CSI 4106

Artificial Intelligence

Study Guide

Fall 2024

Types of Learning

- Supervised Learning: Each example is accompanied by a label
- Unsupervised Learning: No feedback is provided to the algorithm
- Reinforcement Learning: The algorithm receives a reward or a punishment following each action

Learning Phases

- Learning (building a model)
- Inference (using the model)

For learning, we first obtain training data, then we convert the data into features vectors. Then we use an algorithm to train a model.

For inference, we use new data to make new predictions with the model.

Supervised Learning

The dataset is a collection of labelled examples

- $\{(x_i, y_i)\}_{i=1}^{N}$
  - Each x_i is a feature vector with D dimensions
  - $x_i^{(j)}$ is the value of the feature j of the example i, for $j \in 1,\dots,D$ and $i \in 1,\dots,N$
  - The label y_i is either a class, taken from a finite list of classes, $\{1,2,\dots,C\}$, or a real number, or a complex object
- When the label y_i is a class, taken from a finite list of classes, $\{1,2,\dots,C\}$, we call the task a classification task
- When the label y_i is a real number, we call the task a regression task

Regression

- Training data is a collection of labelled examples
  - $\{(x_i, y_i)\}_{i=1}^{N}$
    - Each x_i is a feature vector with D dimensions
    - $x_i^{(j)}$ value of feature j of example i, for $j \in 1,\dots,D$ and $i \in 1,\dots,N$
  - Label y_i is a real number
- Problem: Given the data set as input, create a model that can be used to predict the value of y for an unseen x

Linear Regression

- A linear model assumes that the value of the label, $\hat{y}_i$, can be expressed as a linear combination of the feature values $x_i^{(j)}$:

$$\hat{y}_i = h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \dots + \theta_D x_i^{(D)}$$

- Here, θ_j is the j^{th} parameter of the (linear) model, with θ_0 being the bias term/parameter, and $\theta_1,\dots,\theta_D$ being the feature weights
- Problem: find values for all the model parameters so that the model "best fits" the training data
- Root mean square error is a common performance measure for regression problems

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(h(x_i)-y_i)^2}$$

Building Blocks of Learning Algorithms

- A typical learning algorithm comprises the following components:
  - A model, often consisting of a set of weights whose values will be "learnt"
  - An objective function
  - Optimization algorithm

Optimization

- Until some termination criteria is met:
  - Evaluate the loss function, comparing $h(x_i)$ to y_i
  - Make small changes to the weights, in a way that reduces the value of the loss function

Gradient Descent (Single value)

- A common loss function for regression problems is the root mean squared error

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(h(x_i)-y_i)^2}$$

- In practice, minimizing the mean squared error is easier and gives the same result

$$\frac{1}{N}\sum_{i=1}^{N}(h(x_i)-y_i)^2$$

- Suppose we have a linear model

$$h(x_i) = \theta_0 + \theta_1 x_i$$

and the loss function

$$J(\theta_0,\theta_1) = \frac{1}{N}\sum_{i=1}^{N}(h(x_i)-y_i)^2$$

We want to find θ_0 and θ_1 that minimizes J

- We run gradient descent
  - Initialization: θ_0 and θ_1 with random values or zeros
  - Loop:
    - Repeat until convergence:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1)$$

for $j = 0, 1$

  - α is called the learning rate - size of each step
  - $\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1)$ is partial derivative w.r.t. θ_j

We would have

$$J(\theta_0,\theta_1) = \frac{1}{N}\sum_{i=1}^{N}(\theta_0+\theta_1 x_i - y_i)^2$$

$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{2}{N}\sum_{i=1}^{N}(\theta_0+\theta_1 x_i - y_i)$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1) = \frac{2}{N}\sum_{i=1}^{N}x_i(\theta_0+\theta_1 x_i - y_i)$$

Multivariate Gradient Descent

- A multivariate linear regression model is defined as

$$h(x_i) = \theta_0 + \theta_1 x_i^{(1)} + \dots + \theta_D x_i^{(D)}$$

$x_i^{(j)}$ = value of feature j in i^{th} example

D = number of features

- The new loss function is

$$J(\theta_0,\theta_1,\dots,\theta_D) = \frac{1}{N}\sum_{i=1}^{N}(h(x_i)-y_i)^2$$

Its partial derivative

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{2}{N}\sum_{i=1}^{N}x_i^{(j)}(\theta x_i - y_i)$$

where θ, x_i and y_i are vectors, and θx_i is a vector operation

- The vector containing the partial derivatives of J is called the gradient vector

$$\nabla_\theta J(\theta) = \begin{pmatrix}\frac{\partial}{\partial\theta_0}J(\theta)\\ \frac{\partial}{\partial\theta_1}J(\theta)\\ \vdots \\ \frac{\partial}{\partial\theta_D}J(\theta)\end{pmatrix}$$

- This vector gives the direction of the steepest ascent
- It gives its name to the gradient descent algorithm

$$\theta' = \theta - \alpha\nabla_\theta J(\theta)$$

- The algorithm is as follows:

Repeat until convergence:

{

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1,\dots,\theta_D)$$

for $j \in [0,\dots,D]$ (update simultaneously)

}

- We assume that the objective function is differentiable

Local vs. Global minimum

- A function is convex if for every pair of points on the graph of the function, the line connecting these two points lies above or on the graph
  - A convex function has a single minimum
  - The loss function for the linear regression (MSE) is convex
- For functions that are not convex, the gradient descent algorithm converges to a local minimum
- The loss function generally used with linear or logistic regressions, and SVMs are convex, but not the ones for ANNs

Learning Rate

- Small steps, low values of α, will make the algorithm converge slowly
- Large steps might cause the algorithm to diverge
- Notice how the algorithm slows down naturally when approaching a minimum

Batch Gradient Descent

- This algorithm is known as batch gradient descent since for each iteration, it processes the "whole batch" of training examples
- This algorithm might take more time to converge if the features are on different scales
- The batch gradient descent becomes very slow as the number of training examples increases
- This is because all the training data is seen at each iteration

Stochastic Gradient Descent

- The stochastic gradient descent algorithm randomly selects one training instance to calculate its gradient
- This allows it to work with large training sets
- Its trajectory is not as regular as the batch algorithm
  - Because of its bumpy trajectory, it is often better at finding the global minima when compared to batch
  - Its bumpy trajectory makes it bounce around the local minimum

Mini-batch Gradient Descent

- At each step, rather than selecting one training example as SGD does, mini-batch gradient descent randomly selects a small number of training examples to compute the gradients
- Its trajectory is more regular compared to SGD
  - As the size of the mini-batches increases, the algorithm becomes increasingly similar to batch gradient descent, which uses all the examples at each step
- It can take advantage of the hardware acceleration of matrix operation, particularly with GPUs

Classification

- Binary classification is a supervised learning task where the objective is to categorize instances (examples) into one of two discrete classes
- A multi-class classification task is a task of supervised learning problem where the objective is to categorize instances into one of three or more discrete classes
- Some machine learning algorithms are specifically designed to solve binary classification problems
  - Logistic regression and SVM are such examples
- Any multi-class classification problem can be transformed into a binary classification problem
- One-vs-All
  - A separate binary classifier is trained for each class
  - For each classifier, one class is treated as the positive class, and all other classes are treated as the negative class
  - The final assignment of a class label is made based on the classifier that outputs the highest confidence score for a given input

Decision Boundary

- A decision boundary is a boundary that partitions the underlying feature space into regions corresponding to different class labels
- The data is linearly separable when two classes of data can be perfectly separated by a single linear boundary, such as a line in two-dimensional space or a hyperplane in higher dimensions
- 2 attributes, linear decision boundary would be a line
- 2 attributes, non-linear decision boundary would be a non-linear curve
- 3 attributes, linear decision boundary would be a plane
- 3 attributes, non-linear decision boundary would be a non-linear surface
- >3 attributes, linear decision boundary would be a hyperplane
- >3 attributes, non-linear decision boundary would be a hypersurface
- Revised definition: a decision boundary is a hypersurface that partitions the underlying feature space into regions corresponding to different class labels

Logistic Regression

- Despite its name, logistic regression serves as a classification algorithm rather than a regression technique
- The labels in logistic regression are binary values, denoted as $y_i \in \{0,1\}$, making it a binary classification task
- The primary objective of logistic regression is to determine the probability that a given instance x_i belongs to the positive class, i.e. $y_i = 1$
- The standard logistic function is defined as $\sigma: \mathbb{R} \to (0,1)$

$$\sigma(t) = \frac{1}{1+e^{-t}}$$

- An S-shaped curve, such as the standard logistic function, is termed a squashing function because it maps a wide input domain to a constrained output range.
- Analogous to linear regression, logistic regression computes a weighted sum of the weighted features, expressed as:

$$\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \dots + \theta_D x_i^{(D)}$$

- However, using the standard logistic function limits its output to the range (0,1):

$$\sigma(\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)} + \dots + \theta_D x_i^{(D)})$$

- The logistic regression model, in its vectorized form, is defined as:

$$h_\theta(x_i) = \sigma(\theta x_i) = \frac{1}{1+e^{-\theta x_i}}$$

- In logistic regression, the probability of correctly classifying an example increases as its distance from the decision boundary increases
- The principle holds for both positive and negative classes
- An example lying on the decision boundary has a 50% probability of belonging to either class

- Predictions are made as follows:
  - $y_i = 0$, if $h_\theta(x_i) < 0.5$
  - $y_i = 1$, if $h_\theta(x_i) \geq 0.5$
- The values of θ are learned using gradient descent

Feature Engineering

- The process of creating, transforming, and selecting variables from raw data to improve the performance of machine learning models

Underfitting and Overfitting

- Underfitting
  - Model is too simple
  - Uninformative features
  - Poor performance on both train and test data
- Overfitting
  - Model is too complex
  - Too many features
  - Excellent performance on training set, but poor performance on test set

Learning Curves

- We can visualize learning curves to assess our models
  - A learning curve shows the performance of the model on both training and test set
  - Multiple measurements are obtained by repeatedly training the model on larger and larger subsets of the data

Bias / Variance Tradeoff

- Bias is the error from overly simplistic models. High bias can lead to underfitting
- Variance is the error from overly complex models High variance can lead to overfitting
- Tradeoff: aim for a model that generalizes well to new data

Confusion Matrix

	Positive (predicted)	Negative (predicted)
Positive (Actual)	True Positive (TP)	False Negative (FN)
Negative (Actual)	False Positive (FP)	True Negative (TN)

Accuracy

- The accuracy of a model measures how accurate the result is
- Ratio of correct number of predictions to total number of predictions

$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{N}$$

- Accuracy can be misleading in the case of class imbalance, as it disproportionally reflects the performance on the majority class, masking poor performance on the minority class

Precision

- Also known as positive predictive value
- Proportion of true positive predictions among all positive predictions

$$\text{precision} = \frac{TP}{TP+FP}$$

- An algorithm that makes a small number of high-confidence predictions might achieve a high precision score, but may not be helpful

Recall

- Also known as sensitivity or true positive rate (TPR)

$$\text{recall} = \frac{TP}{TP+FN}$$

- Proportion of true positive instances correctly identified among all actual positive instances

F_1 Score

- Harmonic mean of precision and recall

$$F_1 \text{ score} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

$$= 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$= \frac{TP}{FP + \frac{FN+FP}{2}}$$

Micro Performance Metrics

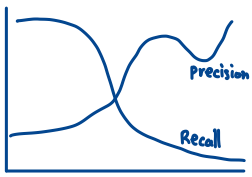
- Micro performance metrics aggregate the contributions of all classes to compute the average performance metric, such as precision, recall, or F_1 score.
- This approach treats each individual prediction equally, providing a balanced evaluation by emphasizing the performance on frequent classes

Macro Performance Metrics

- Compute the performance metric independently for each class and then average these metrics.
- Treats each class equally, regardless of frequency

Precision-Recall Tradeoff

- As the decision threshold decreases, a higher number of examples are predicted positive, potentially leading the classifier to eventually label all instances as positive
- Conversely, as decision threshold decreases, fewer examples are classified as positive, which may result in the classifier predicting no positive instances at all



ROC Curve

Receiver Operating Characteristics curve

- Curve shows the true positive rate against false positive rate
- An ideal classifier has TPR close to 1.0 and FPR close to 0.0
- $TPR = \frac{TP}{TP+FN}$
- TPR approaches one when the number of false negative prediction is low
- $FPR = \frac{FP}{FP+TN}$
- FPR approaches zero when the number of false positive is low