HOMEWORK 6

Devansh Goenka
Student ID: 908 335 1354

Answers

Answer 1.1

We have the following kernel function defined over $z, z' \in Z$:

$$k(z, z') = \begin{cases} 0 & \text{if } z \neq z' \\ 1 & \text{if } z = z \end{cases}$$

Now, for any $m > 0$, we have $z_1, z_2, ...z_m \in Z$. The kernel matrix of this (the inner product space) is defined as $K = [K_{ij}]$. Let's say for $m = 2$, we have K as :

$$K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

This is because $k(z_1, z_1) = 1, k(z_2, z_2) = 1$ and it is 0 for all other values. In fact, for any value of $m > 0$, we have $K = \mathbf{I}_m$

Now, to prove if K is positive semi-definite, we require that $v^\top Kv \geq 0$ given $\forall v \in \mathbb{R}^m$.
We know that $K = \mathbf{I}_m$, thus for any $v \in \mathbb{R}^m$, we have $v^\top K = v^\top \mathbf{I}_m = v^\top$

Therefore, we have $v^\top Kv = v^\top v = ||v||^2$, which is always ≥ 0

Thus, we can say that:

$$\forall v \in \mathbb{R}^m, v^\top Kv \geq 0$$

Hence, the kernel matrix K is positive semi-definite.

Answer 1.2

We know that the label prediction given by the predictor f is $\mathtt{sgn}(f(z))$. Now for any input $z \in Z$, the training set, the predictor takes the form:

$$f(z) = \sum_{i=1}^{n} \alpha_i y_i k(z_i, z) + b$$

Since we sample z from the training set, there will be exactly one instance when $z_i = z$. From the property of the kernel defined above, we see $k(z_i, z) = 1$ only when $z_i = z$. Thus, for any instance $z_j \in Z$ the training set, we have the predictor as :

$$f(z_j) = \sum_{i=1}^{n} \alpha_i y_i k(z_i, z_j) + b$$

$$f(z_j) = \alpha_j y_j + b$$

Now, we can write b in terms of any support vector (x_k^{sv}, y_k^{sv}) as :

$$b = y_k^{sv} - \sum_{i=1}^{m} \alpha_i^{sv} y_i^{sv} k(x_i^{sv}, x_k^{sv})$$

Which then, using the property of our kernel above, reduces to:

$$b = y_k^{sv} - \alpha_k^{sv} y_k^{sv}$$

Plugging this term into the predictor, we get:

$$f(z_j) = \alpha_j y_j + y_k^{sv} - \alpha_k^{sv} y_k^{sv}$$

Now, if we assume a case where all our training points are support vectors, then we can define this bias in terms of that support vector, since the bias is constant but for all support vectors. $y_i^{sv}(w^\top x_i^{sv} + b) = 1$. Thus, y_k^{sv} is actually y_j. Hence:

$$f(z_j) = \alpha_j y_j + y_j - \alpha_j y_j = y_j$$

Thus,

$$\texttt{sgn}(f(z_j)) = \texttt{sgn}(y_j)$$

The above assumed condition will be true whenever all the training points are support vectors or more formally : $\alpha > 0, \forall \alpha \in \{\alpha_1, \alpha_2, ...\alpha_n\}$. Hence, whenever this is true, the kernel k induces a feature space that any training set can be perfectly separated in it.

Answer 1.3

Again, we know that the label prediction given by the predictor f is $\texttt{sgn}(f(z))$. Now for any input not in the training set, the predictor takes the form:

$$f(z) = \sum_{i=1}^{n} \alpha_i y_i k(z_i, z) + b$$

Where z is the input for which the label needs to be predicted.
Now, since $k(z, z') = 1$ only when $z = z'$, and by definition the input z will never be a part of the training set $z_i \in Z$, we can say that $\forall z \notin Z$ and $z_i \in Z, k(z_i, z) = 0$.
Therefore, our predictor reduces to:

$$f(z) = \sum_{i=1}^{n} \alpha_i y_i * 0 + b = b$$

We can further expand this b, which is our bias term as follows:

$$b = y_j^{sv} - \sum_{i=1}^{m} \alpha_i^{sv} y_i^{sv} k(x_i^{sv}, x_j^{sv})$$

Where m is the numer of support vectors and (x_j^{sv}, y_j^{sv}) is any support vector and its label. Using the property of our kernel this further reduces to :

$$b = y_j^{sv} - \alpha_j^{sv} y_j^{sv} = y_j^{sv}(1 - \alpha_j^{sv})$$

Thus, for any test input not belonging to the training set, the predictor produces a fixed prediction which is given by $\texttt{sgn}(b) = \texttt{sgn}\left(y_j^{sv}(1 - \alpha_j^{sv})\right)$.

This label finally depends on the value of α_j^{sv}, and is a fixed prediction for any test input.

Answer 2.1

The implementation for Linear SVM, Kernel SVM, Logistic Regression, Kernel Logistic Regression and Neural Network are in the attached zip file.
Linear SVM is indeed just a special case of Kernel SVM, where the kernel represents the inner product $\langle x_i, x \rangle$ of the feature vectors directly without transformation into a higher dimensional space.

Answer 2.2.1

Using the two mutltivariate Gaussians, here are the results on the different classifers: [Note: The blue points indicate distribution of class -1 and red points indicate distribution of class +1]

Linear SVM
Test accuracy on 250 points = 99.2%
Decision Boundary:

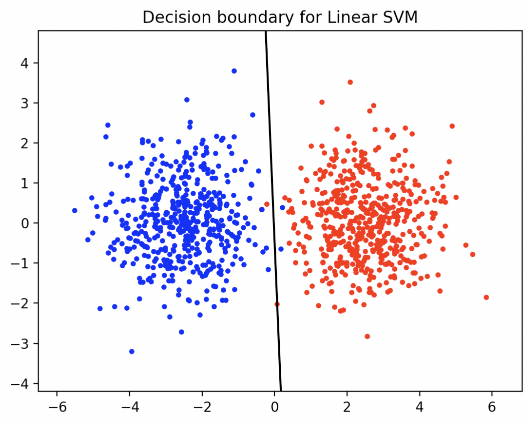

Figure 1: Decision Boundary for Linear SVM

Logistic Regression
Hyperparams: Epochs = 1500, Learning Rate = 0.3, Batch Size = 20
Test accuracy on 250 points = 98.8%
Decision Boundary:

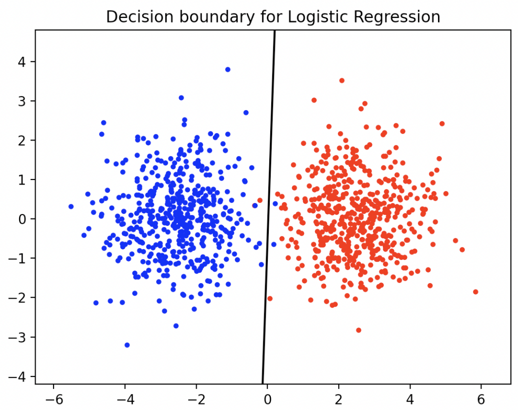

Figure 2: Decision Boundary for Logistic Regression

K Nearest Neighbors
Hyperparams: k = 15
Test accuracy on 250 points = 98.8%
Decision Boundary:

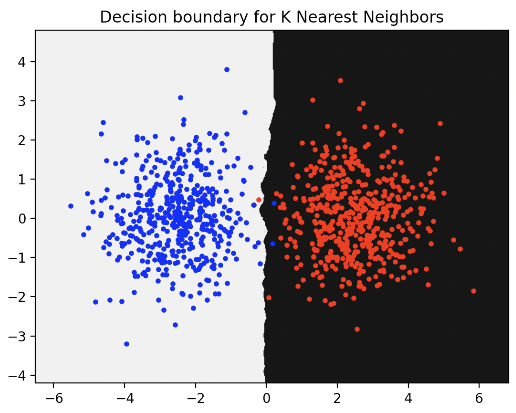

Figure 3: Decision Boundary for kNN with k=15

Naive Bayes
Likelihood : Gaussian
Test accuracy on 250 points = 98.8%
Decision Boundary:

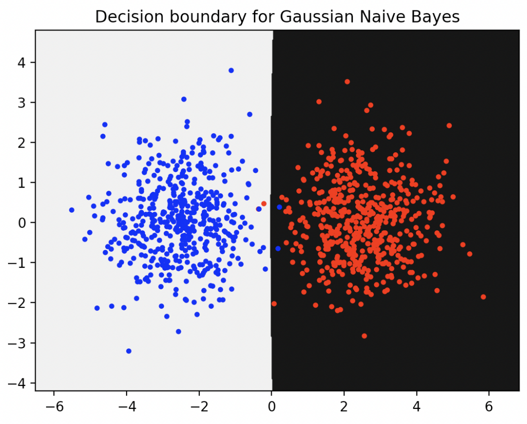

Figure 4: Decision Boundary for Gaussian Naive Bayes

After varying μ of the Gaussian 1.0 to 2.4, here are the test accuracies for each of the above four classifiers:

Linear SVM

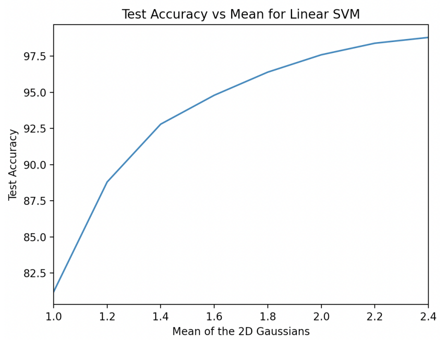

Figure 5: Test Accuracy vs μ for Linear SVM

Logistic Regression

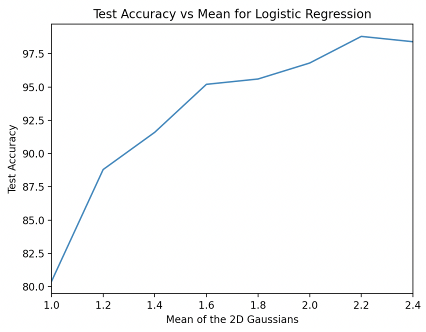

Figure 6: Test Accuracy vs μ for Logistic Regression

K Nearest Neighbors

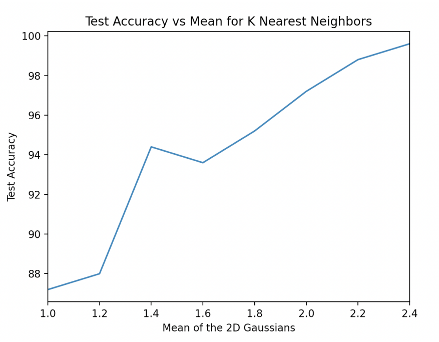

Figure 7: Test Accuracy vs μ for kNN

Naive Bayes

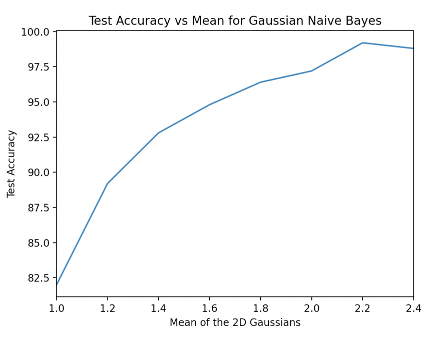

Figure 8: Test Accuracy vs μ for Gaussian Naive Bayes

Analysis
From the plots, it is evident that these 2D datasets are linearly separable to an extent. Therefore, the linear classifiers such as Linear SVM, Logistic Regression perform very well on this dataset. The non-linear classifiers also perform equally well on this set. Since Linear SVM maximizes the margin, it actually performs slightly better than other classifier.

As we vary the μ of Gaussians, we bring the distributions closer to each other at $\mu = 1.0$, hence the test accuracy of all the classifiers suffer. However, the linear classifiers suffer more than kNN and Naive Bayes, due to their linear decision boundaries. As μ increases, the distributions separate out from each other and become increasingly separable by a linear decision boundary, which increases the accuracies of all the classifiers.

Answer 2.2.2

Using the dataset `sklearn.datasets.make_circles`, which is not linearly separable, here are the results on the various classifiers.

Linear SVM

Test accuracy on 250 points = 49.2%
Decision Boundary:

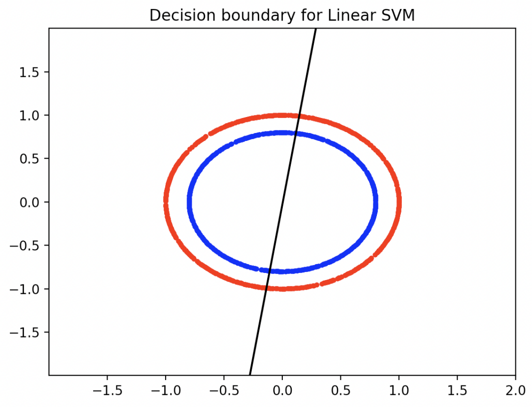

Figure 9: Decision Boundary for Linear SVM

Logistic Regression Hyperparams: Epochs $= 1500$, Learning Rate = 0.3, Batch Size = 20
Test accuracy on 250 points = 52.4%
Decision Boundary:

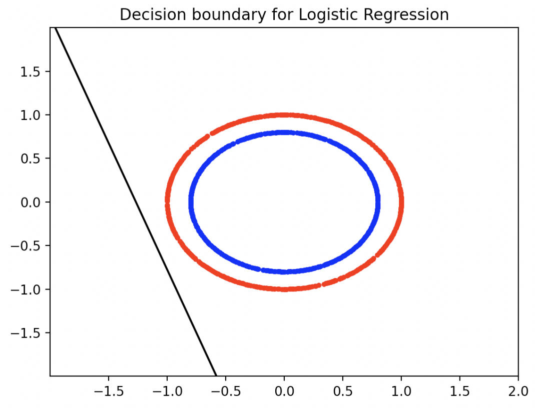

Figure 10: Decision Boundary for Logistic Regression

Polynomial Kernel SVM
Hyperparams: Degree$= 3$
Test accuracy on 250 points = 100%
Decision Boundary:

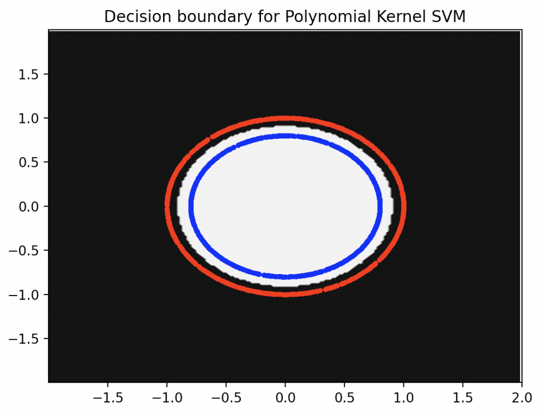

Figure 11: Decision Boundary for Polynomial Kernel SVM

RBF Kernel SVM
Hyperparams: Sigma$= 5.0$
Test accuracy on 250 points = 100%
Decision Boundary:

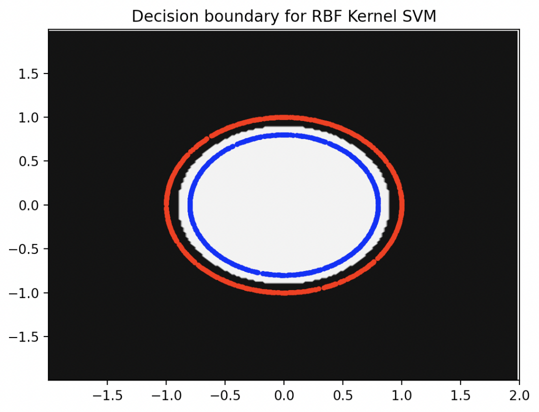

Figure 12: Decision Boundary for RBF Kernel SVM

2 layer Neural Network
Hyperparams: Learning Rate $= 0.03$, Batch Size $= 20$, Epochs $= 500$
Test accuracy on 250 points = 100%
Decision Boundary:

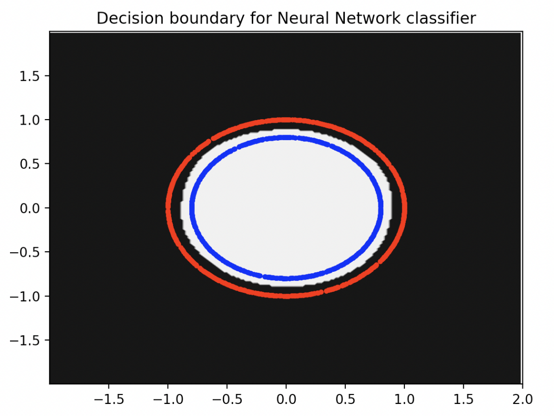

Figure 13: Decision Boundary for 2 layer Neural Network

K Nearest Neighbors Hyperparams: K$= 15$
Test accuracy on 250 points = 100%
Decision Boundary:

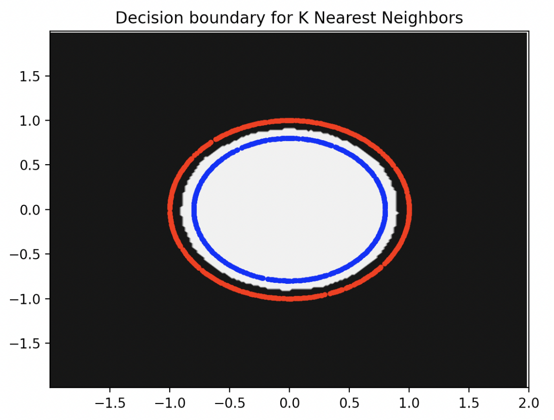

Figure 14: Decision Boundary for KNN

Analysis
It is evident that this dataset is not linearly separable. Therefore, the linear classifiers such as Linear SVM, Logistic Regression suffer (with accuracy almost as bad as random guessing). The non-linear classifiers perform extremely well (achieving 100% accuracy) on this dataset because the points are visually separable by a 2D decision boundary.
Moreover, once we switch over to Kernel SVM, it is now capable of creating a two dimensional decision boundary, and after tuning some hyperparameters, both the RBF and Polynomial Kernel SVM achieve 100% test accuracy on this set.

Answer 2.3

Using the dataset `sklearn.datasets.load_breast_cancer`, the all the above classifiers were trained. The data was first pre-processed by normalizing all the 30 features by their mean and standard deviation, and the same transform was applied to the test set as well.

For all the classifiers, 10% of the data was held out for validation and 10% for the test set.

Linear SVM
Test accuracy on points = $94.7\%(54/57)$
Logistic Regression Hyperparams: Epochs $= 1500$, Learning Rate = 0.3, Batch Size = 20
Test accuracy on points = $94.7\%(54/57)$
Polynomial Kernel SVM
Hyperparams: Degree $= 2$ Test accuracy on points = $96.5\%(55/57)$
RBF Kernel SVM
Hyperparams: Sigma $= 5.0$ Test accuracy on points = $98.2\%(56/57)$
2 Layer Neural Network
Hyperparams: Learning Rate $= 0.03$, Batch Size $= 20$, Epochs $= 500$
Test accuracy on points = $98.2\%(56/57)$
K Nearest Neighbors
Hyperparams: K$= 15$
Test accuracy on points = $98.2\%(56/57)$

Analysis Upon looking at the test accuracies, it is fair to say that this dataset is somewhat linearly separable in the d-dimensional feature space, as the linear models such as Linear SVM and Logistic Regression are almost performing as good as the non-linear models. Moreover, the kernel SVM methods are slightly better, with the RBF kernel SVM having higher accuracy than the polynomial kernel SVM. The improvement over linear models is not that vast though. In fact, the RBF kernel SVM performs on par with KNN and the 2 layer Neural Network.

If we want to figure out the important features, we want to use the L_1 penalty, as it induces sparsity by making the coefficients of those features that are near-zero to be zero. After implementing this L_1 penalty and posing the SVM problem as a unconstrained optimization problem with a hinge loss, we can now solve this using gradient descent or SGD. On implementing the said sparsity, the features that were relevant to the Wisconsin Breast Cancer dataset were:

- 'mean texture'
- 'mean concave points'
- 'mean fractal dimension'
- 'radius error'
- 'smoothness error'
- 'compactness error'
- 'fractal dimension error'
- 'worst radius'
- 'worst texture'
- 'worst area'
- 'worst smoothness'

- ’worst concavity’
- ’worst concave points’
- ’worst symmetry’

Thus, we see, out of 30 features, only 14 were really useful for our classification.