# Machine Learning 101

Daniel Nguyen

November 5, 2018

**Supervised Learning**

- Process of trying to infer from labeled data the underlying function that produced the labels associated with the data

**Unsupervised Learning**

- Find patterns/relationships/structure in data, but are not optimized to solve a particular predictive task

**Regression**

- Learn relationships between inputs and outputs

- Linear regression uses predictor of the form $X\theta = y$ where $X$ is the matrix of inputs, $y$ is the outputs vector, and $\theta$ is the weights vector.

**Mean-squared Error**

$$E = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{X_i} \cdot \boldsymbol{\theta})^2$$

- What we try to minimize in our regression model

**Coefficient of determination ($R^2$ statistic):**

$$FVU(f) = \frac{MSE(f)}{Var(y)}$$

(FVU = fraction of variance unexplained)

- FVU(f) = 1 - Trivial predictor

- FVU(f) = 0 - Perfect predictor

**Occam's razor**

- Among competing hypotheses, the one with the fewest assumptions should be selected

- Hypothesis here is our weights $\boldsymbol{\theta}$.

- We want $\boldsymbol{\theta}$ with few non-zero parameters, meaning only a few features are relevant, and $\boldsymbol{\theta}$ that is almost uniform, meaning only a few features are significantly more relevant than others.

## Regularization

- Process of penalizing the model complexity

### L2 Regularization

$$R(\boldsymbol{\theta}) = \lambda ||\boldsymbol{\theta}||_2^2$$

- Regularized model means that we add the regularization term to our model

- For example, if we are trying to minimize Sum-squared Error, then the model will become

$$E = \frac{1}{N} \sum_{i=1}^{N} (y_i - \boldsymbol{X_i} \cdot \theta)^2 + \lambda ||\boldsymbol{\theta}||_2^2$$

Note: $||\boldsymbol{\theta}||_2^2 = \sqrt{\sum_i \theta_i^2}$

## Data Sets

- Training Set: used to optimize the model's parameters

- Validation Set: used to tune any model parameters that are not directly optimized by the training set

- Training Set: used to report how well we expect the model to perform on unseen data

## Naive Bayes

- Assumes that features are conditionally independent given the label, meaning we're assuming

$$P(feature_i, feature_j | label) = P(feature_i | label) \cdot P(feature_j | label)$$

where $i \neq j$

- Our probability model becomes

$$P(label | features) = \frac{P(label) \cdot \prod_i P(feature_i | label)}{P(features)}$$

- Can double count features that are highly correlated (i.e. features that represent the same information) because it makes the assumption that each $P(data | label)$ is independent, when it is not.

## Logistic Regression

- Optimize the likelihood of the data by modeling $P(label|data)$. We use a sigmoid function to convert $\boldsymbol{\theta} \cdot \boldsymbol{x}$ to a probability.

$$\sigma(t) = \frac{1}{1 + e^{-t}} = P(label|data) = y$$

  where $t = \boldsymbol{\theta} \cdot \boldsymbol{x}$

- The Cross-Entropy cost function (what we are trying to minimize) is given by

$$E(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \{t^n \ln(y^n) + (1 - t^n) \ln(1 - y^n)\}$$

  Note: the $n$ superscript notation refers to the values associated with the input $n$, and not the $n^{th}$ power

- The gradient of the cost function with respect to weight $j$ is given by

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_j} = -\sum_{n=1}^{N} (t^n - y^n) x_j^n$$

- Tend to overfit with small amount of data

## Softmax Regression

- A probabilistic model, similar to logistic regression; but is able to classify more than two outputs.

-
$$\sigma(t_i) = \frac{e^{t_i}}{\sum_{k=1}^{c} e^{t_k}} = P(label_i|data) = y_i$$

  where $t_i = \boldsymbol{\theta_i} \cdot \boldsymbol{x}$

- The Cross-Entropy cost function (what we are trying to minimize) is given by

$$E(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \sum_{k=1}^{c} t_k^n \ln(y_k^n)$$

  Note: the $n$ superscript notation refers to the values associated with the input $n$, and not the $n^{th}$ power

- The gradient of the cost function with respect to weight $jk$ is given by

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_{jk}} = -\sum_{n=1}^{N} (t_k^n - y_k^n) x_j^n$$

**Support Vector Machine**

- Try to minimize misclassification error using the classifier of the form

$$y_i = \begin{cases} 1 & \boldsymbol{X_i} \cdot \boldsymbol{\theta} - b > 0 \\ -1 & otherwise \end{cases}$$

- Since we are minimizing the number of misclassifications, we are doing

$$\arg min_{\boldsymbol{\theta}} \sum_i \delta(y_i(\boldsymbol{X_i} \cdot \boldsymbol{\theta} - b) \leq 0)$$

where the term $\boldsymbol{X_i} \cdot \boldsymbol{\theta} - b$ refers to the model's prediction, $y_i$ is the actual label, and $\delta$ is the indicator function that has the value 1 if the argument is true and 0 if the argument is false.

We can see that all we're doing here is counting how many were misclassified. If our prediction and label match, we will get positive values since we are multiplying them; otherwise, we get negative values.

**Classifier Evaluation**

- True Positive (TP): Labeled as T, predicted as T
  True Negative (TN): Labeled as F, predicted as F
  False Positive (FP): Labeled as F, predicted as T
  False Negative (FN): Labeled as T, predicted as F
  True Positive Rate (TPR) = true positives / #labeled positives
  True Negative Rate (TNR) = true negatives / #labeled negatives

- Classification Accuracy: correct predictions / #predictions

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

Error Rate: incorrect predictions / #predictions

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

Balanced Error Rate (BER):

$$BER = 1 - \frac{1}{2}(TPR + TNR) = \frac{1}{2}(FPR + FNR)$$

- Precision (How many of the selected items are relevant):

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

Recall (How many of the relevant items are selected (i.e. Total number of true in truth labels)):

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

4

- $F_1$ score:
$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

$F_\beta$ score:
$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 precision + recall}$$

We use low $\beta$ in case where precision is more important and high $\beta$ in case where recall is more important.

## Dimensionality Reduction

- Take high-dimensional data and describe it compactly using a small number of dimensions.

- Methods to achieve this: Principal Component Analysis (PCA), K-means clustering, Hierarchical Clustering, Community Detection

- **PCA** Finding a basis matrix $\beta$ such that when the input matrix $X$ is rotated ($Y = \beta X$)

    - Dimension with the highest variance is $Y_0$
    - Dimension with the 2nd highest variance is $Y_1$
    - So on...

  Then discard the dimensions with lower variance because that means that those features are highly correlated and do not give us useful information.

- **K-means Clustering** Find $K$ centroids (center of cluster) (denoted by the matrix $C$) and cluster assignments $\boldsymbol{y}$ so that the reconstruction error is minimized.

  Reconstruction error:
$$\sum_i ||\boldsymbol{X_i} - \boldsymbol{C_{y_i}}||_2^2$$

  Note 1: Each row of the matrix $C$ is a centroid for a cluster.

  Note 2: $y_i$ is the assigned cluster for $X_i$ so $\boldsymbol{C_{y_i}}$ is the centroid of the assigned cluster for $X_i$.

- **Hierarchical Clustering** Works by gradually fusing clusters whose points are close together.

- How to choose the hyperparameters K in K-means clustering or how many dimensions of PCA to keep?

    - As a means of compressing data, we can choose however many dimensions we can afford.

- As a means of generating useful features for predictive task, increasing the number of dimensions/number of clusters can give us more features to work with. Often times, we choose the dimensions that give us the lowest error on held out data.

- **Community Detection** Compactly represent relationships between points.

  - Used when points are not defined by their features but their relationships to each other
  - Difference between community detection and clustering is that community detection groups sets of points based on their connectivity while clustering groups sets of points based on their features.

A community is defined to have the following properties:

- Members are connected (i.e. community is a connected component)
- Has few edges between communities (i.e. minimum cut). The goal is to cut the network into two partitions such that the number of edges crossed by the cut is minimal.

  We prefer cut that favors large communities over small ones. To do this, we have to minimize cut cost.

  The normalized cut cost is defined by

$$\text{Normalized Cut}(C) = \frac{1}{|C|} \sum_{c \in C} \frac{\text{cut}(c, \bar{c})}{\sum \text{degrees in } c}$$

- "Cliqueishness". Find cliques using clique percolation method:
  1. Given clique of size K
  2. Initialize every K-clique to be its own community
  3. While (two communities I and J haves (K-1)-clique in common):
  4. Merge I and J into one community

- Network Modularity (dense inside, few edges outside)

  Null model: Edges are equally likely between any pair of nodes, regardless of structure. Our goal is to deviate from this model.

  Calculating network modularity:

$$e_{kk} = \frac{\# \text{ edges with both endpoints in community } k}{\# \text{ edges}}$$

$$a_k = \frac{\# \text{ edge endpoints in community } k}{\# \text{ edge endpoints}}$$

$$Modularity = Q = \sum_{k=1}^{K} (e_{kk} - a_k^2)$$

  Our goal is to find a set of communities such that network modularity is maximized.

**Recommender System**
How can we make a recommender system that is personalized to each user?

Definition:
$I_u$ = set of items purchased by user $u$
$U_i$ = set of users who purchased item $i$.

- Measure similarity between users in terms of the items they purchased.

- **Jaccard Similarity**

$$Jaccard(U_i, U_j) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|}$$

Has maximum of 1 if two items were purchased by the same set of users

Has minimum of 0 if two items were purchased by completely disjoint sets of items

Only works with binary decision vector (in this case purchase or not purchase)

- **Cosine Similarity**

$$\cos\theta = \frac{U_i \cdot U_j}{||U_i||||U_j||}$$

  - Works for arbitrary vectors (e.g. different ratings)
  - If $\cos\theta = 1$, that means $\theta = 0^o$, meaning both vectors $U_i$ and $U_j$ point in the same direction, signifying they are similar.
  - If $\cos\theta = 0$, that means $\theta = 90^o$, meaning both vectors $U_i$ and $U_j$ are orthogonal, signifying they are different.

- **Pearson Correlation**

$$Sim(u, v) = \frac{\sum_{i \in I_u \cap I_v}(R_{u,i} - \overline{R_u})(R_{v,i} - \overline{R_v})}{\sqrt{\sum_{i \in I_u \cap I_v}(R_{u,i} - \overline{R_u})^2 \sum_{i \in I_u \cap I_v}(R_{v,i} - \overline{R_v})^2}}$$

  - Similar to cosine similarity, but we subtract the average rating first so we have negative values for below-average ratings and positive values for above-average ratings.

- **Linear Model** $f(u, i) = \alpha + \beta_u + \beta_i$
  Definition:
  $\alpha$ = bias term
  $\beta_u$ = how much does this user tend to rate things above the mean
  $\beta_i$ = how much does this item tend to receive higher ratings than others

  The optimization problem becomes

$$\arg min_{\alpha,\beta} \sum_{u,i}(\alpha + \beta_u + \beta_i - R_{u,i})^2 + \lambda\left[\sum_u \beta_u^2 + \sum_i \beta_i^2\right]$$

We can solve this with gradient descent, but a much more efficient way is to use the closed-form formulas for the parameters. We repeat the updates formula until convergence.

$$\alpha = \frac{\sum_{u,i \in train}(R_{u,i} - (\beta_u + \beta_i))}{N_{train}}$$

$$\beta_u = \frac{\sum_{i \in I_u}(R_{u,i} - (\alpha + \beta_i))}{\lambda + |I_u|}$$

$$\beta_u = \frac{\sum_{u \in U_i}(R_{u,i} - (\alpha + \beta_u))}{\lambda + |U_i|}$$

This model still has the problem of fitting a function that treats users and items independently.

- **Latent-Factor Model** $f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$
  Definition:
  $\gamma_i$ = an item's properties
  $\gamma_u$ = a user's preferences
  $\gamma_i \cdot \gamma_u$ = prediction for matching user $u$ to item $i$

  The optimization problem becomes

  $$\arg min_{\alpha,\beta,\gamma} \sum_{u,i}(\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda \left[ \sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i ||\gamma_i||_2^2 + \sum_u ||\gamma_u||_2^2 \right]$$

  We solve this using an approximate solution.

  1. Fix $\gamma_i$. Solve $\arg min_{\alpha,\beta,\gamma_u} objective(\alpha, \beta, \gamma)$
  2. Fix $\gamma_u$. Solve $\arg min_{\alpha,\beta,\gamma_i} objective(\alpha, \beta, \gamma)$
  3. Repeat until convergence

  This model takes into account users' personal preferences through the $\gamma$ parameters. However, this model completely ignores user features and movies features. All the model is doing is take into account how the user often rates things and how the item often gets rated.

- **Binary Prediction with Logistic Regression**
  Definition:
  $P(i$ is preferred over $j) = \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$

  Maximizing the log-likelihood:

  $$\max \ln \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

  We can solve this using gradient ascent.

  In practice, it is not feasible to consider all pairs of $i$ and $j$ so we randomly sample pairs.

- **Extension of Latent-Factor Model**
  How can we incorporate features about users and items, handle implicit feedback, and handle change over time?

  1. **Add in features**
  $$f(u, i) = \alpha + \beta_u + \beta_i + (\gamma_u + \sum_{a \in A(u)} \rho_a) \cdot \gamma_i$$

  This model looks similar to the original latent-factor model, except for the term $\sum_{a \in A(u)} \rho_a$.
  The vector $A(u)$ is a binary attributes vector for user $u$ that encodes at each entry whether the user expresses that attribute using 0 or 1. Each term $\rho_a$ represents a parameter vector associated with attribute $a$. The parameter vector essentially encodes "How does having this attribute impact $\gamma_u$?"
  We fit this model using Sum-squared Error as usual, that is:
  $$\arg min_{\alpha, \beta, \gamma, \rho} \sum_{u, i \in train} (f(u, i) - R_{u,i})^2 + \lambda \Omega(\beta, \gamma)$$

  2. **Implicit Feedback**
  Users may not rate things, but still interact with the system (e.g. clicking on the items links). Similar approach to adding in item features, we use binary attributes vector that encodes user's actions.
  $$f(u, i) = \alpha + \beta_u + \beta_i + (\gamma_u + \frac{1}{||N(u)||} \sum_{a \in N(u)} \rho_a) \cdot \gamma_i$$

  Here, $N(u)$ represents a binary attributes vector for user $u$ that encodes at each entry the action the user takes. This time, we normalize by the magnitude of the actions vector to prevent the danger of

  (a) User might watch a lot of movies, so $\gamma_u$ might become very large. This could lead to the user getting recommended any items.
  (b) User might click on a lot of links before watching a movie, same thing happens as in (a).

  Again, we fit this the same way using Sum-squared Error.

  3. **Change Over Time** Rating data might change over time. We account for this by defining some parameters in our model as a function of time, as such
  $$f(u, i, t) = \alpha + \beta_u(t) + \beta_i(t) + \gamma_u(t) \cdot \gamma_i$$

  Our optimization task becomes
  $$\arg min_{\alpha, \beta, \gamma} \sum_{u, i, t \in train} (f(u, i, t) - R_{u,i,t})^2 + \lambda_1 \Omega(\beta, \gamma) + \lambda_2 ||\gamma(t) - \gamma(t + \delta)||$$

  Note the regularizer term contains $\lambda_2 ||\gamma(t) - \gamma(t + \delta)||$. This is to allow the parameters to change smoothly since the parameters should not change abruptly in short periods of time.