within it. Then:
    \[
    p(x) \approx \frac{k}{n V_{h(k)}}
    \]
One of the simplest machine learning approaches is K-Nearest Neighbor (KNN)

- Given a set of (class) labeled data points and a query $x$
  $$\{(x_1, c_1), \ldots, (x_n, c_n)\}$$
  - Can also be used for regression problems
- Pick the $k$ data points most similar to $x$
- Return the label that is associated with the largest number of the data points

KNN is a supervised learning algorithm using a point-based density representation
KNN Classification

- What is the answer to the KNN learning problem?
  - Data: \{((1, 1); S), ((2, 1); F), ((2, 2); F), ((1, 3); S), (3, 3; S)\}; K=3; Point=(1.25, 1.75)
  - Answer:
    - Three nearest points (using Cartesian distance) are: ((1, 1); S), ((2, 2); F), ((2, 1); F)
    - Answer is F
  - Why is that a good answer?
    - What would the elements of a formal learning algorithm be that gives us this solution?
KNN Classification

- Representation
  - Hypothesis space:
    - Set of point-based class probability distributions
  - Representation:
    - sets of labeled data points and a distance/similarity measure forming a kernel-based probability distribution
      - In the simplest form the kernels are spherical windows
    - Data points represent limited range probability estimates supporting the class label

\[
p(x \mid C) \approx \frac{k_c}{n_c V_{h(k)}}; \quad p(x) \approx \frac{k}{nV_{h(k)}}; \quad P(C) \approx \frac{n_c}{n}
\]
KNN Classification

Evaluation

Performance function:

- Probability of a data point at \( x \) having label \( l \) assuming generalization through the chosen kernel function
  - In standard KNN the kernel function is a hyperspherical window

\[
P(C \mid x) = \frac{p(x \mid C)P(C)}{p(x)} = \frac{k_c}{n_c} \frac{n}{n V_{h(k)}} = \frac{k_c}{k}
\]

Optimization criterion:

- Maximum Posterior Estimate (MAP) in the spherical region around \( x \) containing \( K \) data points

\[
\hat{C} = \arg\max_C P(C \mid x)
\]
KNN Classification

- KNN is a simple algorithm that provides an approximate MAP estimate of the class
  - Makes a number of assumptions
    - Probability density interpolates according to kernel
      - Weighted KNN corresponds to distance-dependent kernels
    - Point estimates are independent
  - Can be extended to regression problems by using weights and estimated value as the output
  - Requires no training time but significant prediction time that grows with the size of the training set
KNN uses a non-parametric representation to learn MAP classification
- Representation contains all data points in the training data set
- There are ways to reduce this size

Other, parametric representations can be used for the same problem
- Simplest representation would be storing the underlying parameters of the MAP problem
  - $P(c)$, $P(x|c)$ - Bayesian Learning
Bayesian Classification

- Bayesian classification determines the class with the highest a posteriori (MAP) probability for a given input using the training data set.
  - Class is picked as the one with the highest MAP probability.
  $$
  \hat{c} = \arg\max_c P(c \mid x) = \arg\max_c \frac{P(x \mid c)P(c)}{P(x)} = \arg\max_c P(x \mid c)P(c)
  $$

- Training data set is used to learn the best parameters for the classifier $P(x \mid c), P(c)$.
  - The parameter space is exponential in the number of features $x_i$ in $x$.
    $$
    x = \langle x_1, \ldots, x_m \rangle
    $$
Bayesian Classification

- Bayesian classification using conditional probabilities as parameters is intractable
  - Number of features grows rapidly
    - Very fast exceeds the amount of memory available
  - Often more parameters than data points
    - Parameters can not be learned well from the data
- To use Bayesian classification, some assumptions have to be made
  - Conditional independence of features given a class
Bayesian Classification

- Conditional independence of some of the features leads to a reduction of the parameter space
  - Partial conditional independence – features are conditionally independent of some other features, but not all of them
    - Bayesian networks
  - Complete conditional independence – all features are conditionally independent given the class
    - Naïve Bayes
Naïve Bayes Classification

- Making the assumption that all features are conditionally independent given the class reduces the number of parameters

\[ P(x_i | c_i), \quad P(c_i) \]

- Number of parameters is linear in the number of features

- MAP parameters can be computed from the smaller set of parameters

\[ P(x | c) = \prod_i P(x_i | c) \]
Naïve Bayes Classification

- Independence reduces the classifier to
  \[ \hat{c} = \text{argmax}_c P(c \mid x) = \text{argmax}_c P(c) \prod_i P(x_i \mid c) \]

- Learning requires determining the parameters \( P(c_j) \) and \( P(x_i \mid c_j) \) from the data
  - MLE for the parameters can be used

\[
P(c_i) = \frac{\#(x^{(j)}, y^{(j)}) \in D : y^{(j)} = c_i}{|D|}
\]

\[
P(x_i = a \mid c_j) = \frac{\#(x^{(k)}, y^{(k)}) \in D : y^{(k)} = c_j \land x_i^{(k)} = a}{\#(x^{(k)}, y^{(k)}) \in D : y^{(k)} = c_j}
\]
Naïve Bayes Classification

- MLE for the parameters has problems
  - Estimation quality deteriorates when there is limited data
    - Many parameters end up 0 or 1
- Can use MAP estimate instead
  - For binary features the answer is the MAP of a Bernoulli distribution with a Beta distribution prior

\[
P(x_i = a \mid c_j) = \frac{\left( \#(x^{(k)}, y^{(k)}) \in D : y^{(k)} = c_j \land x_i^{(k)} = a \right) + \beta_a - 1}{\left( \#(x^{(k)}, y^{(k)}) \in D : y^{(k)} = c_j \right) + \beta_a + \beta_a - 2}
\]
Naïve Bayes Classification

- For multinomial features MAP corresponds to a multinomial distribution with its conjugate Dirichlet prior
  - This corresponds to Laplace smoothing
    - For the case where we all $\beta$ are the same this gives
      \[
      P(c_i) = \frac{\left(\#(x^{(k)}, y^{(k)}) \in D : y^{(k)} = c_j\right) + \beta_{ci}}{\left(\#(x^{(k)}, y^{(k)}) \in D\right) + \sum_j \beta_{cj}} + \beta
      \]
      \[
      P(x_i = a \mid c_j) = \frac{\left(\#(x^{(k)}, y^{(k)}) \in D : y^{(k)} = c_j \wedge x_i^{(k)} = a\right) + \beta_{xia}}{\left(\#(x^{(k)}, y^{(k)}) \in D : y^{(k)} = c_j\right) + \sum_a \beta_{xib}} + \beta
      \]
  - This is equivalent to adding $\beta$ observations to each count
Naïve Bayes Classification

- Naïve Bayesian classification (and Bayesian classification in general) has additional numeric problems
  - The more features, the smaller the posterior in general
    - More stable to solve Bayesian classification in log likelihood space

\[
\hat{c} = \text{argmax}_c P(c) \prod_i P(x_i \mid c) = \text{argmax}_c \log \left( P(c) \prod_i P(x_i \mid c) \right)
\]

\[
= \text{argmax}_c \left( \log P(c) + \sum_i \log P(x_i \mid c) \right)
\]
Bayesian Classification

- If features are continuous it is no longer possible to represent all conditional probability density values even under the Naïve Bayes assumption
  - Need to introduce parameters to represent the probability density function of every feature given a class
    - Common choice is a Gaussian distribution
      \[ p(x_i = z \mid c_j) = N(z \mid \mu_{i,j}, \sigma_{i,j}) \]
Gaussian Naïve Bayes

- Naïve Bayes criterion stays the same
  \[ \hat{c} = \arg\max_c P(c \mid x) = \arg\max_c P(c) \prod_i p(x_i \mid c) \]

- Need to learn the Gaussian parameters from data
  - Using MLE the expected mean of the Gaussian is the expected value in the data
    \[ \hat{\mu}_{i,j} = E[x_i \mid c_j] = \frac{\sum_{k:y^{(k)}=c_j} x_i^{(k)}}{\#(x^{(k)}, y^{(k)}) : y^{(k)} = c_j} \]
  - Similarly the expected variance is the unbiased variance of the data in the class
    \[ \hat{\sigma}_{i,j}^2 = \frac{\sum_{k:y^{(k)}=c_j} (x_i^{(k)} - \mu_{i,j})^2}{\#(x^{(k)}, y^{(k)}) : y^{(k)} = c_j - 1} \]
Naïve Bayes Classification

- Naïve Bayes is a commonly applied technique for probabilistic classification
  - Naïve Bayes is usually relatively easy to apply
  - Uses relatively intuitive parameters
  - Number of parameters is linear in the number of features
  - Often yields good results
  - Independence assumption is usually not strictly correct
  - Requires significant data to estimate its parameters
  - Makes relatively limiting assumptions for continuous features
Generative vs. Discriminative Learners

- Supervised learning algorithms are often divided into two groups
  - Generative algorithms
  - Discriminative algorithms
- Generative algorithms are algorithms that learn a model for the process that generates the data and use this to predict the output
- Discriminative algorithms directly learn to predict/discriminate the output
Generative vs. Discriminative Learners

- Generative algorithms allow to predict the output and to generate simulated data
  - Learn $p(x \mid y)$ and potentially $p(y)$
  - Use MLE or MAP to predict the output

- Discriminative algorithms directly learn to predict/discriminate the output
  - Learn $p(y \mid x)$ or directly the MAP estimate for $y$
Generative vs. Discriminative Learners

- Both types of learners have their advantages and disadvantages
  - Generative algorithms can be used to produce simulated data
  - Parameters for generative algorithms are often easier to extract from the data
  - Discriminative algorithms often have more compact representations (particularly if they directly learn to predict the MAP estimate)