Comp 551: Applied Machine Learning

Shereen Elaidi

Winter 2020 Term

Contents

1 Week 1

1.1 Learning Objectives

2 Week 2

2.1 Learning Objectives

1. Linear Regression
    a) Linear model
    b) Evaluation Criteria
    c) How to find the best fit
    d) Geometric interpretation
2. Logistic Regression
    a) Linear Classifier
    b) Logistic regression – model
    c) Logistic regression – loss function
    d) Maximum likelihood view
    e) Multi-class classification

3 Week 3

3.1 Learning Objectives

1. Naive Bayes Classifier
    a) Generative vs. discriminative classifier
    b) Naive Bayes classifier – assumption
    c) Naive Bayes classifier – different design choices

4 Week 4

4.1 Learning Objects

1. Regularisation
    a) Overfitting and underfitting
    b) Regularisation (L1 and L2)
    c) MLE vs MAP estimator
    d) Bias and variance tradeoff
    e) Evaluation metrics and cross-validation

5 Week 5

5.1 Learning Objectives

1. Gradient Descent Methods
    a) Gradient Descent
    b) Stochastic Gradient Descent
    c) Method of momentum
    d) Sub-gradient
    e) Application to linear regression and classification
2. Evaluation
    a) Different types of error
    b) Common evaluation metrics
    c) Cross validation

6 Week 6

Topics covered: perceptron, max margin classifier, support vector machines, soft margin constraints, hinge loss, decision trees, greedy heuristic, entropy, mutual information, gini index, and overfitting.

6.1 Learning Objectives

1. Perceptron and Support Vector Machines
    a) The geometry of linear classification
    b) Perceptron learning algorithm
    c) Margin maximisation and support vectors
    d) Hinge loss and relation to logistic regression
2. Decision Trees
    a) Decision Trees: model, cost function, and how it is optimised.
    b) How to grow a tree and why you should prune.

6.2 Perceptron

The **perceptron** is the first machine learning algorithm. The assumption made by the perceptron is that given the data, there must be a hyperplane that separates one class from another. It additionally assumes that the data is binary, however we can later extend this to multi-class predictions. That is, for a binary classifier, all the points of one class lie on one side of a hyperplane, and the other points lie on the other side of a hyperplane. In high dimensional spaces, this almost always holds. This doesn't really happen often in lower-dimensional spaces. You can actually prove that this always holds in *infinite-dimensional* spaces.

In some sense, the perceptron is the opposite of the KNN. The KNN is very nice in low-dimensional spaces, because you don't need to deal with the "curse of dimensionality" and it's also much faster in lower-dimensional spaces. The perceptron is optimal for high-dimensional spaces. Moreover, KNN is not a linear classifier. For KNN, the decision boundaries are not necessarily linear (as we saw on the quiz); however, for the perceptron method it *must* be linear.

Assuming that such a hyperplane exists, the perceptron tries to find it. To mathematically define a hyperplane, you need a normal vector w and a bias term b:

$$\mathcal{H} := \{x \in \mathbb{R}^n \mid w^t x + b = 0\} \tag{1}$$

it has one less dimension than the ambient space. During test time, if you get an unknown point x, you classify it based on which side of the hyperplane it lies on. This is nice since computing which side of the hyperplane that the point lies on is always the same; all you need to do is compute $\text{sgn}(w^t x + b)$.

6.2.1 How can we find the hyperplane?

The way that we formalise that our label set is binary is by writing:

$$\mathcal{Y} = \{-1, +1\}$$

We have to learn two things: w and b. We want to learn only one thing, so assume that there is no offset. So, compute the following maps:

$$x_i \mapsto \begin{bmatrix} x_i \\ 1 \end{bmatrix}$$
$$w \mapsto \begin{bmatrix} w \\ b \end{bmatrix}$$

This is valid since

$$\left\langle \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \begin{bmatrix} w \\ b \end{bmatrix} \right\rangle = w^t x + b$$

This transformation just absorbs b into the data. This is reflected in the new hyperplane:

$$\mathcal{H} := \{x \in \mathbb{R}^{n+1} \mid w^t x = 0\}$$

Geometrically, what we are doing is insisting that our hyperplane now goes through the origin. It will still be the same solution, just in a higher dimension.

6.2.2 The Algorithm

High-level intuition: Every time you get a point wrong, you adjust your hyperplane. Loop over the dataset. Once you make no more mistakes, you know you've found a hyperplane that separates the data, and then you stop.

Algorithm 1: Perceptron Algorithm

Result: Perceptron Algorithm
$w = 0$ (set weights to zero, this will get everything wrong);
while *true* **do**
  $m = 0$ (the counter of how many pts mis-classified);
  for $\forall\ (x, y) \in \mathcal{D}$ **do**
    (checks which side of the hyperplane you are on:);
    if $yw^t x \leq 0$ **then**
      (this is the case if and only if the point is wrong, since you want the signs to align.);
      $w \leftarrow w^t yx$;
      (the above piece of code tries to reinforce the points if positive to make positive larger inner products, negative points have small inner products);
      $m \leftarrow m + 1$;
    end
    if $m = 0$ **then**
      (do this until you make a full pass over the dataset without getting anything wrong);
      break;
    end
  end
end

Some observations: the algorithm will take a long time if there is a lot of "wiggle room" between the data points (this is encoded by what is called the **margin size**, which will be discussed later), since it's easy to continue to offshoot the ideal hyperplane.

The **Perceptron Convergence Theorem** asserts that the algorithm is guaranteed to converge in a finite number of steps if our data is linearly separable. For a given x, the amount of times at which we get x incorrectly, encoded by k, is at most:

$$k \leq \frac{w^t x}{x^t x} \tag{2}$$