Kernel SVM for Image Classification

by Nathaniel Hobbs
May 5, 2018

Instructor: Professor Dana

1 Introduction

This work investigates the utility of Kernel SVMs for the task of image classification. This is a supervised learning task: the learning algorithm is given a set of labeled data on which to train a model that will then be used to classify images. We use a subset of the CIFAR-10 data set which consists of 60,000 32×32 color images in 10 classes, with 6000 images per class. These are broken into 50,000 images for training and 10,000 images for testing. Details of the experiment are given in Section 4.

Because SVMs are binary classifiers and we wish to classify into more than two classes, we will employ the "one-vs-one" reduction method. That is, for K classes we train (K choose 2) binary classifiers. Each classifier will be trained on samples from a pair of classes and learn to distinguish between them. When making a prediction on an unlabeled image, each of the $K(K-1)/2$ classifiers will be given the image, and then "vote" on which class they think the image belongs to, i.e. the class that gets the highest number of positive predictions among all classifiers' predictions will be taken as the "winner".

For each binary classifier, the training data will of the form $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, where $x^{(i)} \in \mathcal{X}$ is a 32×32 pixel RGB image from CIFAR-10 and the label is $y^{(i)} \in \{-1, +1\}$, representing in being in one (binary) class over the other.

2 Soft-Margin SVM Primal to Dual

2.1 Primal

Recall that the Primal form of the optimal soft-margin classifier (aka L1 soft-margin SVM):

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, i = 1, \ldots, m \\ & \xi_i \geq 0, i = 1, \ldots, m \end{aligned} \tag{1}$$

where

- $x^{(i)} \in \mathbb{R}^n$ is a point
- $y^{(i)} \in \{-1, +1\}$ is the label of $x^{(i)}$
- $w \in \mathbb{R}^n$ is a vector that is normal to the separating hyperplane
- $b \in \mathbb{R}$ is a bias parameter (how shifted the separating hyperplane is from the origin
- $\xi_i \in \mathbb{R}$ is a slack (error) variable that represents the amount that a point $x^{(i)}$ has a (functional) margin less than 1

- $C \in \mathbb{R}$ is a parameter to control the relative weighting of maximizing the margin (minimizing $\|w\|^2$) and ensuring that most examples have functional margin at least 1. If a misclassified point (i.e. $\xi_i > 0$) has functional margin $1 - \xi_i$, then the cost of the objective function increases by $C\xi_i$

We note that this optimization problem is a *quadratic linear program*, because the objective is quadratic and the constraints are linear.

Furthermore, we have that the vector normal to the separating hyperplane is $w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$ and the bias term is $b = y^{(i)} - wx^{(i)}$. Lastly, the predictor function $f : \mathcal{X} \to \{-1, +1\}$ is $f(x) = \text{sign}(w^T x + b)$.

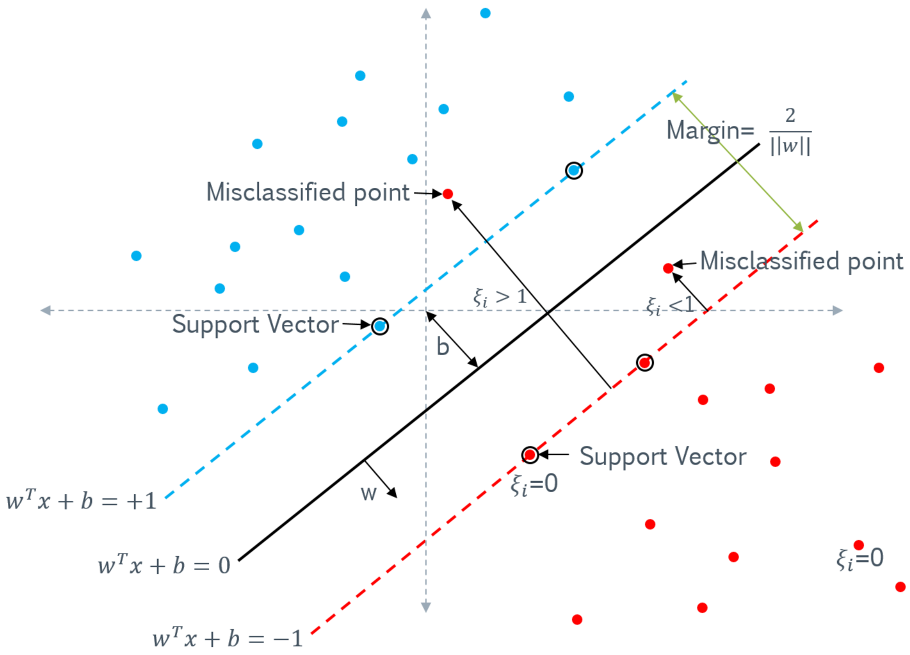


Figure 1: Visualization of Soft Margin SVM. Points are in $\mathbb{R}^2$ and the two classes are represented with colors red ($y = -1$) and blue($y = +1$). The gray cross is the axis, the black line is the separating hyperplane, the dashed blue and red lines are the supporting hyperplanes to the blue and red points, respectively. The points circled in black (touching the supporting hyperplanes) are the support vectors. There are two misclassified red points, both labeled, along with corresponding values of ξ that show the difference in its value depending on how much over the wrong side of supporting hyperplane of their class they lie. The green arrow between the red and blue dashed lines represents the margin that the SVM is trying to maximize.

2.2 Dual

2.2.1 Lagrangian

The Lagrangian for the optimization problem is given by:

$$\mathcal{L}(w,b,\xi,\alpha,\beta) = \underbrace{\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i}_{\substack{\text{Primal Objective}\\ f_0}} + \underbrace{\sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}(w^Tx^{(i)}+b)]}_{\substack{\text{First set of constraints}\\ f_1,\dots,f_m}} + \underbrace{\sum_{i=1}^{m}\beta_i(-\xi_i)}_{\substack{\text{Second set of constraints}\\ f_{m+1},\dots,f_{2m}}} \tag{2}$$

2.2.2 Reduced Lagrangian

We wish to find a solution to the following:

$$\max_{\substack{\alpha,\beta:\\ \alpha_i\ge 0,\beta_i\ge 0}} \theta_{\mathcal{D}}(\alpha,\beta) = \max_{\substack{\alpha,\beta:\\ \alpha_i\ge 0,\beta_i\ge 0}} \min_{\substack{w,b,\xi:\\ \xi_i\ge 0}} \mathcal{L}(w,b,\xi,\alpha,\beta) \tag{3}$$

Where $\theta_{\mathcal{D}}(\alpha,\beta)$ is the dual problem and the α_is and β_is are Lagrange multipliers, each constrained be ≥ 0. To find the dual form of the problem, fix α and β and minimize $\mathcal{L}$ with respect to w, to b, and ξ. When we set the respective derivatives to zero, we will find the point where the Lagrangian is minimized, and then we can maximize the reduced Lagrangian with respect to α and β.

Taking the derivative of the Lagrangian with respect to w and setting it to zero, we have:

$$\begin{aligned}
\frac{\partial}{\partial w}\mathcal{L}(w,b,\xi,\alpha,\beta) &= \frac{\partial}{\partial w}\Big(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}(w^Tx^{(i)}+b)] + \sum_{i=1}^{m}\beta_i(-\xi_i)\Big) = 0 \\
&\implies w - \sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)} = 0 \\
&\implies \boxed{w = \sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}}
\end{aligned} \tag{4}$$

Now, taking the derivative of the Lagrangian with respect to b and setting it to zero, we have:

$$\begin{aligned}
\frac{\partial}{\partial b}\mathcal{L}(w,b,\xi,\alpha,\beta) &= \frac{\partial}{\partial b}\Big(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}(w^Tx^{(i)}+b)] + \sum_{i=1}^{m}\beta_i(-\xi_i)\Big) = 0 \\
&\implies \boxed{\sum_{i=1}^{m}\alpha_i y^{(i)} = 0}
\end{aligned} \tag{5}$$

And finally, taking the derivative of the Lagrangian with respect to ξ_i and setting it to zero, we have:

$$\frac{\partial}{\partial \xi_i}\mathcal{L}(w,b,\xi,\alpha,\beta) = \frac{\partial}{\partial \xi_i}\Big(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}(w^T x^{(i)} + b)] + \sum_{i=1}^{m}\beta_i(-\xi_i)\Big) = 0$$

$$\implies C - \alpha_i - \beta_i = 0 \tag{6}$$

$$\implies \beta_i = C - \alpha_i$$

But because the β_is are a dual variables with $\beta_i \geq 0$, then this leads to a constraint that $C - \alpha_i \geq 0$ or $\alpha_i \leq C$. This along with the fact that α_i are dual variables with $\alpha_i \geq 0$ we have

$$\boxed{0 \leq \alpha_i \leq C} \tag{7}$$

We now will take these results and plug them back into our full Lagrangian to get a reduced Lagrangian that depends only on α and β:

Our original Lagrangian

$$\mathcal{L}(w,b,\xi,\alpha,\beta) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}(w^T x^{(i)} + b)] + \sum_{i=1}^{m}\beta_i(-\xi_i)$$

Replace w with $\sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)}$ from (4)

$$= \frac{1}{2}\|\sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)}\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$+ \sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}((\sum_{j=1}^{m}\alpha_j y^{(j)} x^{(j)})^T x^{(j)} + b)]$$

$$- \sum_{i=1}^{m}\beta_i\xi_i$$

Exapand squared norm

$$= \frac{1}{2}(\sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)})^T(\sum_{j=1}^{m}\alpha_j y^{(j)} x^{(j)}) + C\sum_{i=1}^{m}\xi_i$$

$$+ \sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}((\sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)})^T x^{(i)} + b)]$$

$$- \sum_{i=1}^{m}\beta_i\xi_i$$

Group like terms

$$= \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)} y^{(j)} (x^{(i)})^T x^{(j)} + C\sum_{i=1}^{m}\xi_i$$

$$+ \sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}((\sum_{j=1}^{m}\alpha_j y^{(j)} x^{(j)})^T x^{(i)} + b)]$$

$$- \sum_{i=1}^{m}\beta_i\xi_i$$

Distribute y_i

$$= \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + C\sum_{i=1}^{m}\xi_i$$
$$+\sum_{i=1}^{m}\alpha_i[(1-\xi_i) - (\sum_{j=1}^{m}\alpha_j y^{(i)}y^{(j)}x^{(j)})^T x^{(i)} + by^{(i)}]$$
$$-\sum_{i=1}^{m}\beta_i\xi_i$$

Distribute $\sum_{i=1}^{m}\alpha_i$

$$= \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + C\sum_{i=1}^{m}\xi_i$$
$$+\sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i - \sum_{i=1,j}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(j)})^T x^{(i)} + b\sum_{i=1}^{m}\alpha_i y^{(i)}$$
$$-\sum_{i=1}^{m}\beta_i\xi_i$$

Note that $(x^{(j)})^T x^{(i)}$ is the same as $(x^{(i)})^T x^{(j)}$, so replace

$$= \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + C\sum_{i=1}^{m}\xi_i$$
$$+\sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i - \sum_{i=1,j}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + b\sum_{i=1}^{m}\alpha_i y^{(i)}$$
$$-\sum_{i=1}^{m}\beta_i\xi_i$$

Simplify

$$= -\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + C\sum_{i=1}^{m}\xi_i$$
$$+\sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i + b\sum_{i=1}^{m}\alpha_i y^{(i)}$$
$$-\sum_{i=1}^{m}\beta_i(\xi_i)$$

We have from (5) that $\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$

$$= -\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + C\sum_{i=1}^{m}\xi_i$$
$$+\sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i$$
$$-\sum_{i=1}^{m}\beta_i\xi_i$$

We have from (6) that $C = \alpha_i + \beta_i$

$$= -\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + \sum_{i=1}^{m}\alpha_i\xi_i + \sum_{i=1}^{m}\beta_i\xi_i$$

$$+\sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i$$

$$-\sum_{i=1}^{m}\beta_i\xi_i$$

Simplify

$$= -\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} + \sum_{i=1}^{m}\alpha_i$$

Re-arrange terms

$$= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)}$$

Finally then, we have a reduced Lagrangian that only depends on the value of α:

$$\boxed{\mathcal{L}(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^T x^{(j)}} \tag{8}$$

2.2.3 Dual Problem

Recall that the dual function always has as a constraint that $\alpha \geq 0$. Putting this together with the above formulation of the Lagrangian, as well as the constraints we obtained from (5) and (7), we obtain the dual optimization problem:

$$\begin{aligned} \max_{\alpha} \quad & J(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}\langle x^{(i)}, x^{(j)}\rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, i = 1, \ldots, m \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned} \tag{9}$$

Note that we have replaced $(x^{(i)})^T x^{(j)}$ with an inner product $\langle x^{(i)}, x^{(j)}\rangle$. This is just the definition of the inner product, and will be useful conceptually for developing the idea of the kernel shortly.

A quick note on some intuition behind the slack variable: without it, α_i can go to ∞ when constraints are violated (i.e. points are misclassified). Upper-bounding the α_is by C allows some leeway in having points cross the supporting hyperplane of their class, the number of points and how much error is tolerated can be tweaked by varying C.

2.2.4 KKT Conditions and Strong Duality

We note that in the Primal the objective function f_0 and constraints $f_1, \ldots, f_{2m}$ are all convex, and that the constraints are (strictly) feasible.

In this case, there must exist a w^*, b^*, ξ^* that are a solution to the primal and α^*, β^* that are the solution to the dual such that p^* (the optimal value of the primal) equals d^* (the optimal solution to the dual) equals $\mathcal{L}(w, b, \xi, \alpha, \beta)$.

Furthermore, $w^*, b^*, \xi^*, \alpha^*, \beta^*$ satisfy the **KKT conditions**:

1. The primal constraints hold, i.e.
   - $f_i(w^*, b^*, \xi^*) \leq 0, i = 1, \ldots, 2m$
2. The dual constraints hold, i.e.
   - $\alpha^* \geq 0$
   - $\beta^* \geq 0$
3. Complimentary slackness
   - $\alpha_i^* f_i = 0, i = 1, \ldots, m$
   - $\beta_i^* f_i = 0, i = m+1, \ldots, m$
4. The gradient of the Lagrangian with respect to w, b, and ξ vanishes
   - $\frac{\partial}{\partial w}\mathcal{L}(w^*, b^*, \xi^*, \alpha^*, \beta^*) = 0$
   - $\frac{\partial}{\partial b}\mathcal{L}(w^*, b^*, \xi^*, \alpha^*, \beta^*) = 0$
   - $\frac{\partial}{\partial \xi}\mathcal{L}(w^*, b^*, \xi^*, \alpha^*, \beta^*) = 0$

In our formulation of the dual, we have ensured that all the KKT conditions are satisfied as well as the conditions for p^* to equal d^*. So, we are justified in using the dual form of the problem.

2.2.5 Interesting Utility of Complimentary Slackness to Dual SVM

By complimentary slackness, we have $\alpha_i^* f_i = 0, i = 1, \ldots, m$. This means that for any training points x_i, either $\alpha_i = 0$ or $y^{(i)}(w^T x_i + b) = 1 - \xi_i$ (the corresponding primal constraint is tight).

If $\alpha_i = 0$ then $y^{(i)}(w^T x_i + b) = 1 - \xi_i \implies y^{(i)}(w^T x_i + b) \geq 1$.

If $\alpha_i = C$ then $y^{(i)}(w^T x_i + b) = 1 - \xi_i \implies y^{(i)}(w^T x_i + b) \leq 1$.

If $0 < \alpha_i < C$ then $y^{(i)}(w^T x_i + b) = 1 - \xi_i \implies y^{(i)}(w^T x_i + b) = 1$.

That is to say, once we've solved the dual, we can look at the vector α and any $0 < \alpha_i < C$ entry corresponds to a support vector. This will give us an easy way to see how well the SVM has performed at generalized learning (fewer support vectors indicates better learning a la VC dimension) as well as an easy way to retrieve points from the training set that are support vectors.

2.2.6 Classification with the Dual

Classification with the dual is relatively straight forward. First, recall the Lagrangian:

$$\mathcal{L}(w,b,\xi,\alpha,\beta) = \underbrace{\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i}_{\substack{\text{Primal Objective} \\ f_0}} + \underbrace{\sum_{i=1}^{m}\alpha_i[(1-\xi_i) - y^{(i)}(w^T x^{(i)} + b)]}_{\substack{\text{First set of constraints} \\ f_1,\dots,f_m}} + \underbrace{\sum_{i=1}^{m}\beta_i(-\xi_i)}_{\substack{\text{Second set of constraints} \\ f_{m+1},\dots,f_{2m}}} \tag{10}$$

Once the dual is solved and some optimal α^* is found, $0 < \alpha_k^* < C$ for some k implies the corresponding constraint f_k is tight and $\xi_i = 0$ (see Section 2.2.5), i.e. $y^{(k)}(w^T x^{(k)} + b) = 1$

By complimentary slackness and for $0 < \alpha_k < C$

$$y^{(k)}(w^T x^{(k)} + b) = 1$$

Multiply both sides by $y^{(k)}$

$$y^{(k)} y^{(k)}(w^T x^{(k)} + b) = y^{(k)}$$

Because $y^{(k)} \in \{-1, +1\}$, then $(y^{(k)})^2 = 1$

$$w^T x^{(k)} + b = y^{(k)}$$

Solving for b we get:

$$\boxed{b = y^{(k)} - w^T x^{(k)}} \tag{11}$$

The KKT conditions also tell us that the gradient of the Lagrangian vanishes with respect to w, so , we have from (4) that

$$w = \sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)}$$

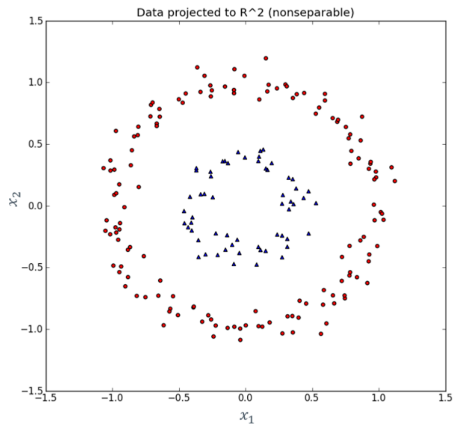
Therefore the linear separator can be found by solving $w = \sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)}$, using that value of w to solve for $b = y^{(k)} - w x^{(k)}$ for any k where $0 < \alpha_k^* < C$.

Unlabeled points can then be solved using the same function as the primal, i.e. $f(x) = \text{sign}(w^T x + b)$

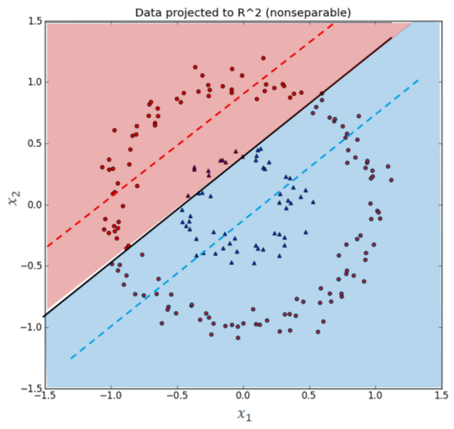
3 Non-Linearly Separable Data

3.1 Lifting to Higher-Dimensional Space

In many cases, the data that we wish to classify isn't linearly separable in an immediate sense. Taking an example from [5] (in which we modify some of the following images): imagine a set of points in $\mathbb{R}^2$ where all points belong to two classes. All points are generated by a true function where points in one class are defined as those around a small ring or radius ≈ 0.5 plus some Gaussian noise, and points in the other other class are those which lye around a larger ring of radius ≈ 1 plus some Gaussian noise.



(a) Points in $\mathbb{R}^2$. The blue triangles forming the smaller circle is one class and the red dots forming the larger circle is another class. [5]



(b) An example classifier (black line) along with shaded regions representing the classification of points to either the blue class or the red class. It should be clear that no matter how a line is drawn (in $\mathbb{R}^2$), a good separation of the points where almost blue triangles fall on one side and all red circles fall on the other of the black line is impossible.

Figure 2: Points in $\mathbb{R}^2$ that are not linearly separable in $\mathbb{R}^2$

If we were to apply an SVM directly to the learn from the data in this space, it's clear that there is no good linear separator. However, if we can "lift" the points to a higher dimensional space by e.g. applying a function to each of the the points in the space, then perhaps in that higher dimensional space there will exist a good separating hyperplane between the classes.

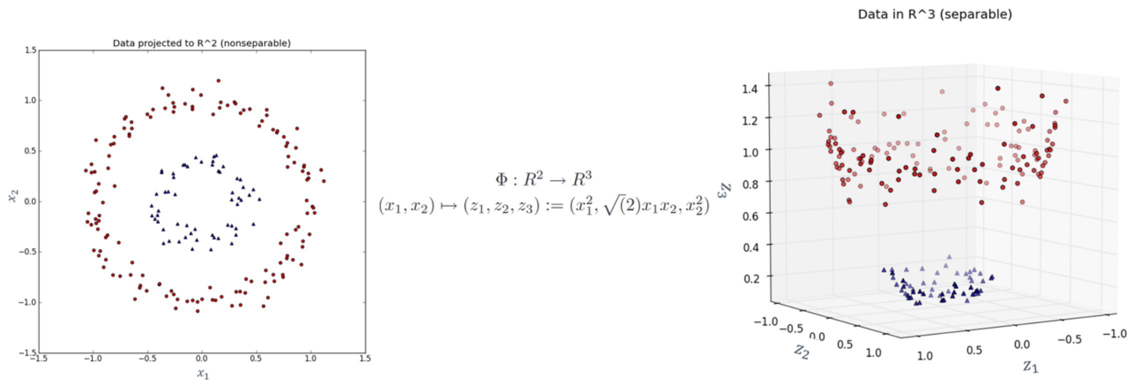


Figure 3: Points in original space on the left, "lifted" points are on the right. The transformation function is shown in the middle. The transformation here is essentially taking a parabola centered at the origin that is then rotated creating a conic surface, the mapping is projecting the points from $\mathbb{R}^2$ onto the resulting surface (i.e. lifting them up into the z_3 dimension by this projection while maintaining their values in the z_1 and z_2 dimensions from the values in the x_1 and x_2 dimensions, respectively.)

Once a separating hyperplane has been found in the higher dimensional space, we can project it back down to the original space and use that as a classifying function.

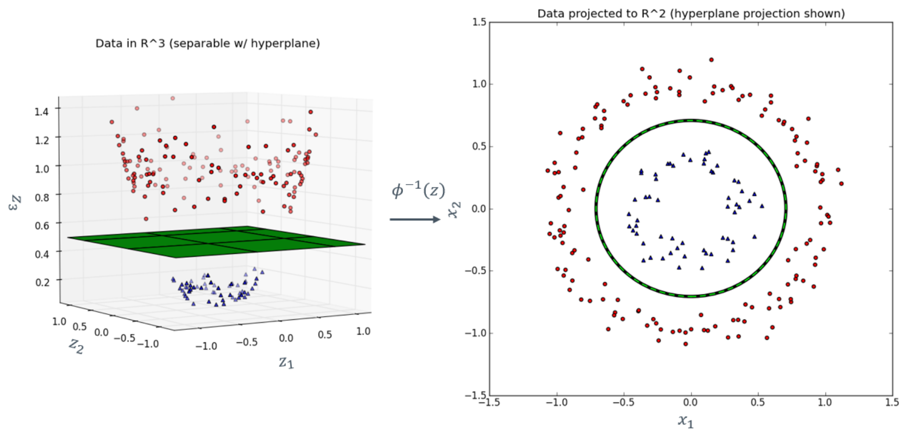


Figure 4: On the left, a separating hyperplane is shown in the $\mathbb{R}^3$ space that splits points in the red circle class from the points in the blue triangle class. On the right is the projection of this separating hyperplane in the original $\mathbb{R}^2$ space. While in the original space the classifier looks non-linear, it is simply the projection of a linear separator in a higher dimensional space.

3.2 The Kernel Trick

3.2.1 Putting everything in terms of the inner product

The basic idea is that it is possible to get all the benefits of lifting to a higher dimension without the computational baggage. We will argue that everything that SVMs benefit from lifting techniques can be taken in terms of taking the inner product of the points in a higher dimensional space. Therefore, if there's a way to do the inner product efficiently, then SVMs can get the benefits of lifting without ever "visiting" the higher dimensional space (i.e. mapping the points into the space, and then taking inner products).

As a straightforward example to begin establishing this fact, take the dual optimization problem. The only place where it matters that points have been lifted in into a higher dimension happens in the objective function:

$$J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \phi(x^{(i)}, \phi(x^{(j)}) \rangle$$

So, an inner product is enough for training. Assuming that there will be a payoff by putting things in terms of the inner product, let's see how classification would work.

We have from (11)

$$b = y^{(k)} - w^T \phi(x^{(k)})$$

and from (4) we have that $w = \sum_{i=1}^m \alpha_i y^{(i)} \phi(x^{(i)})$. Substituting this w we get

$$b = y^{(k)} - (\sum_{i=1}^m \alpha_i y^{(i)} \phi(x^{(i)}))^T \phi(x^{(k)})$$

Simplifying we get

$$b = y^{(k)} - \sum_{i=1}^m \alpha_i y^{(i)} (\phi(x^{(i)}))^T \phi(x^{(k)})$$

By definition of the inner product

$$\boxed{b = y^{(k)} - \sum_{i=1}^m \alpha_i y^{(i)} \langle \phi(x^{(i)}), \phi(x^{(k)}) \rangle} \tag{12}$$

Having b and the equation from (4) for w, we can rewrite the primal classification function in terms of the inner product as well.

Recall the classification function $f(x)$ from the primal

$$f(x) = \text{sign}\left(w^T \phi(x^{(k)}) + b\right)$$

Substitute w from (4)

$$f(x) = \text{sign}\left((\sum_{i=1}^m \alpha_i y^{(i)} \phi(x^{(i)}))^T \phi(x^{(k)}) + b\right)$$

Simplify

$$f(x) = \text{sign}\left(\sum_{i=1}^m \alpha_i y^{(i)} (\phi(x^{(i)}))^T \phi(x^{(k)}) + b\right)$$

Again, by definition of the inner product we get

$$\boxed{f(x) = \text{sign}\left(\sum_{i=1}^m \alpha_i y^{(i)} \langle \phi(x^{(i)}), \phi(x^{(k)}) \rangle + b\right)} \tag{13}$$

In conclusion, both training and prediction can be done using just the results of inner products without ever mapping the points themselves into any higher dimensional space.

3.2.2 Defining the Kernel

The Kernel is defined to be any function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ such that for all $x, x' \in \mathcal{X}$ for some $\phi(\cdot)$:

$$\boxed{K(x, x') = \langle \phi(x), \phi(x') \rangle} \tag{14}$$

provided the above is positive semidefinite and symmetric. This is known as **Mercer's Theorem**.

As a somewhat non-trivial example, consider points $x, x' \in \mathbb{R}$ that we wish to map into $\mathbb{R}^4$ using a degree 3 polynomial, i.e. , where

Definition of $\phi : \mathbb{R}^2 \to \mathbb{R}^4$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3, z_4) \coloneqq (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$$

Mapping points $x, x' \in \mathbb{R}^2$ to $\mathbb{R}^4$ using ϕ and finding their inner product

$$\langle \phi(x_1, x_2), \phi(x_1', x_2') \rangle = \Big\langle (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3), ({x_1'}^3, \sqrt{3}{x_1'}^2x_2', \sqrt{3}x_1'{x_2'}^2, {x_2'}^2) \Big\rangle$$

Expanding the inner product we get

$$= x_1^3{x_1'}^3 + 3x_1^2{x_1'}^2x_2x_2' + 3x_2^2{x_2'}^2x_1x_1' + x_2^3{x_2'}^3$$

We can define a Kernel function on the points $x, x' \in \mathbb{R}^2$ that achieve the same result

Define $K : \mathbb{R}^2 \to \mathbb{R}$ as

$$K(x, x') \coloneqq \big(\langle x, x' \rangle\big)^3$$

Plugging in values of x and x' we get

$$K(x, x') = \Big(\big\langle (x_1, x_2), (x_1', x_2') \big\rangle\Big)^3$$

Expanding the inner product we get

$$= (x_1x_1' + x_2x_2')^3$$

Expanding the polynomial we get

$$= x_1^3{x_1'}^3 + 3x_1^2{x_1'}^2x_2x_2' + 3x_2^2{x_2'}^2x_1x_1' + x_2^3{x_2'}^3$$

This is an explicit example of the polynomial kernel of degree 3. In general, the kernel for polynomial of degree d is defined as:

$$\boxed{K_{poly}(x, x') = \big(\langle x, x' \rangle\big)^d} \tag{15}$$

Other kernels are the linear kernel (in our case, this results in the equivalent of just finding a standard linear separator):

$$\boxed{K_{linear}(x, x') = \langle x, x' \rangle} \tag{16}$$

As well as the radial basis function (RBF) kernel, which centers a "bump" or "cavity", i.e. a Gaussian with variance σ^2, around each point:

$$\boxed{K_{rbf}(x, x') = \frac{e^{-\|x - x'\|^2}}{2\sigma^2}} \tag{17}$$

The RBF function is particularly interesting because it achieves the equivalent of lifting into an infinite dimensional space to find a separating hyperplane. However, using the kernel trick, we can achieve the same results without going through what would otherwise be a computational intractable procedure.

The RBF kernel has been shown to give some of the best state-of-the-art results with SVM. It can be thought of as putting a "bump" or "cavity" around each point in the training set, either "pushing up" or "pulling down" the space, depending on the class. The accumulation of these bumps and cavities aggregate creating a complicated surface. σ controls how tall or fat the bumps/crevices are.

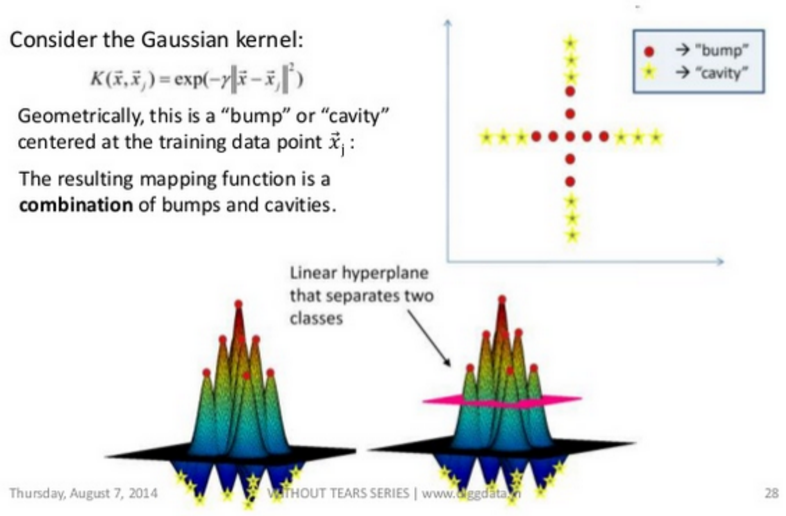

Consider the Gaussian kernel:
$K(\vec{x},\vec{x}_j) = \exp(-\gamma\|\vec{x}-\vec{x}_j\|^2)$
Geometrically, this is a "bump" or "cavity" centered at the training data point $\vec{x}_j$:
The resulting mapping function is a **combination** of bumps and cavities.
● → "bump"
★ → "cavity"
Linear hyperplane that separates two classes
Thursday, August 7, 2014
WITHOUT TEARS SERIES | www.bigdata.in
28

Figure 5: Two classes shown as red circles and yellow stars. Red points are "bumps" that raise the surface, yellow stars are "cavities" that drag down the surface, both created by the Gaussian Kernel (very similar to RBF). A seperating hyperplane is shown for classification.(Image from [6])

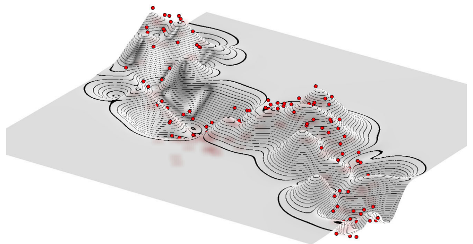


Figure 6: Another example of the geometric surface created applying an RBF kernel. While all the points are the same color, the classes can be deduced by if the points produce a "hill" above the flat surface or a "crevice" below. Separating hyperplane not shown. (Image from [7])

3.2.3 Training and Classification via Kernel SVM

As shown above, we can replace any inner product in the SVM with the equivalent kernel and achieve the same results as lifting to a higher degree space. Denoting with $K(\cdot,\cdot)$ the choice of whichever kernel one wishes to use to emulate mapping $\phi(\cdot)$, we have for training:

$$\begin{aligned} \max_{\alpha} \quad & J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} K(x^{(i)}, x^{(j)}) \\ \text{s.t.} \quad & K(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle \\ & 0 \le \alpha_i \le C, i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{18}$$

And for classification, assuming point $x^{(k)}, y^{(k)}$ corresponding to $0 < \alpha_k < C$:

$$b = y^{(k)} - \sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x^{(k)}) \tag{19}$$

And, assuming the same point $x^{(k)}$ and b as above:

$$f(x) = \text{sign}\Big(\sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x^{(k)}) + b\Big) \tag{20}$$

4 Experiments

4.1 Setup

The images we are classifying are a subset of the CIFAR-10 [1] data set which consists of 60000 32×32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images.

Specifically, we will be classifying into a subset of four classes: airplane, automobile, bird, and cat. We are choosing 1500 images from each class for training and 1000 images for testing. We are choosing this number of subsets and number of training samples for time reasons. Training a single classifier takes around 1 hour, and we need to train six classifiers to employ the "one-vs-one" reduction method.

The "one-vs-one" reduction method for K class classification requires (K choose 2) binary classifiers. Each classifier is trained on samples from a pair of classes. When making a prediction on an unlabeled image, each of the $K(K-1)/2$ classifiers are given the image, and then "vote" on which class the image belongs to. That is, the class that gets the highest number of positive predictions among all classifiers' predictions will be taken as the "winner".

We perform a number of different rounds of this multi-class classification task to compare different Kernel methods. Specifically, we train with a linear kernel (the same as a linear classifier), a polynomial kernel of degree two and degree 3, and finally with an RBF kernel with $\sigma = 1$ and $\sigma = 4$. For each of these the soft-margin penalty C is set to 1.

We also perform feature extraction using Histogram of Oriented Gradients (HOG) with cell size of 8x8 on each image, and use the resulting feature vector as the representation of the image on which we train/classify.

4.2 Image Preprocessing/Feature Extraction

In order to map our pictures to points in space, we perform a preprocessing step involving the Histogram of Oriented Gradients (HOG).

HOG results in compressing and encoding images. More specifically, HOG takes an an image gradient at different locations in an image and gives the intensity change in that location as an oriented gradient. For each patch of the image, gradient orientations are computed and then these orientations are then plotted into a histogram. The histogram is akin to placing in different "bins" vectors that correspond to different ranges of orientation, e.g. one bin for vector orientations 0-degrees to 15-degrees, another bin for vector orientations of 15-degrees to 30-degrees, etc. This results in the ability to calculate the probability of existence of a gradient with specific orientation in each patch of the image.

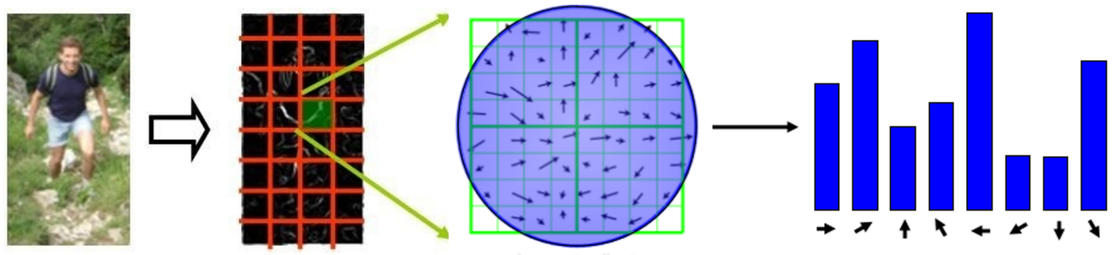


Figure 7: This is an image from [2], not from CIFAR-10, but it does a nice job of illustrating the way that HOG works. The red cells are the “patches” of the image that will result in different histograms. Because there are 4x8 patches, a total of 4x8=32 histograms will be computed. These will then be concatenated with each other to represent the image. The image then, would be represented by a vector in $\mathbb{R}^{32\times 32\times \#\text{bins}}$ space.

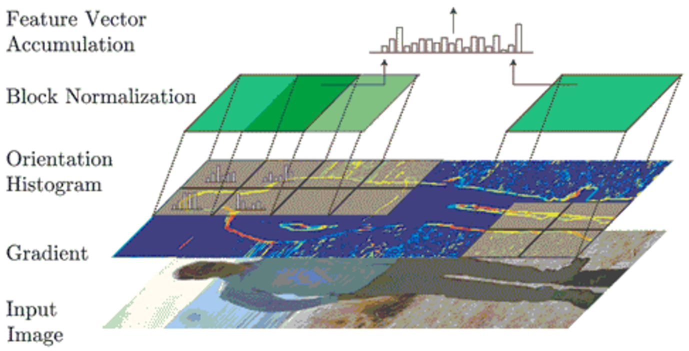


Figure 8: This is also an image from [2] that illustrates the whole process of HOG. An input image is given, gradients (changes in intensity) are found, oriented histograms are computed, block normalization (i.e. binning according to ranges) is performed, and finally accumulation (concatenating all the patches histograms) to represent the image as a single vector.

There is a trade off in the grid size, i.e. the number of patches and the results you can get from HOG. If the grid is too fine, then not only is it more computationally expensive to compute the features, but it might also miss broader structure in the image. However, if the grid size is too course, then local structure can be lost. It is generally best to try different grid sizes and choose one that works best.

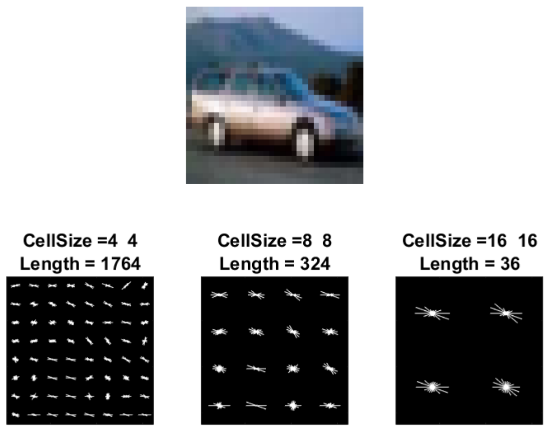


Figure 9: Here is an image from the automobile class of the CIFAR-10 data set with HOG features shown at different levels of granularity, i.e. with cell (patch) sizes of 4x4, 8x8, and 16x16. We choose 8x8 for our experiments because it seems to make a nice trade-off between performance and accuracy.

4.3 Experimental Results

Table 1: Linear Kernel: **Accuracy 25%**

Polynomial (Degree 3)	Airplane	Automobile	Bird	Cat
Airplane	0	0	0	0
Automobile	0	0	0	0
Bird	0	0	0	0
Cat	1000	1000	1000	1000

This linear separator doesn't seem to have performed well at all. In fact, it's accuracy achieves no better than random guessing, and it's clear that the classification decision was simply "classify everything as a cat".

Table 2: Polynomial Kernel (Degree 2): **Accuracy 46.40%**

Polynomial (Degree 3)	Airplane	Automobile	Bird	Cat
Airplane	859	387	328	77
Automobile	8	55	6	2
Bird	2	2	23	2
Cat	131	556	643	919

This confusion matrix tells us that out of the 1000 images of Airplanes, 859 were correctly classified as such; the rest were incorrectly classified: 8 as automobile, 2 as bird, and 131 as cat. Out of the 1000 images of automobiles, 55 were correctly classified as such; the rest were incorrectly classified: 387 as airplane, 2 as bird, and 556 as cat. Similarly we can see that out of the 1000 images of birds, 23 were correctly classified, and 977 were incorrectly classified. Lastly, out of the 1000 images of cats 919 were correctly classified and 81 were incorrectly classified. The overall accuracy is around 46%, which is better than random guessing (which would be 25% accuracy for 4 classes). The airplane and cat classes seems to be preferred in general, with the automobile and bird categories being overwhelmingly misclassified as such.

Table 3: Polynomial Kernel (Degree 3): **Accuracy 75.67%**

Polynomial (Degree 3)	Airplane	Automobile	Bird	Cat
Airplane	782	49	110	73
Automobile	53	889	33	58
Bird	99	27	641	154
Cat	66	35	216	715

This confusion matrix tells us that out of the 1000 images of Airplanes, 782 were correctly classified as such; the rest were incorrectly classified: 53 as automobile, 99 as bird, and 66 as cat. Out of the 1000 images of automobiles, 889 were correctly classified as such; the rest were incorrectly classified: 49 as airplane, 27 as bird, and 35 as cat. Similarly we can see that out of the 1000 images of birds, 641 were correctly classified, and 359 were incorrectly classified. Lastly, out of the 1000 images of cats 715 were correctly classified and 285 were incorrectly classified. The overall accuracy is around 76%. No class seems to be preferred in general, but mis-classifications of one group as another seems to be apparent (e.g. birds misclassified as cats and vice versa).

Table 4: RBF Kernel ($\sigma = 1$): **Accuracy 77.55%**

Polynomial (Degree 3)	Airplane	Automobile	Bird	Cat
Airplane	824	42	108	62
Automobile	32	897	25	41
Bird	96	22	646	162
Cat	48	39	221	735

This confusion matrix tells us that out of the 1000 images of Airplanes, 824 were correctly classified as such; the rest were incorrectly classified: 32 as automobile, 96 as bird, and 48 as cat. Out of the 1000 images of automobiles, 897 were correctly classified as such; the rest were incorrectly classified: 42 as airplane, 22 as bird, and 39 as cat. Similarly we can see that out of the 1000 images of birds, 646 were correctly classified, and 354 were incorrectly classified, most of these as cats at 221. Lastly, out of the 1000 images of cats 735 were correctly classified and 265 were incorrectly classified. The overall accuracy is around 78%, which is highest out of all kernels tested. Classifying correctly birds as such was the poorest performing, mis-classifying them as cats most, followed by airplanes. Perhaps, this is because birds and cats are both animals that could be located in similar environments and airplanes and birds in flight might have similar shapes that causes this. Investigating the misclassified images, their HOG features, and the specifics of the classifier might help elucidate if this hypothesis has any weight.

Table 5: RBF Kernel ($\sigma = 2$): **Accuracy 74.08%**

Polynomial (Degree 3)	Airplane	Automobile	Bird	Cat
Airplane	763	47	92	50
Automobile	66	880	42	66
Bird	119	26	611	175
Cat	52	47	255	709

This confusion matrix tells us that out of the 1000 images of Airplanes, 763 were correctly classified as such; the rest were incorrectly classified: 66 as automobile, 119 as bird, and 52 as cat. Out of the 1000 images of automobiles, 880 were correctly classified as such; the rest were incorrectly classified: 47 as airplane, 26 as bird, and 47 as cat. Similarly we can see that out of the 1000 images of birds, 611 were correctly classified, and 389 were incorrectly classified, most of these as cats at 255. Lastly, out of the 1000 images of cats 709 were correctly classified and 291 were incorrectly classified. The overall accuracy is around 74%, which is the third highest out of all kernels tested. Interestingly, airplanes were more incorrectly classified as birds with a broader variance compared to $\sigma = 1$. Besides this, the number and type of mis-categorizations was fairly close to the RBF Kernel $\sigma = 1$.

5 Conclusions and Final Thoughts

SVMs are very powerful classification tools that are conceptually simple and much more computationally efficient compared to other state-of-the-art classifiers like Neural Nets.

In many cases, in the original feature space there is no linear function that can separate instances into classes. One way of handling this is to embed the points into a different space such that they are linearly separable by e.g. lifting the points into a higher dimensional space.

The problem with the above is that the process of embedding can be extremely computationally expensive, and that, in fact, the benefits of this process don't come from knowing anything about the points in this new space other than their inner product.

Luckily, everything involved in training an SVM model to learn a separating hyperplane as well as classifying unlabeled points requires only knowing the result of such an inner product. So instead of e.g. lifting and computing inner products, a kernel function can be used that can preprocess the results of the same inner product at a substantial computational savings. Even more, the kernel function for e.g. the radial basis function (RBF) allows the SVM to learn a classifier that would require lifting points in the original space into an infinite dimensional space, which is certainly computational intractable.

In this work, the derivation of the dual form of a soft-margin SVM was performed, as well as the derivation for a non-trivial kernel (polynomial degree 3), and other kernel methods were given.

Using these equations, the kernel SVM was implemented in Matlab and applied to the classification task of a subset of images from the CIFAR-10 data set. Results were compared between a linear kernel, polynomial (degree 2 and degree 3) kernel, and RBF ($\sigma = 1$ and $\sigma = 2$) kernel, and results were given and analyzed.

The RBF kernel performed the best in general, but the polynomial degree 3 kernel also performed quite well, all achieving about or above 75% accuracy. The linear kernel performed the poorest, only matching the accuracy of random guessing and in fact just classified all images as cats.

More insight could be gained by looking into the details of mis-classifications, such as looking into the HOG features as well as analyzing the generated hyperplanes and where points lie in the new space.

Accuracy could probably also be improved by using different features other than (or in conjunction with) HOG, and by more tuning of the hyperparameters such as the degree of the polynomial, the σ parameter in RBF, and also in the misclassification penalty function C in the dual objective.

References

[1] Alex Krizhevsky, *The CIFAR-10 dataset*, https://www.cs.toronto.edu/ kriz/cifar.html

[2] Sistu Ganesh, *What are HOG features in computer vision in layman's terms?*, https://www.quora.com/What-are-HOG-features-in-computer-vision-in-laymans-terms

[3] Andrew Ng, *CS229 Lecture Notes - Support Vector Machines*, http://cs229.stanford.edu/notes/cs229-notes3.pdf

[4] Yaser Abu-Mostafa, *Caltech CS156 Lecture 15 - Kernel Methods*, https://www.youtube.com/watch?v=XUj5JbQihlU

[5] Eric Kim, *Kernel Trick*, http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

[6] Ankit K Sharma, *Support Vector Machines without tears (Slide #28)*, https://www.slideshare.net/ankitksharma/svm-37753690

[7] Basilio Noris, *ML Demos (Screenshots)*, http://mldemos.b4silio.com/