Hossein Kargar, Ali Salmani, Nima Roshanzadeh

1 SVM for Classification

1.1

In the linearly separable case, if one of the training samples is removed:(1) If the point is not a support vector, then the margin remains unchanged. (2) If the point is a support vector, then the margin length can become larger and move towards the point which is removed if the point was the only support vector or remain unchanged otherwise. Logistic regression focuses on maximizing the probability of the data. The farther the data lies from the separating hyperplane, the happier LR is as opposed to SVM which tries to explicitly find the maximum margin. If a point is not a support vector, it doesn't really matter. Since LR is a density estimation technique, each point will carry some weight and have some effect on the decision boundary

1.2

The hinge loss is the upper bound on the number of misclassified instances. Here, we choose the hinge loss function as $h(z) = \max(0, 1 - z)$. The primal optimization of SVM is given by

$$\underset{w,\xi_i}{\text{minimize}} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

Now the slack variable appears in 1 constraint and we try to minimize that, i.e., we satisfy one of the two constraints given, i.e.,

$$y_i\left(w^T x_i\right) \geq 1 - \xi_i$$

or

$$\xi_i \geq 0$$

The slack variable is the tighter/larger one of the two numbers. So it can either be zero or $1 - y_i\left(w^T x_i\right)$ Thus,

$$\xi_i = \max\left(0, 1 - y_i\left(w^T x_i\right)\right)$$

which is the hinge loss function. Hence,

$$\xi_i = \max\left(0, 1 - y_i\left(w^T x_i\right)\right) = h\left(y_i\left(w^T x_i\right)\right)$$

And we know that the hinge loss is the upper bound on the number of misclassified instances. Thus, the upper bound is given by

$$\implies \sum_{i=1}^{n} h\left(y_i\left(w^T x_i\right)\right)$$

or simply

$$\sum_{i=1}^{n}\left(\xi_i > 1\right)$$

1.3

C is the trade-off parameter that tells us whether we would rather have a small norm of w, meaning a large margin, or rather have no violations of the margin constraints, meaning the small sum of hinge loss. (1) When $C \to 0$, more emphasis is given to finding the largest margin irrespective of several noises, i.e. no misclassification will be penalized. (2) When $C \to \infty$, we put a higher and higher weight on violations of margin constraints, so we find a hyperplane where the required slack is minimized even at the expense of the margin.

1.4

When two classes are linearly separable, both Hard SVM and Logistic Regression can always find a solution. The major difference is that Logistic Regression finds a decision boundary that maximizes its likelihood function while Hard SVM finds a decision boundary with a maximal margin.

1.5

When the two classes are not linearly separable, Logistic Regression will still find a decision boundary that maximizes its likelihood function. For Soft SVM, it will find a decision boundary that best balances the margin and errors. (Note: the key difference between SVM and Logistic Regression is that SVM is more of a geometric-motivated model while Logistic Regression is more of a probability-motivated model.)

2 Composing Kernel Functions

2.1

Since $K^{(1)}$ is valid kernel so we have $z^T K^{(1)} z \geq 0$, Therefore, for any $c > 0$, we have $\mathrm{z}^\top \left(c\, \mathrm{K}^{(1)}\right) \mathrm{z} = c\mathrm{z}^\top \mathrm{K}^{(1)} \mathrm{z} \geq 0, \forall \mathrm{z} \in \mathbb{R}^n$. So $c\, \mathrm{K}^{(1)}$ is also a valid kernel.

2.2

We have $z^\top \left(K^{(1)} + K^{(2)}\right) z = z^\top K^{(1)} z + z^\top K^{(2)} \mathrm{z}$: Since $\mathrm{z}^\top \mathrm{K}^{(1)} \mathrm{z} \geq 0, \forall \mathrm{z} \in \mathbb{R}^n$ and $\mathrm{z}^\top \mathrm{K}^{(2)} \mathrm{z} \geq 0, \forall \mathrm{z} \in \mathbb{R}^n$ therefore $\mathrm{z}^\top \mathrm{K}^{(1)} \mathrm{z} + \mathrm{z}^\top \mathrm{K}^{(2)} \mathrm{z} \geq 0$ so $K^{(1)}(x, x') + K^{(2)}(x, x')$ is a valid kernel.

2.3

We rewrite the equation as follows:

$$\begin{aligned} K(x, x') = K^{(1)}(x, x') K^{(2)}(x, x') &= \left(\sum_{m=1}^{M} \Phi_m^1(x) \Phi_m^1(x')\right) \left(\sum_{n=1}^{N} \Phi_n^2(x) \Phi_n^2(x')\right) \\ &= \sum_{m=1}^{M} \sum_{n=1}^{N} \left[\Phi_m^1(x) \Phi_n^2(x)\right] \left[\Phi_m^1(x') \Phi_n^2(x')\right] \end{aligned}$$

We can define $\Phi_{mn}^3(x) = \Phi_m^1(x) \Phi_n^2(x)$ therefore we have:

$$\begin{aligned} \sum_{m=1}^{M} \sum_{n=1}^{N} \left[\Phi_m^1(x) \Phi_n^2(x)\right] \left[\Phi_m^1(x') \Phi_n^2(x')\right] &= \sum_{m=1}^{M} \sum_{n=1}^{N} \left[\Phi_{mn}^3(x)\right] \left[\Phi_{mn}^3(x')\right] \\ &= \Phi^3(x)^T \Phi^3(x') \end{aligned}$$

Now since we can write the above kernel as an inner product using feature map Φ^3 we can conclude that the kernel is valid.

2.4

2.4.1 We can define a feature $\mathrm{map}\, \Phi(x) = \left(\frac{x^0}{\sqrt{0!}}, \frac{x^1}{\sqrt{1!}}, \frac{x^2}{\sqrt{2!}}, \ldots\right)$ then we have:

$$\Phi(x)^T \Phi(x') = 1 + xx' + \frac{(xx')^2}{2!} + \ldots + \frac{(xx')^i}{i!} = \exp(xx')$$

Since the kernel function is an infinite-dimensional vector it is not suitable, since the dot product of two infinite-dimensional vectors is not tractable. 2.4.2 We can expand $\exp$ like previous part as :

$$\exp(K) = 1 + K + \frac{K^2}{2} + \frac{K^3}{6} + \ldots$$

Since K^1 is valid and we know by previous parts that addition and multiplication of valid kernels are also valid, the above equation concludes that $\exp\left(K^1\right)$ is valid too.

2.5

We can rewrite as:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2}\right)$$
$$= \exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right)\exp\left(\frac{x^T y}{\sigma^2}\right)\exp\left(-\frac{\|x'\|_2^2}{2\sigma^2}\right)$$

We know that $\|x\|_2^2, x^T y, \|x'\|_2^2$ are dot products and they are valid kernels by definition. By other properties we learned from previous parts, we know that addition, exp, and multiplication of valid kernels are valid which results in the above equation being valid too.

3 K-fold Cross-Validation

3.1

It divides the dataset into k parts or folds then trained on $\mathrm{k}-1$ fold and validated on the remaining fold. We do this k times where each fold is used for validation exactly once. $3.2\mathrm{k}$-fold cross-validation VS validation set approach VS LOOCV

3.2

Advantage: k-fold uses all datasets for training and testing while in the validation set approach, only a portion of data is used for testing. In K-fold, we get more reliable results since we average across multiple results.
K-fold is computationally less expensive since it requires fewer iterations than LOOCV.
Disadvantage: K-fold can be computationally expensive, especially for large datasets.
k-fold may not be suitable for small datasets since it uses a small amount of data that may cause high variance in the performance.

3.3

In this approach, we randomly select (without replacement) some fraction of your data to form the training set, and then assign the rest of the points to the test set. And we repeat this N times.
The difference between MCCV and K-fold is that some data points may appear in test sets multiple times.

3.4

Pros: MCVV uses multiple random splits of the data for training and testing, which can reduce bias and variance.
Cons: MCVV can be more computationally expensive than k-fold cross-validation, as it requires multiple random splits of the data

4 Hyperparameter Optimization

4.1

C constant in SVM, Learning rate, Number of epochs Some bad things that may happen are: overfitting, underfitting, slow training, bad generalization

4.2

Model parameters are optimized during training by the model itself, but hyperparameters are optimized by trial and error or by other heuristic methods or manually.

4.3

Choosing appropriate hyperparameters, designing specific loss functions, feature engineering can help us to incorporate our knowledge or insight into an optimization problem

5 Decision Tree

5.1

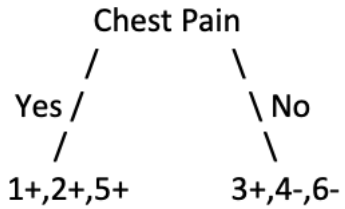
Chest Pain: Yes: $\frac{3}{6} * [-\frac{3}{3} * \log(\frac{3}{3}) - \frac{0}{3} * \log(\frac{0}{3})]$ No: $\frac{3}{6} * [-\frac{1}{3} * \log(\frac{1}{3}) - \frac{2}{3} * \log(\frac{2}{3})]$ Total: 0 + 0.47 = 0.47

Male: Yes: $\frac{4}{6} * [-\frac{2}{4} * \log(\frac{2}{4}) - \frac{2}{4} * \log(\frac{2}{4})]$ No: $\frac{2}{6} * [-\frac{2}{2} * \log(\frac{0}{2}) - \frac{2}{3} * \log(\frac{2}{3})]$ Total: 0.66 + 0 = 0.66

Smokes: Yes: $\frac{4}{6} * [-\frac{3}{4} * \log(\frac{3}{4}) - \frac{1}{4} * \log(\frac{1}{4})]$ No: $\frac{2}{6} * [-\frac{1}{2} * \log(\frac{1}{2}) - \frac{2}{3} * \log(\frac{2}{3})]$ Total: 0.53 + 0.33 = 0.86

Exercises: Yes: $\frac{4}{6} * [-\frac{2}{4} * \log(\frac{2}{4}) - \frac{2}{4} * \log(\frac{2}{4})]$ No: $\frac{2}{6} * [-\frac{2}{2} * \log(\frac{0}{2}) - \frac{2}{3} * \log(\frac{2}{3})]$ Total: 0.66 + 0 = 0.66

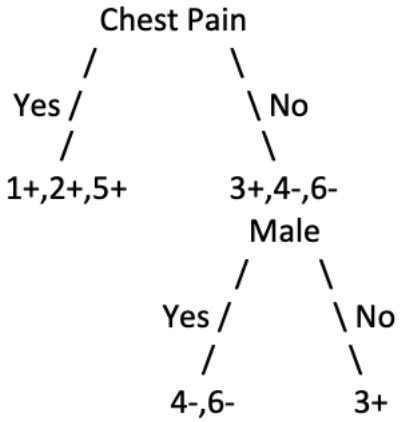
So chest pain is the root node.



Now we will split "No" branch.

Male: Yes: $\frac{2}{3} * [-\frac{0}{2} * \log(\frac{0}{2}) - \frac{2}{2} * \log(\frac{2}{2})]$ No: $\frac{1}{3} * [-\frac{1}{1} * \log(\frac{1}{1}) - \frac{0}{1} * \log(\frac{0}{1})]$ Total: 0 + 0 = 0

Since the minimum possible entropy is 0 and Male has that minimum possible entropy, it is selected as the best attribute.



Each branch ends in a homogeneous set of examples.

5.2

Based on the tree that we constructed, the answer is "Yes".

6 AdaBoost Algorithm

6.1

We know that we use weak learning assumptions in Adaboost, in the sense that there is a classifier h in each step whose error is less than 0.5 according to the classification of that stage. Now we try to find the error h_t in step t+1. (That is, according to D_{t+1}.)

$$\begin{aligned}\widehat{R}_{D_{t+1}}(h_t) &= \sum_{i=1}^{m} \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t} 1_{y_i h_t(x_i)<0} \\ &= \sum_{y_i h_t(x_i)<0}^{m} \frac{D_t(i)e^{\alpha_t}}{Z_t} \\ &= \frac{e^{\alpha_t}}{Z_t} \sum_{y_i h_t(x_i)<0}^{m} D_t(i) \\ &= \frac{\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}}{2\sqrt{\epsilon_t(1-\epsilon_t)}}\epsilon_t = \frac{1}{2}\end{aligned}$$

Now because its error became 0.5, as a result, this classifier can't be selected in step $t+1$.

6.2

We dot product these two vectors together:

$$\sum_{i=1}^{m} D_{t+1}(i) y_i h_t(x_i) = \sum_{i=1}^{m} \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)} y_i h_t(x_i)}{Z_t} = \frac{1}{Z_t}\frac{dZ_t}{d\alpha_t}$$

where z_t is the normalization factor. Be careful at every step we used to choose α_t so that it is minimum compared to z_t. As a result, the above equality holds. So the dot product of these two vectors is zero.