COMP3105: Assignment 2

Alexander Breeze 101 143 291
Victor Litzanov 101 143 028

Environment:

- Windows 10 20H2 x64
- Python 3.10.7
- Numpy 1.23.3
- Cvxopt 1.3.0
- Scipy 1.9.1
- Matplotlib 3.6.0

Solutions:

1e) Table 1: Training accuracies with different hyper-parameters

λ	BinDev			Hinge		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
0.001	**0.9994**	**0.5706**	**0.8731**	**0.9977**	**0.6128**	**0.8653**
0.01	**0.9973**	**0.5868**	**0.873**	**0.9972**	**0.6169**	**0.8633**
0.1	**0.992**	**0.569**	**0.8723**	**0.9931**	**0.6087**	**0.8662**
1.0	**0.9861**	**0.5718**	**0.8618**	**0.9898**	**0.6098**	**0.8635**

Table 2: Test accuracies with different hyper-parameters

λ	BinDev			Hinge		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
0.001	**0.98971**	**0.54147**	**0.88156**	**0.99245**	**0.60286**	**0.87408**
0.01	**0.98794**	**0.54901**	**0.88114**	**0.99287**	**0.59722**	**0.87335**
0.1	**0.9828**	**0.54379**	**0.88006**	**0.98856**	**0.60032**	**0.87382**
1.0	**0.97577**	**0.54372**	**0.86745**	**0.98453**	**0.59235**	**0.87117**

The first model is well modelled thanks to the fact that the data is clearly separable by a single straight line. Since we are using a model which only learns a straight line, this data is able

to be modelled with a high degree of precision. The second model is the worst, barely better than 50% accuracy, because it requires a circle to model which our model certainly cannot train. The third model does okay, it seems like it would be best modelled with a cubic line but by looking at an individual plot we can see that our model trains a straight line to go through the line of best fit of the ideal trinomial. This means that the data can be approximately modelled by any polynomial with proper regularization parameter lambda, unlike model 2 which requires a non-functional model (two y values for a single x input, aka not a math function).

The accuracy in the training set decreases as lambda increases, which is due to a larger lambda meaning less overfitting. In the test sets this trend sometimes holds, however sometimes accuracy increases temporarily when increasing lambda. This shows how reducing overfitting in the training set can lead to better performance in the test set. In our case, either the data is too simple or lambda values are not perfectly selected, which leads to this result being only occasionally visible.

The test accuracies for model 1 are slightly worse than the training accuracies, as can be expected from slight bias to the training data. However in this case the difference is greater for accuracies with larger lambdas, e.g. increasing lambda makes the model perform worse in both training and testing, which is odd since a larger lambda should mean the weights are more regularized, e.g. less overfit, resulting in it being more generalized and thus performing better in the training set. The best explanation for this is that our given test and train data is so simple, well-separated, or similar (small variance between data points, very linearly separable) that lambda simply isn't necessary, e.g. overfitting isn't really an issue since the data is so easy to model reliably.

In general, BinDev seems to do slightly better than Hinge for model 1, worse for model 2 and equal for model 3. This seems to indicate that BinDev is slightly better at optimizing, although the difference is minute. From inspecting the plots, it seems that the strategy for model 2 is to make the whole screen one colour. That said, BinDev occasionally has a corner or another colour, which seems to decrease its score by additionally misclassifying some outliers.

2e) Table 3: Training accuracies with different kernels

Kernel	BinDev			Hinge		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
Linear	**0.998**	**0.613**	**0.869**	**0.974**	**0.624**	**0.5**
Poly(d=2)	**1**	**1**	**0.874**	**0.99**	**0.577**	**0.5**
Poly(d=3)	**1**	**1**	**1**	**0.988**	**0.6**	**0.651**

Gauss(σ=1)	**1**	**0.999**	**1**	**0.981**	**0.658**	**0.939**
Gauss(σ=0.5)	**1**	**1**	**1**	**1**	**0.922**	**1**
Gauss(σ=0.1)	**1**	**1**	**1**	**1**	**1**	**1**

Table 4: Test accuracies with different kernels

Kernel	BinDev			Hinge		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
Linear	**0.9915**	**0.5986**	**0.8822**	**0.9818**	**0.6146**	**0.5**
Poly(d=2)	**0.9846**	**0.9746**	**0.881**	**0.9776**	**0.5987**	**0.5**
Poly(d=3)	**0.9794**	**0.9696**	**0.9926**	**0.9776**	**0.6146**	**0.6497**
Gauss(σ=1)	**0.9714**	**0.969**	**0.9969**	**0.9584**	**0.6656**	**0.9379**
Gauss(σ=0.5)	**0.9587**	**0.9619**	**0.9984**	**0.9557**	**0.899**	**0.9963**
Gauss(σ=0.1)	**0.7856**	**0.8442**	**0.9956**	**0.838**	**0.9353**	**0.9968**

The results of the top row (linear model) are the same as in question 1. It handles model 1 very well since model 1 is linearly separable, does badly on model 2 because that requires a circle, and does okay on model 3 since it requires a trinomial but can be mostly approximated by a linear function. We can also see that it tends to do slightly worse on the test set, due to its training weights having a slight bias towards the training set.

The polynomial kernel has results similar to the linear kernel for model 1 (basically perfect) and significantly better on models 2 and 3. Polykernel performs better with a degree of 3, this is likely due to degree 3 allowing for the creation of trinomials (like in model 3) and circles (like in model 2). Once again, it does worse on the test set due to overfitting on the training data.

We can see that both linear and polynomial kernels with a degree of 2 get an accuracy of 0.5 on model 3 with hinge loss, both in training and testing. This is because they cannot model it properly, and so classify all data points the same, resulting in exactly 50% accuracy.

The gaussian kernel has the highest accuracy, in fact with $\sigma = 0.1$ it achieves 100% on all training data. Testing with BinDev it has the highest accuracy with $\sigma = 0.5$, as 0.1 was too

small and thus overfitting. However, with hinge loss σ = 0.1, it actually achieves the best test accuracy. It is difficult to explain these results definitively, since σ affects the kernel function, its result K, and both the value and regularization of both binomial and hinge loss. What we can say is that decreasing σ increases K, as σ acts as the denominator in the gauss function. Then we can say that the losses always stay above 0 and decrease as K increases, meaning that as K grows it begins to only affects the regularization terms $\frac{\lambda}{2}\alpha^{\top}K\alpha$ in both loss functions. This allows us to increase the weights more while finding the minimum, leading to more model overfitting. This can be seen in the training data being more accurate with smaller σ. We can also say that, since binomial is an exponential, the loss function stays above 0 even when $e^{-y_i\cdot(k_i^{\top}\alpha+\alpha 0)} > 1$. This means it doesn't overfit as much on small K's unlike Hinge loss which goes straight to $max() = 0$ once past $-\ y_i \cdot (k_i^{\top}\alpha\ +\ \alpha 0) = 1$, and makes the minimization function act only on the regularization term. This effect reduces as σ decreases, as that increases K which makes the loss of the BinDev approach 0 resulting in both loss functions focusing only on the regularization term. This allows Hinge to overtake BinDev.

3c) To find the best hyperparameters, we used K-cross-validation. The data input is half 4 and half 9, so we split each half (500) into 10 parts (50 each), normalized so each pixel is between 0 and 1. Then, for each lambda and each model from a set of promising possibilities, we train a model on all but part k of each of the 10 parts of each half of the data and test the results of the training on the last k parts in each half. We repeat this k times, once for each value of k, and find the median of all the results. This median is, generally speaking, the expected accuracy given these hyperparameters. We take the median instead of the mean because our values will only be tested once against the test data, so we want to select hyperparameters which are reliably good in an isolated test. Finally we find the hyperparameters with the highest median, and conclude that those are generally the best.

3d) Our best hyperparameters are:

```
lamb = 0.01
kernel_func = lambda X1, X2: polyKernel(X1, X2, 3)
```