STA561: Probabilistic machine learning

Review of methods and validation approaches (11/13/13)

Lecturer: Barbara Engelhardt *Scribes: Scribe names*

1 Overview

Today we are going to review the machine learning approaches we have studied in this course in the context of the problem we are addressing. We will also review general approaches to testing and validating our models and methods. Each of the references to book sections refer to the main textbook: Murphy's *Machine Learning, a Probabilistic Perspective*. Unless otherwise referenced, much of the information in this review comes from this text.

In particular, let us consider the methods we have developed to address the following problem categories:

- Clustering
- Classification
- Prediction (regression)
- Dimensionality reduction

We will also consider various approaches to evaluating models for data and quantifying results from a model (cross validation, bootstrap, and loss functions). We will not review (for a lack of time) directed or undirected graphical models, inference methods (exact, approximate), underlying statistics and probability distributions, exponential family, model selection, or optimization approaches for parameter estimation.

2 Clustering methods

The general formulation of a clustering problem is as follows: given a set of measurements for p features for n samples, where $x_i \in \Re^p$ for $i = 1, \dots, n$, cluster these features into K groups. The training data for fitting a clustering method is $\mathcal{D} = \{x_1, x_2, \dots, x_n\}$, where x_i is a p-vector of feature values for sample i. Then, for a new sample x^*, the goal is to determine the cluster for this sample in the form of a categorical variable.

Soft clustering gives the probability of a new sample belonging to one of the K clusters (where the sum of the probability over all K clusters, or $K+1$ clusters in the infinite mixture model case, sum to one), whereas *hard clustering* assigns the new sample to a single cluster.

Clustering is 'the canonical unsupervised problem' because we are exploring the data with no idea of what the underlying structure looks like. The clustering problem can often be thought of as a density estimation problem. We assume that the data can be explained by a latent variable (i.e., the cluster) and we then estimate, for each cluster, what the samples coming from that cluster will look like, or $p(x_i|z_i = k)$.

2.1 K-means clustering (11.4.2.5)

Given K underlying clusters, this method, which is not based on a probabilistic model, but instead on a given distance function between pairs of points, describes each data point as being associated with one of K cluster centroids. Parameter estimation proceeds by iteratively making a hard assignment of each data point to a cluster (the cluster that minimizes the distance from the data point to the centroid) and re-estimating the centroids for each cluster given the cluster assignments.

2.2 Gaussian mixture model (11.2.1)

Given K underlying mixture components, each with probability π_k, a Gaussian mixture model describes each sample x_i as being generated from one of the K components. The mixture components in the Gaussian case are distributed as a normal distribution, with cluster-specific mean and variance parameters. More generally, mixture models may emit samples from an arbitrary distribution.

2.3 Infinite mixture model (25.2.3)

A Dirichlet process prior is included on the mixture components, allowing the posterior distribution over the number of parameters to be sampled given the data.

2.4 Clustering pitfalls

Label switching. (11.2.3)

3 Classification methods

The underlying problem for classification is, given a sample x_i, determine which of a set of classes x_i belongs to. The training data for fitting a classifier is $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where x_i is a p-vector of feature values for sample i and y_i is a class label (a categorical variable). The main goal in classification is to make predictions for new inputs or samples, which is the *generalization* error (discussed further below).

As in clustering, there is distinction between a *soft classification*, where, for a new sample, the model produces the probability of belonging to each of the K classes, versus *hard classification*, where the method produces an assignment to a single class. Furthermore, there is a distinction between *binary classification*, or $K = 2$, where the class variable may be modeled as a Bernoulli variable, and multi-class classification, or $K > 2$.

3.1 Logistic regression for binary classification (8.2)

Logistic regression models the response variable in linear regression model as a Bernoulli variable, squishing it to be between 0 and 1. It is a specific instance of a generalized linear model that can be used for binary classification.

There are extensions of logistic regression to multi-way classification (8.3.7).

3.2 K-nearest neighbors (KNN) (1.4.2)

The KNN classifier identifies the K nearest neighbors of the new sample x^* according to a given distance function, and classifies this sample according to the majority vote of the nearest neighbors. Given a data set, this model does not have parameters to fit. We also discussed *kernelized KNN*, where the distance between pairs of points is evaluated according to the kernel function $\kappa(x, x')$.

3.3 Naive Bayes classifier (10.2.1)

For class y and features x, a Naive Bayes classifier models $p(y|x) \propto p(x|y)p(y) = p(y) \prod_j p(x_i|y)$, or the probability of each class times the probability of each feature being generated from that class (independently).

3.4 Gaussian process classifiers (15.3)

GP classification is similar to logistic regression: the response from GP regression is passed through a function that ensures the value is between 0 and 1 (e.g., the logistic function).

3.5 Support vector machines (SVM) (14.5.2)

These are methods that maximize the margin between the two classes of data using the hinge loss function, using a sparse number of *support vectors*, or samples on the convex hull of each category that constrain this margin. These samples may be projected up to a high dimensional space using kernel functions; the implicit linear separator may better distinguish the classes in this high dimensional space.

3.6 Decision trees/random forests (16.2.5)

Adaptive basis functions are similar to kernels but are learned directly from the data. Decision trees recursively partition the input space and define a local model in each region of this space. Random forests are an *ensemble* classifier, or a collection of trees estimated from bagged versions (i.e., subsamples with replacement) of the data, that collectively vote on the classification.

3.7 Perceptrons and Neural Networks (28.3)

Perceptrons combine multiple weighted inputs into a non-linear function to get an output. Neural networks take an input sample, and project this sample through a number of network layers of hidden variable nodes of this form, each with specific edges and functions, to produce an output classification. The weights in these models are estimated using an algorithm known as *backpropagation*. The area of deep learning has begun to build these type of networks with a very large number of layers.

4 Prediction (regression) methods

The regression problem is similar to the classification problem – we are learning a mapping from an input x_i to an output y_i – but the output, or response, is continuous instead of categorical.

4.1 Linear regression (7.2)

Given data $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $y_i \in \Re$ and $\mathbf{X} \in \Re^{n \times p}$, define (multivariate) linear regression as follows:

$$\begin{aligned} \mathbf{y} &= \mathbf{X}\beta + \epsilon && (1) \\ \epsilon &= \mathcal{N}(0, \sigma^2). && (2) \end{aligned}$$

The goal of linear regression is to find $\beta \in \Re^p$ in order to predict the response y from a given sample x. This implies that the response vector is distributed according to a multivariate normal of dimension n: $p(\mathbf{y}|\mathbf{X}, \beta) = \mathcal{N}_n(\mathbf{X}\beta, diag(\sigma^2))$.

4.2 Generalized linear models (9.3)

Given the same setup as for linear regression, GLMs model the response function (or, equivalently, the residual errors) as coming from a non-Gaussian distribution. In class, we discussed logistic regression and Poisson regression.

4.3 Penalized regression

When $p \gg n$, the system is underconstrained, and there are many optimal solutions to the regression problem of finding the optimal value of the coefficients β. In this setting, we might choose to perform penalized regression, which shrinks the values of the coefficients towards zero (or another fixed number).

4.3.1 Bayesian linear regression (7.6)

Returning to the setup of linear regression, Bayesian linear regression includes a prior on the coefficients β and the residual variance term σ^2. If we were to choose priors allowing conjugacy, we would select a normal distribution for the β coefficients and an inverse gamma distribution for the σ^2 variables. One might choose

4.4 Ridge regression (7.5)

Ridge regression is L_2 penalized regression, and is equivalent to a normal prior on the β variables in Bayesian linear regression. There is a closed form solution for the β parameters.

4.4.1 Lasso (ℓ_1 regularized) regression (13.3)

Lasso regression puts an ℓ_1 penalty on the β coefficients, which induces sparsity (i.e., zeros in the β vector that correspond to removing that predictor from the regression model).

4.4.2 Bayesian spike-and-slab regression (13.2.1)

The canonical two-groups sparsity-inducing prior in Bayesian literature, this prior (that can be placed on the regression coefficients β) is a mixture of a point mass at 0 and a Gaussian distribution, mixing over shrinkage to zero of noise and ℓ_2 regularization of signal in the coefficients.

4.4.3 Automatic relevance determination (13.7)

The ARD prior puts a zero-mean normal on each regression coefficient β, where the variance is a parameter τ_j^2 with an inverse gamma distribution that is estimated for each coefficient separately. This local prior has ℓ_1 properties, and induces sparsity in the coefficients.

4.4.4 Greedy methods, e.g., Forward stepwise regression (13.2.3.1)

These methods for sparse regression iteratively include, in a linear model, the single coefficient that produces the greatest improvement in model score (e.g., BIC, MDL).

4.4.5 Gaussian process regression (15.2)

A Gaussian process is a distribution over functions $f(\cdot)$, where we assume that the joint probability of the function output at every point x_i, $p(f(x_1), \ldots, f(x_n))$ is jointly Gaussian. The covariance matrix of this multivariate Gaussian $\sigma_{i,j} = \kappa(x_i, x_j)$ is a Mercer kernel, producing a positive definite Gram matrix that captures the covariance between the output of the function at those different points. Normal error around each of these points is assumed for Gaussian process regression.

4.4.6 Classification and Regression Trees (16.2)

Regression based on decision trees and random forests.

5 Dimensionality reduction

Dimensionality reduction considers a matrix $X \in \Re^{n \times p}$, and models this matrix as a product of two low dimensional matrices. The idea is that the data live in a low dimensional manifold despite having a high dimensional form, and these methods serve to project the data down to this low dimensional space, and also to describe this space. This problem firmly falls in the realm of *unsupervised learning*, as the goal is the discovery of interesting latent structure in the data, which is usually unknown beforehand. Correspondingly, the results are often very sensitive to the data input and not robust (see Bootstrap methods to evaluate this sensitivity), and it is difficult to evaluate the results because there is normally no ground truth.

5.1 Principal component analysis (12.2)

Uses an orthogonal transformation of a set of vectors into a set of uncorrelated values called principal components. The number of PCs is, by definition, less than or equal to the number of underlying vectors, and is equal to the rank of the matrix. The interpretation of the PCs is that the first PC will explain the greatest amount of variance in the original matrix, followed by the second PC, etc. [`http://en.wikipedia.org/wiki/Principal_`
Although there are probabilistic versions of PCA, the basic method can be understood in terms of the singular value decomposition of the matrix X, or the eigenvalue decomposition of the empirical covariance matrix $X^T X$.

PCA is commonly used in exploratory data analysis. If you find yourself with a matrix of data, it is often quite valuable to find the PCs of these samples and plot the first few PCs against each other to see if there is, in fact, clustering in the data. While these plots are purely exploratory, they often reveal interesting and unexpected structure in the data.

5.2 Factor analysis (12.1.1)

The idea behind factor analysis is that row i of matrix X has the following distribution: $p(x_i|z_i, \theta) = \mathcal{N}_p(\Lambda z_i, \Psi)$, where $\Psi = diag(\psi_1, \ldots, \psi_p)$. In other words, each sample in the matrix is the linear combination of a factor loading vector Λ and a sample-specific factor z_i, with diagonal covariance matrix Ψ.

5.3 Latent Dirichlet allocation/topic models (27.3)

Can be interpreted as a solution to dimensionality reduction. For a matrix of n documents and p word counts, we can fit a LDA model to produce a $n \times K$ matrix that captures the proportion of words in the document that are predicted to come from topic k, and a $K \times p$ matrix that includes the probability of any word being generated from topic k (according to a multinomial distribution, so the kth row of this matrix will sum to one as the parameters of this distribution).

6 Time series data

These models may be applied when we have noisy observations over a time period (or over a space), and we would like to infer the latent variables in that space considering the values of the neighboring latent variables in addition to the observations at each point.

6.1 Hidden Markov models (17.3)

A first order HMM has observations for each state with an arbitrary distribution; the goal is to infer the categorical latent states that have a Markov dependency. Time is discrete in this model. When the latent states are available for a subset of the data, the model parameters (specifically, the transition probabilities and emission probabilities) may be inferred directly from these observed states.

6.2 State space model (e.g., Kalman filters) (18.1)

A state space model is identical to an HMM, except that the latent states are not categorical but continuous variables. Time is still discrete.

7 Validation methods

7.1 Cross validation (6.5.3)

Cross validation enables an estimation of the *generalization error*, or the performance of your method on data that was not used to train (or fit) the model, using a single data set. In general, generalization error is used to quantify the performance of a model for data analysis: if the model overfits the training data, or if the model is to general to capture the complexities of the data, then this will be reflected in a poor generalization error. The exact quantification of 'error' or 'loss' can be chosen in a problem-specific way, and some examples of common loss functions are included below.

If we have $n = |\mathcal{D}|$ samples in the data set, we split these samples into K partitions at random of approximately equal size; denote the kth partition by $\mathcal{D}_k$, and all the data except for the kth partition $\mathcal{D}_{\neg k}$. Let $F(\mathcal{D}, h)$ be a function for fitting a model that takes a training data set $\mathcal{D}$ and a set of parameters/hyperparameters h (e.g., the number of clusters, the penalty term for regularization), and returns a set of estimated parameter values $\hat{\theta}_{\mathcal{D},h}$. Let $\hat{y} = P(x, \hat{\theta}_{\mathcal{D},h})$ be a function that, given a sample x and a fitted model, returns a prediction $\hat{y}$ for that sample. Finally, define the *loss* in accuracy of predicting $\hat{y}_i$ when the actual value is y_i by $L(y_i, \hat{y}_i)$ (again, it is your choice on how to define this loss; see below). Then, we can define the K-fold cross validation risk, a value that quantifies the ability of a model to generalize to unobserved samples, as follows:

$$R(\mathcal{D}, h, K) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in \mathcal{D}_k} L(y_i, P(x_i, \hat{\theta}_{\mathcal{D}_{\neg k},h})).$$

In words, this is the average loss when comparing, across all data *folds* the predicted values for sample x_i from a model fitted with $\mathcal{D}_{\neg k}$, the training data for that fold.

There are many variants of cross validation, including *leave one out cross validation* (LOOCV) where each sample is held out one-by-one and the model is fit with the remaining data. This requires fitting the model n times, once for every point in the data (i.e., n-fold cross validation).

We can use cross-validation to set parameter values for a model. For example, we can select a set of possible values for a parameter, $\lambda \in \{0.1, 1, 10, 100\}$. For each value of the parameter, we can compute the K-fold cross validation risk (for the same data folds, to be consistent). The value of the parameter may be set to the value for which this risk is minimized. This approach becomes infeasible when there are more than one or two parameters (because the grid grows exponentially quickly). Also, evaluation of this fitted model then should ideally take place on a further held out data set so that the data used to select the parameter values is not used again for validation.

7.2 Bootstrap (6.2.1)

The bootstrap is a Monte Carlo technique for approximating the sampling distribution, which is related to (but not quite the same as) the posterior distribution of the model parameters. The idea is that if we knew the true model parameters, θ^*, we could generate a large number of data sets from the 'true' distribution $x_i^s \sim p(\cdot|\theta^*)$, for $i = 1 : n, s = 1 : S$. We could then fit our model using each of these sampled data sets, $\hat{\theta}_s = F(x^s, h)$, and look at the empirical distribution of our estimated $\hat{\theta}_i$ values to estimate the *sampling distribution*. In other words, from a hypothetical data set of size n, what is the corresponding distribution over possible parameters that we estimate using these data.

Because θ^* is unknown, we have two approaches. The *parametric bootstrap* simulates data sets using parameter $\hat{\theta} = F(\mathcal{D})$, or the estimated parameter from the actual data. the *nonparametric bootstrap* samples n x_i^s with replacement from the original data set $\mathcal{D}$ S times, and then fits the model as before. The reason we are interested in this is because there is naturally a discrepancy between the sampling distribution of $\hat{\theta}_s$, which is a function of the data set, and samples from the posterior distribution $\theta_s \sim p(\cdot|\mathcal{D})$. When the prior is weak, they can be quite similar, but it is possible to construct cases where they are substantially. The bootstrap has been called the 'poor man's posterior', because it is simple to do in the case that it is difficult to sample from the posterior of a model directly.

8 Metrics for evaluating each of the methods

We can define a loss function to evaluate our 'loss' when using our fitted model to classify or predict based on a new sample as compared to the actual class or response value associated with that sample; these approaches, where we are predicting a mapping from input x to output y, where some y are observed, are referred to as *supervised* methods. Clustering, dimensionality reduction, and other tasks that do not have available 'ground truth' (often called *unsupervised* methods, because the goal is to find patterns in the data) require other types of metrics to evaluate their performance.

- Clustering metrics
  - F-measure
  - Jaccard index
- Classification metrics
  - 0-1 loss function
  - hinge loss
- Prediction metrics
  - Mean squared error (MSE)
  - Coefficient of determination (r^2)
  - Log loss, exponential loss
  - L_p loss, or loss based on the L_p norm: $||\mathbf{y} - \hat{\mathbf{y}}||_p = (|y_1 - \hat{y}_1|^p + \ldots + |y_n - \hat{y}_n|^p)^{\frac{1}{p}}$
  - Root mean squared error (RMSE); equivalent to L_2 loss.
- Dimensionality reduction
  - Percentage of variance explained (PVE)

9 General thoughts about machine learning

"All models are wrong, but some models are useful." – George Box 1987.

When considering the best methods to apply to a data set and a well-posed question regarding that data set, you might ask yourself the following questions.

- What data do I have? How can I translate these data into a well defined quantitative question? How can features be processed or transformed so that the model captures the features well?
- What is the simplest model that will address this problem? How can I extend and adapt this model to improve performance? How can I evaluate this model?
- What is the dimension of my data and of my model? Do I have a large sample size relative to the model, or am I in a sparse data scenario?
- Are there missing data (e.g., not all users rated all of the movies)? Are these data missing at random, or is there missingness systematic? Can I afford to throw out samples with missing data, or should I ignore or explicitly model the missingness when I perform inference, or can I try to *impute*, or 'fill in' the missing values and then perform inference?

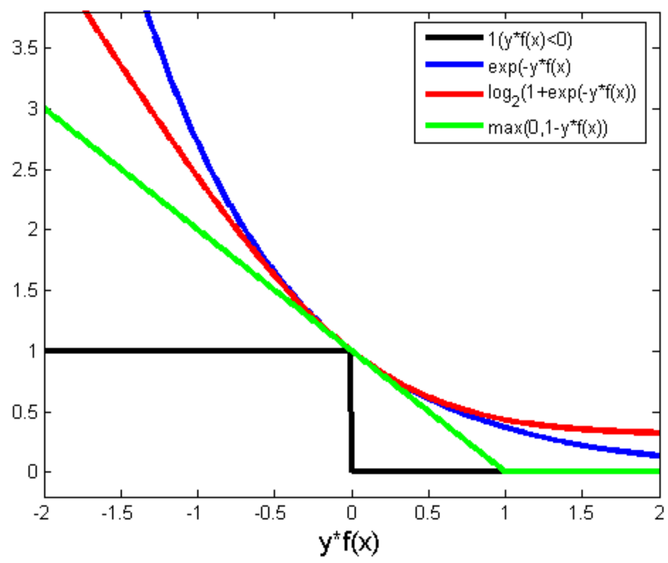

Figure 1: Four different loss functions. Black: 0-1 loss. Blue: exponential loss. Red: log loss. Green: hinge loss. Figure stolen from `http://stats.stackexchange.com/questions/74019/comparing-different-types-of-losses-as-functions-of-l`

- Do I know the answer to individually posed questions? In other words, can I hand classify samples (e.g., a website is about news, or graduate students, or seminars, or a picture contains a car, a person, or a cat)? Can I transform my unsupervised problem into a supervised (or semi-supervised) problem?
- How will I fit my model? Is a point estimate of the parameters reasonable, or would a posterior distribution be more useful? If I choose exact methods, how flexible are they to changes in the model? If I choose iterative methods, how can I assess step size and convergence? If I choose sampling methods, how can I design my sampler so that it mixes efficiently and samples from the posterior distribution quickly? If I choose variational methods, how robust are these methods to starting points? There are a large number of considerations to think about when designing inference proceedures.
- How can I evaluate overfitting in my approach? What can I change about my model to avoid overfitting?
- How will my approach scale or need to change as more (and possibly evolving) data are available?