Final Exam Review Notes

Andrés Ponce

June 21, 2020

All the topics from classification, regression, and different models for each will be covered.

Linear Models for Classification

In supervised learning, we might have scenarios where we have an input vector $\mathbf{x}$ Given some of the training data, we would like to assign $\mathbf{x}$ to possibly k classes. There are some different approaches:

1. **Discriminant Functions**: [1] Here we have a function whose output value indicates which class the data point gets assigned to. For example, if a function has a value greater than y, then we assign it to one class, and another class otherwise. Basically, it would amount to having an equation like

$$\begin{cases} 0, & y(x) \leq 0 \\ 1 & y(x) > 0 \end{cases}$$

2. **Generative approach** With a generative approach, we try to model the class-conditional probability

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}$$

[1] Yikes!

The weight vector $\mathbf{w}$ lies on orthogonal to the **decision boundary**, or the boundary that differentiates between being assigned to C_1 or C_2.

For multiclass classifiers, we can have a couple different approaches

1. **One-versus-the-rest**: We build $K - 1$ classifiers to handle a binary classification. We basically ask the question "Is this point in class $k \in K$, or is it in not?" This means that for all classes we can learn a model that basically predicts whether a point will fall in this class or outside.

   The problem with this approach lies in that there may be **ambiguous** regions in the input space, typically when multiple classifiers predict a point lying outside of their region and then it becomes unclear which class exactly they belong to. When one model says "this point does not lie in C_k, several other classees may be saying this and lead to conflicting spaces.

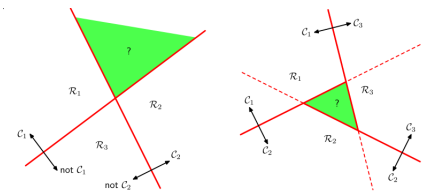

Figure 1: Here, when we map the contents outside of the main class, it can conflict with the classification made some of the other models.

2. **One-versus-one** Here we learn $K(K-1)/2$ models, one for each (C_1, C_2) pair. Thus, we have one model for each combination of pairs, and then the final classification is done by majority vote.

   How does this avoid ambiguous regions?

   $j \neq k$.

Least Squares for Classification

Using a data point for $\mathbf{x}$ for classfication, we define $\tilde{x}$ as being $(1, x^T)^T$. When we predict a point as belonging to class k, we can write it as a 1-0 vector with 1 being the index of the class the point was assigned to, for example $[0, 1, 0]$ if the point was assigned to the second class out of three.

Least Squares in general are more susceptible to outliers, in that points that lie outside the normal range will have more of an effect on the decision boundary, even if the majority of data points lie in a same range.

Another method to do classification is to use **Fisher's Linear Discriminant**

Fisher's Linear Discriminant

Fisher's Linear Discriminant focuses on the ratio S, which is the ratio between the ***between-class variance*** and the ***between-class variance***.

$$S = \frac{\sigma^2_{between}}{\sigma^2_{within}} \tag{1}$$

What we want to achieve is a classifier that separates the classes as far as possible. The weight vector $\mathbf{w}$ is going to be orthogonal to the decision boundary. We are going to project the points onto the decision boundary, and make a decision for our classes based on how the data points actually line up compared to the decision boundary. The between class variance is defined as

$$S_b\mathbf{w} = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$$

The above vector is in the direction of $\mathbf{m}_2 - \mathbf{m}_1$.

Once the data is projected [2] we can have several ways of actually carrying out the classification. We can do a **threshold** [3] where we just separate the classes based on how they all along that line.

We can also use a **nearest neighbors rule**, where we predice the class a point belongs to based on the points that are around it, and do a majority vote.

For multiple classes, we have to calculate the mean of all the input vectors, and use that measure for the between class variance vector.

[2] Remeber the projection onto the decision boundary is carried out by $\mathbf{w}\mathbf{x}\mathbf{w}^T$

[3] Like in Hw2, we had a value which we used to separate the values in C1 from C2.

Perceptron

The perceptron algorithm functions for two-class classification. In this algorithm, we first transform the input vector **x** with a function $\phi(x)$. We then do

$$y(x) = f(\mathbf{w}^T \phi(x))$$

where $f(x)$ is the **activation function**. This output then maps to $\{-1, 1\}$ depending on which class the point is estimated to belong to.

How do we know if the data point we just classified is correct? We can take the product

$$\mathbf{w}^T \phi(x_n) t_n$$

and see whether the result is greater than 0. Why is this the criterion? If the result is indeed greater than zero, it means that both the target **t** and the actual classification prediction had the same sign value, which means we predicted it correctly (maybe both values are -1, for example).

For the perceptron, to get the error we add up all the values that were misclassified, and then add it to the values of the perceptron.

Ensemble Methods

The main idea of using ensemble methods is to combine several weak classifiers, and to in the end have a *strong classifier*. For input points with multiple features, we can have multiple classifiers which individually learn a single feature, and then combine the classifiers with a single feature.

AdaBoost

With the **AdaBoost** algorithm, we continuosly draw simple classifiers, and for the inputs that we mistakenly classify, we increase their weight values so that they are "taken more seriously" the next times. The way the algorithm works is that we take the first classifier(with an initial weak distribution estimation) , then we update the estimation for time $\tau + 1$. The weight classifier is related to the **weight classifier** α, which is

$$\alpha_t = \frac{1}{2} ln \frac{1 - \epsilon_t}{\epsilon_t}$$

The final decision made by the **strong classifier** is the combined result of the weak classifiers.

$$\text{sign}(H(x) = \sum_{t=1}^{T} \alpha_t h_t(x))$$

Similar to when we did classification, whenever we had the product of our prediction and the actual class value,we can check the

prodct of the calss value y_i with our prediction. If the value is positive, then it means our classification to class -1 or 1 was indeed correct.

Face Detection with AdaBoost

When we try to perform face detection, we might try to have some sliding windows, which move and then try to detect some faces in the images. Since we are comparing several sub-areas of the image, the amount of times we have to perform face detection can be computationally too expensive. Since most of the regions in the image will not contain faces, we can focus only on the regions which contain the possibility for faces.

Viola-Jones Face Detection

Viola Jones is the most used way to do face-detection in almost real-time. We have several **integral images**. At a coordinate (x, y), we store the sum of the pixel values above and to the left of the pixel. This way, performing value calculations on a range of pixels can be done with a couple of additions and subtractions.

The complexity of Viola Jones is $O(MNK)$, where M is the number of iterations, N is the number of training examples, and K is the number of feature rectangles (could be a lot!).

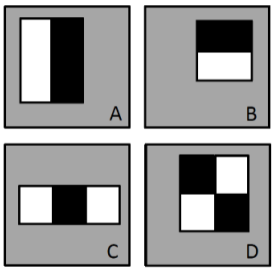

Figure 2: There are four patterns of rectangles available in Viola Jones. To find the value of a feature, we calculate the difference between the light and dark areas.

Attentional Cascade

The idea behind **Attentional cascades** is to chain progressively more strict classifiers which each provide less false positives. The error rate of all the classifiers is found by multiplying the errors of each one. Since the first classifier will not be quite that computationally complex, the features it throws out will definitely not be part of a face. This means that the only features the more strict classifiers will check actually come from the features that have a chance of being a feature in a face.

Decision Trees

With decision trees, we implement a tree where at each node we split the dataset according to a certain feature. For each feature we can split on, we have to calculate which one results in a more **pure**.

We have two criteria for trees:

1. **entropy**:

$$I_m = -\sum_{k=1}^{K} p_m^k \log p_m^k$$