Anthony Poerio (adp59@pitt.edu)

CS1674: Homework 8 - Written

Due: 11/7/2016, 11:59pm

QUESTIONS

1) **Why do we want a classifier that gives a large margin between positive and negative examples?**

   In Support Vector Machines, vectors in **positive space** signify one categorization (yes)—and vectors in **negative space** signify the opposite (no).

   The purpose of our classifier is to demarcate the division between these two spaces.

   Therefore, we want a classifier with a larger margin between the two spaces because this *helps us make a clearer distinction* between **positive** and **negative** categorization.

   With a larger margin between the spaces, we have a smaller probability of misclassifying future examples. Thus, we can expect fewer errors.

2) **Why can we do the "kernel trick" and not explicitly compute a lifting transformation?**

   For the purposes of an SVM, a 'kernel' can be viewed as a mapping function that transforms information from one kind of space into some higher dimensional space, using the **inner product** of the vectors we are studying.

   Inner product is roughly a measure of similarity between any two vectors, and the **kernel trick** allows us to transform our dataset by mapping it into some higher-dimensional space, while also taking similarity into account.

   Thus, if we can transform our data set into some higher dimensional space (based on similarity), and in the new higher-dimensional space our data set becomes linearly separable → then we can use this new higher-dimensional space as basis for separating our vectors into two classifications.

   That's what the 'kernel trick' does.

3) **Name the two ways to implement a multi-class classifier.**

The two ways to implement a multi-class classifier are:

1. **One-vs-All**
    - Train a set of classifiers, and map them into a feature space, *competing against all other vectors at the same time* (i.e. – 1-vs-all, 2-vs-all, etc..)
    - Map the vector that we are interested in, into that same feature space, to find the highest match, numerically, overall
    - Whichever area in feature space we are the closest match (highest value) '**wins**'

2. **One-vs-One**
    - Train all vectors against **every other** vector, separately
    - This gives us a set of binary classifiers (i.e. – 1-vs-2, 1-vs-3, etc…)
    - Each pair gets 1 vote for a 'label'.
    - Whichever classifier gets the ***most votes*** 'wins'

.

4) **Describe the problem of "overfitting", and why it is tricky to handle.**

Overfitting happens when we create a model that is very complex, and as a result, we begin to match against characteristics of the model that are not relevant in practice.

In this case, we are making classifications against 'noise' data points in the model, as well as relevant ones.

The problem is that this results in inaccurate classifications, even though the model 'thinks' that it is finding classifications with a high degree of accuracy.

The problem is difficult to handle, because of the **bias-vs-variance tradeoff**. If we remove parameters from our model to solve the overfitting problem (a result of too much sensitivity or 'variance'), ***we risk creating a model with a large 'bias'***, or not enough flexibility → in the worst case, we can create a model that suffers from "underfitting".

Therefore, the goal is to find an accurate balance.

.

5) **What two things do we need, in order to enable zero-shot classification of never-seen animals?**

a) ***First***, we need to know abstract attributes that the animal has.
b) ***Second***, we need to have classifiers trained to detect those attributes, using images of other objects (animals, for instance).