CS 480 Course Notes

Introduction to Machine Learning

Michael Socha

University of Waterloo
Winter 2019

Contents

1 Course Overview **1**

2 Introduction - What is Machine Learning? **2**
2.1 Learning Frameworks 2
2.2 Challenges 2

3 Decision Trees **3**
3.1 Predictions Using Decision Trees 3
3.2 Encoding Functions in Decision Trees 3
3.3 Training and Testing 3
3.3.1 Information Content 4
3.3.2 Entropy 4
3.4 Overfitting 4
3.5 Inductive Bias 4
3.6 Advantages and Disadvantages 4

4 Evaluation of Learning **6**
4.1 Performance Formulation 6
4.2 Common Learning Challenges 6
4.3 Training vs Validation vs Test Sets 6
4.4 Bias-Variance Decomposition 6
4.5 Cross Validation 7
4.6 Bootstrapping 7
4.7 Avoiding Overfitting 7
4.8 Performance Evaluation of Classifiers 8
4.8.1 Accuracy and Error 8
4.8.2 Precision and Recall 8
4.8.3 F-Measure 8
4.8.4 Sensitivity and Specificity 9

5 Instance-Based Learning **10**
5.1 Parametric vs Non-Parametric Methods for Supervised Learning 10
5.1.1 Approximation 10
5.1.2 Efficiency 10
5.2 K-Nearest Neighbors 10
5.2.1 Implementation 10
5.2.2 Distance Function 10
5.2.3 Decision Boundaries 11
5.2.4 Selection of K 11
5.2.5 Pre-Processing 11
5.2.6 Distance-Weighted Nearest Neighbor 11
5.2.7 High Dimensionality 12

6 Perceptron **13**

6.1 Formulation . . . 13
6.2 Decision Boundary . . . 13
6.3 Learning Algorithm . . . 13
6.3.1 Hyper-parameters . . . 14
6.3.2 Extensions . . . 14
6.4 Convergence . . . 14
6.4.1 Rosenblatt's Perceptron Convergence Theorem . . . 14
6.5 Limitations . . . 15

7 Linear Models **16**
7.1 Formulation . . . 16
7.2 Training Parameters . . . 16
7.2.1 Gradient Descent Solution . . . 16
7.2.2 Closed Form Solution . . . 17
7.3 Regularization . . . 17
7.3.1 P-norm . . . 17
7.3.2 Ridge Regression . . . 18
7.3.3 Lasso Regression . . . 18
7.4 Application to Classification Problems . . . 18

1 Course Overview

This is an applied introductory machine learning course covering the basics of machine learning algorithms and data analysis. Topics covered include:

- Regression analysis
- Probabilistic modeling
- Support vector machines
- Supervised vs unsupervised learning
- Reinforcement learning
- Neural networks

2 Introduction - What is Machine Learning?

Machine learning is the field of study of how computers can improve their performance at tasks (i.e. learn) without being explicitly programmed to do so. Machine learning can be useful for tasks for which it is difficult to write a step-by-step imperative program. Sample applications include optical character recognition, computer vision, and game playing.

2.1 Learning Frameworks

- **Supervised Learning:** Goal is to learn a function based on its input and output (e.g. determining if an email is spam based on a set of emails labeled as spam or not spam).
- **Unsupervised Learning:** Goal is to learn a function based on its input alone (e.g. organizing data into clusters).
- **Reinforcement Learning:** Goal is to learn a sequence of actions that maximize some notion of reward (e.g. learning how to control a vehicle to perform some maneuver).

2.2 Challenges

Some of the challenges facing machine learning today are:

- Dealing with large amounts of data (algorithm complexity and distributed computing become very relevant)
- Generating reproducible results
- Challenges concerning real-world adoption of work (e.g. human computer interaction, robustness, ethical concerns)

3 Decision Trees

Decision trees contain questions as nodes and answers as edges, and serve to guide to an answer based on a set of observations.

3.1 Predictions Using Decision Trees

Consider a data set of employees and their job satisfaction, where we use the data set to predict whether an employee is satisfied with their job. An employee can have many features, such as age, salary, seniority, working hours. It is infeasible to build all possible decision trees when there are many features due to the exponential growth of the tree. Instead, for predictive purposes, it makes sense to focus on those questions that are informative to the prediction. For example, if there is no major difference in job satisfaction based on an employee's age, then it is unnecessary to include age in a decision tree, since the answer is not informative for the prediction.

In general, supervised learning problems have a set of possible inputs X, an unknown target function $f : X \to Y$, and a set of function hypotheses $\{h|h : X \to Y\}$. Supervised learning algorithms accept training examples (a pair (x, y) where $x \in X$, $y \in Y$) and output a function hypotheses that approximates f. When using decision trees for learning, each decision tree is a function hypothesis.

3.2 Encoding Functions in Decision Trees

Boolean functions can be fully expressed in decision trees, with one branch for true and another for false. Other function can be approximated as a boolean function. For example, instead of a node asking what an employee's salary is, it could ask whether the salary is above a certain amount.

3.3 Training and Testing

The key idea behind decision tree generation algorithms is to grow a tree until it correctly classifies all training examples. The rough procedure followed is:

1. If all training data has the same class, create a leaf node and return.
2. Else, create the best (i.e. most informative) node on which the split the data.
3. Split the training set over the above node.
4. Continue the procedure on each subset of training data generated.

3.3.1 Information Content

Criteria for finding the "best" split of data can be defined mathematically. If event E occurs with probability $P(E)$, then when E occurs, we receive $I(E) = \log_2 \frac{1}{P(E)}$ bits of information. This can be interpreted as that less likely events yield more information.

3.3.2 Entropy

An information source S which emits results $s_1, s_2, ..., s_i$ with probabilities $p_1, p_2, ..., p_i$ produces information at $H(S) = \sum_i^k p_i I(s_i)$. $H(S)$ is known as the information entropy of S. Information entropy can vary between 0, which indicates no uncertainty (i.e. all members of S in same class), and 1, which indicates high uncertainty (i.e. equal probability of all classes). The best split of data for classification is one that maximally decreases its entropy (a concept known as information gain).

3.4 Overfitting

A hypothesis $h_1 \in H$ is said to overfit training data if there is some alternative hypothesis $h_2 \in H$ that has a larger error over the training data but a smaller error over a larger set of inputs. Overfitting can occur due to errors in a data set or just due to coincidental irregularities (especially in a small dataset). Overfitting can be avoided by removing nodes with low information gain, either by stopping decision tree construction early or by pruning such nodes after the tree is constructed.

3.5 Inductive Bias

Inductive bias refers to the assumptions made about the target function to predict future outputs. Common inductive biases for decision trees are:

- Assumption that simplest hypothesis is the best (i.e. Occam's razor).
- Decision trees with information gain closer to the root are considered better.

These are examples of preference bias, which influence the ordering of the hypothesis space. This is distinct from restriction bias, which limits the hypothesis space.

3.6 Advantages and Disadvantages

Decision trees are good for:

- Ease of interpretation
- Speed of learning

Limitations of decision trees include:

- High sensitivity, with tree output changing significantly due to small changes in input.
- Not good for learning data sets without axis-orthogonal (i.e. constant in at least 1 dimension) decision boundaries.

4 Evaluation of Learning

4.1 Performance Formulation

Let $\hat{y}$ be an output generated by a function f approximating some target function. Let y be the corresponding output of the target function. A loss function $l(y, \hat{y})$ can be used to measure the accuracy of the approximation function f. Some common loss functions include:

- Squared Loss: $l(y, \hat{y}) = (y - \hat{y})^2$
- Absolute Loss: $l(y, \hat{y}) = |y - \hat{y}|$
- Zero/One Loss: $l(y, \hat{y}) = 1_{y \neq \hat{y}}$

We assume that the data coming from our target function comes from some probability distribution D, and that our training data is a random sample of (x, y) pairs from D. A Bayes Optimal Classifier is a classifier that for any input x, returns the y most likely to be generated by D.

Based on the available training data, the goal of supervised learning is to find a mapping f from x to y such that generalization error $\sum_{(x,y)} D(x, y)l(y, f(x))$ is minimized. However, since D is unknown, we instead estimate the error from the average error in our training or test data, which is $\frac{1}{N} \sum_{n=1}^{N} l(y_n, f(x_n))$.

4.2 Common Learning Challenges

Some common challenges in learning include:

- Inductive bias of the algorithm being distant from the concept actually being learned.
- The data itself having challenging characteristics, such as lots of noise, ambiguity, or missing information.

4.3 Training vs Validation vs Test Sets

Models are initially built based on a training dataset. Test sets (also known as holdout sets) are then used to estimate the generalization error. Validation sets are also used to measure the model's performance, but unlike test sets, validation sets can make changes to the model's parameters.

4.4 Bias-Variance Decomposition

Many machine learning algorithms are based on building a formal model based on the training data (e.g. a decision tree). Models have parameters, which are characteristics that can help

in classification (e.g. a node in a decision tree). Models may also have hyper-parameters, which in turn control other parameters in a model (e.g. max height of decision tree).

Generalization errors result from a combination of noise, variance, and bias. Bias concerns how well the type of model fits the data. Models with high bias pay little attention to training data and suffer from underfitting, while models with low bias may pay too much attention to training data and become overfitted. Bias and variance tend to be at odds with one another (high bias typically leads to low variance, and vice versa). For example, a decision tree that makes the same prediction for all input has high bias and low variance, while a decision tree trained to return a correct prediction for each point of training data will have low bias and likely high variance.

4.5 Cross Validation

Cross validation is a technique for measuring how well a model generalizes. The idea behind it is to break up a training data set into K equally sized partitions, and use $K-1$ of the partitions as training data and the remaining partition for testing. This should be repeated K times, so that all points of data are at some point used for testing. Higher values of K lower the amount of variance of in the error estimation. To avoid training and testing data having a different probability distribution, the data should be shuffled before being split.

4.6 Bootstrapping

Bootstrapping is an alternative to cross validation where instead of dividing a training data set into partitions, a random sample of points (with possible duplicates) is used as training data. The remaining points are then used as testing data, with the goal being similar to that of cross validation.

4.7 Avoiding Overfitting

Overfitting often occurs when training data has many features, which allows for an approximation of the target function with many degrees of freedom. Overfitting can be avoided by only considering features with a strong correlation to the output. Cross-validation could be applied as follows:

1. Divide training data into K groups at random.
2. Within each group, find a small set of features with strong correlation to the output. Running this step for each group individually instead of for the entire training set further helps avoid overfitting.
3. Build a classifier using the features and examples from the $K-1$ groups.
4. Use the classifier to predict the examples and in group K and measure the error.

5. Repeat 2-4 to produce an overall cross-validation estimate of the error.

Bootstrapping can be applied in a similar way to avoid overfitting.

4.8 Performance Evaluation of Classifiers

Consider the following terminology for classification problems:

- True positive (TP) - Examples of class 1 predicted as class 1
- False positive (FP) - Examples of class 0 predicted as class 1 (Type 1 Error)
- True negative (TN) - Examples of class 0 predicted as class 0
- False negative (FN) - Examples of class 1 predicted as class 0 (Type 2 Error)

4.8.1 Accuracy and Error

The following formulas can be used to measure accuracy and error:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$ErrorRate = \frac{FP + FN}{TP + TN + FP + FN}$$

4.8.2 Precision and Recall

Precision and recall can be measured as follows:

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

Precision measures the ratio of positive predictions that were correct, while recall measures the ratio of total positive instances that were predicted. Similarly to how variance and bias are often at odds with one another, so are precision and recall.

4.8.3 F-Measure

An F-measure (also known as a F1 score) measures a model's accuracy by taking into account both precision and recall as follows:

$$F = \frac{2PR}{P + R}$$

To adjust the relative importance of precision vs recall, a weighted F-measure can be used, which is defined as follows:

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

In a standard F-measure, $\beta = 1$, $\beta < 1$ means that precision is valued over recall, while $\beta > 1$ means recall is valued over precision.

4.8.4 Sensitivity and Specificity

Sensitivity is the same measure as recall. Specificity is a measure of how well a classifier avoids false positives, and is measured as:

$$Specificity = \frac{TN}{TN + FP}$$

5 Instance-Based Learning

5.1 Parametric vs Non-Parametric Methods for Supervised Learning

Datasets can be represented as a set of points in a high-dimensional space; a data point with n features $f_1, f_2, ...f_n$ can be represented with the feature vector $(f_1, f_2, ..., f_n)$ in n-dimensional space. Parametric methods of supervised learning attempt to model the data using these parameters, while non-parametric (also known as instance-based) methods do not.

5.1.1 Approximation

Parametric methods use parameters to create global approximations. Non-parametric methods instead create approximations based on local data.

5.1.2 Efficiency

Parametric methods do most of their computation beforehand, and the summarize their results in a set of parameters. Non-parametric methods tend to have a shorter training time but a longer query answering time.

5.2 K-Nearest Neighbors

K-nearest neighbors (KNN) is a common non-parametric method. The idea is to predict the value of a new point based on the values of the K most similar (i.e. closest) points.

5.2.1 Implementation

A common implementation of KNN involves looping through all N points in a training set and computing their distance to some point x. Then the K nearest points are selected. This process can be sped up by storing the data points in a data structure that helps facilitate distance-based search (e.g. a k-d tree).

5.2.2 Distance Function

"Nearby" means of minimal distance, which is commonly defined by Euclidean distance. Other distance functions $d(x, x')$ can be used, though must meet the following conditions:

- $d(x, x') = d(x', x)$ (i.e. symmetric)

- $d(x, x) = 0$ (i.e. definite)
- $d(a, c) \leq d(a, b) + d(b, c)$ (i.e. triangle inequality holds)

5.2.3 Decision Boundaries

Decision boundaries define the borders of a single classification of input. These boundaries are formed of sections of straight lines that are equidistant to two points of different classes. A highly jagged line is an indicator of overfitting, while a simple line is an indicator of underfitting.

5.2.4 Selection of K

The selection of the value of K is a bias-variance tradeoff. Low values of K have high variance but low bias, while high values of K have low variance but high bias. High-values of K result in smoother decision boundaries, which can be a sign of underfitting, and vice versa.

K can be selected experimentally by evaluating the performance for different values of K through cross-validation or against a testing set. In theory, as the number of training examples approaches infinity, the error rate of a 1NN classifier is at worst twice that of the Bayes Optimal Classifier.

5.2.5 Pre-Processing

Some common forms of pre-processing for KNN include:

- Removing undesirable inputs. Common removal methods are:
  - Editing methods, which involve eliminating noisy points of data.
  - Condensation methods, which involve selecting a subset of data that produces the same or very similar classifications.
- Use custom weights for each feature (not all features may be equally relevant for the situation)

5.2.6 Distance-Weighted Nearest Neighbor

A common problem with KNN is that it can be sensitive to small changes in the training data. One way to mitigate with drawback is to compute a weight for each neighbor based on its distance (e.g. through a Gaussian distribution), and this weight determines how much of an influence that point's value has. This differs from standard KNN which weighs the values of the K nearest neighbors equally and ignores all other values.

5.2.7 High Dimensionality

In uniformly distributed high-dimensional spaces, distances between points tend to be roughly equal, since there are so many features that changing a few features results in only a small change in distance. However, KNN can still be applied in practice for high-dimensional spaces, since data in high-dimensional spaces tends to be concentrated around certain hubs rather than uniformly distributed.

6 Perceptron

A perceptron is a binary classification algorithm that accepts a set of features associated with weights.

6.1 Formulation

Let $x = (x_1, x_2, ..., x_D)$ be a feature vector with D features, let $w = (w_1, w_2, ..., w_D)$ be the corresponding weight vector, and let b be some fixed bias/threshold term. Activation a is defined as:

$$a = w^T x + b = \sum_{d=1}^{D} w_d x_d + b$$

If $a \geq 0$, then the perceptron classifier predicts a positive classification, and otherwise it predicts a negative classification.

6.2 Decision Boundary

A perceptron decision boundary is where the sign of the activation changes from negative to non-negative. It can be expressed as $a = w^T x + b = 0$, so $w^T x = -b$. Hence, the decision boundary is a hyperplane perpendicular to w. The bias term can shift the hyperplane, but does not change its alignment.

If the weight and bias terms are scaled together, then the decision boundary does not change. However, it is common to normalize the weight vector to have a magnitude of 1.

6.3 Learning Algorithm

The perceptron learning algorithm starts with any (e.g random) w and b parameters, and iterates over each point training point as follows:

1. Compute the activation a for the point.
2. Compute the classification y based on the activation a (i.e. 1 is $a \geq 0$, otherwise 0).
3. If the perceptron correctly classifies the point, continue.
4. Otherwise, update the weights as $w_d = w_d + yx_d$, and update the bias as $b = b + y$. This guarantees that this point is correctly classified for this iteration.

The perceptron algorithm is guaranteed to converge on parameters (i.e. weights and bias) that correctly classify a set of points if such parameters exist.

The algorithm is considered to be:

- **Online**, since data becomes available in sequential order instead of all at once.
- **Error driven**, since it requires an incorrectly classified point to update its parameters.

6.3.1 Hyper-parameters

The only hyper-parameter of the perceptron algorithm is the maximum number of iterations of the above algorithm (i.e. the number of times each point is considered). A high number of iterations can lead to overfitting, while a low number of iterations can lead to underfitting. Since the order in which training points are encountered matters in the training algorithm, it is important to permute the order of examples between iterations.

6.3.2 Extensions

A potential problem with the perceptron algorithm described above is that later training points are weighted more than earlier ones. For instance, it is possible that after selecting weights that correctly classify thousands of training points, a single mislabeled point updates the weights in a way that drastically changes the decision boundary in order to correctly classify this new point, but in doing so, mislabels many training points that were previously correctly classified.

This problem can be mitigated by favoring weight vectors that "survive" a long time. This can be done by letting the activation a use a weighted sum of weight vectors, each of which in term is weighted by its survival time (i.e. voted perceptron algorithm). Alternatively, a single weight vector and bias term can be tracked using the weighted average of their survival times (i.e. average perceptron algorithm).

6.4 Convergence

Data can only be classified by a perceptron if it is linearly separable. Otherwise, the perceptron algorithm will not converge, and the decision boundary will oscillate.

The speed of convergence of the perceptron algorithm depends on its margin, which is the distance from the separating hyperplane to the nearest data point. Large margins result in faster convergence, since there is more "wiggle room" to adjust the parameters defining the hyperplane.

6.4.1 Rosenblatt's Perceptron Convergence Theorem

Let the margin γ of a linearly separable dataset D be the largest attainable margin on that dataset. Rosenblatt's perceptron convergence theorem states that if $||x|| < 1$ for all $x \in D$, then the perceptron algorithm will converge after at most $\frac{1}{\gamma^2}$ updates. This is an upper bound; the number of updates also depends on the dataset and the initial parameters.

6.5 Limitations

Two key limitations of perceptron classifiers are:

- The generated solutions (i.e. separating hyperplanes) may not be unique.
- Non linearly-separable data cannot be modeled. In some cases, this limitation can be overcome by transforming feature vectors so they become linearly separable, or by combining multiple perceptrons, which create a network that might be able to learn non-linear boundaries.

7 Linear Models

7.1 Formulation

A loss function concerning the distance d between the value of points x predicted by some model w, b and their actual values y can be minimized as follows:

$$min_{w,b} \sum_n d(y_n, w^T x_n + b) + \lambda R(w, b)$$

The first term measures the training error, while the second term is known as a regularizer, and exists to prevent overfitting. λ is a constant hyper-parameter while R is a function that imposes some penalty based on the complexity of parameters w, b.

7.2 Training Parameters

7.2.1 Gradient Descent Solution

The gradient of error can be used to determine the direction of steepest decline in error, and the training algorithm then adjusts parameters to move in that direction. The training algorithm maintains the state of some parameter z, finds its gradient g, and updates z to be $z + \eta g$, where η is the learning rate (i.e. the size of each step).

As an example, consider a loss function:

$$L(w, b) = \sum_n exp(-y_n(w^T x_n + b)) + \frac{\lambda}{2}||w||^2$$

The partial derivative of L over b is $-\sum_n y_n \exp(-y_n(w^T x_n + b))$, so b is adjusted in each step to $b - \eta \sum_n y_n \exp(-y_n(w^T x_n + b))$.

The partial derivative of L over w is $(-\sum_n y_n x_n \exp(-y_n(w^T x_n + b))) + \lambda w$, so w is adjusted in each step to $w - \eta((-\sum_n y_n x_n \exp(-y_n(w^T x_n + b))) + \lambda w)$.

The learning rate η impacts the size of each step. A high learning rate can result in oscillations around local minima, while a low learning rate results in training being slow. It is common to lower the learning rate during later steps. When parameters start to not change by much, then a local minimum has likely been approached, and the algorithm can stop.

Note that gradient descent does not guarantee convergence to a global minima; it can get stuck in a local minima that may not be optimal. However, approximations using linear functions have a single global minimum.

7.2.2 Closed Form Solution

The gradient descent solution iteratively approaches some minimum. On the other hand, a closed form solution provides a formula for the absolute minimum. A closed form solution is not always attainable, but this section focuses on an attainable case, which is where the loss function is measured by Euclidean distance.

Our model can start as linear; we assume that the target function can be modeled as $f_w(x) = w_0 + w_1x_1 + w_2x_2 + ... + w_Dx_D$. A linear model is often a poor approximation, so further input variables can be added based on $x_1, ..., x_D$, such as:

- Transformations of variables, such as $\log x_i$
- Interaction terms, such as $x_i \cdot x_{i+1}$
- Numeric encoding of qualitative variables

The goal of the closed-form solution is to minimize $\sum_{i=1}^{n}(y_i - w^T x_i)^2$ (i.e. minimize Euclidean distance of errors). It is convenient to rewrite the problem in matrix format where:

- X is a NxD training data matrix, where $X_{n,d}$ is the value of feature d on example n.
- w is a size D column vector storing weights.
- $\hat{Y} = Xw$ are the predicted values.

Since we aim to minimize squared error, we aim to minimize $||\hat{Y} - Y||^2 + \lambda||w||^2$. Taking the derivative of this against w and setting it to 0 results in $w = (X^TX + \lambda I_D)^{-1}X^TY$. This equation takes $\theta(D^3 + D^2N + DN)$ steps to compute, so this method can be computationally expensive for very high-dimensional problems.

7.3 Regularization

The Gauss-Markov theorem states that in a linear regression model, such as the one described above, the least-squares estimate (i.e. Euclidean distance) results in a model with the least bias. We nonetheless add a regularization term, which increases bias, to avoid overfitting. A few desirable features of a regularization function are:

- It is convex
- Small (even 0) weight vector components are preferred (note that this is a form of inductive bias).

7.3.1 P-norm

The p-norm of the weight vector can be used as a regularization function, defined as follows:

$$R(w) = (\sum_d |w_d|^p)^{\frac{1}{p}}$$

The p-2 norm is the same as the Euclidean norm used above.

7.3.2 Ridge Regression

Ridge regression uses the following regularization function:

$$R(w) = \sum_d w_d^2$$

Ridge regression tends to give a smooth solution with low weights.

7.3.3 Lasso Regression

Lasso regression uses the following regularization function:

$$R(w) = \sum_d |w_d|$$

Lasso regression leads to a less smooth and more computationally expensive solution than ridge regression. It is equivalent to a p-1 norm.

7.4 Application to Classification Problems

The perceptron algorithm covered above can find a hyperplane that splits a linearly separable dataset. However, the perceptron algorithm does not converge for datasets that are not linearly separable.

Due to this limitation, it can make sense to relax the problem to finding a hyperplane that splits data with the fewest possible classification errors. However, due to the discrete nature of 0/1 error, it is NP hard to minimize. Thus, 0/1 classification functions are commonly modeled with continuous approximations. One common class of approximations are convex surrogate loss functions, which extend between 0 and 1 and are convex, which is a property that makes them easy to minimize.