CS5691: Pattern Recognition and Machine Learning

Programming Assignment # 2

Sourav Sahoo, EE17B040

July 28, 2020

Contents

List of Figures

List of Tables

1 Kernel Ridge Regression

Regression is a method of modelling a target value based on independent predictors. This method is mostly used for forecasting and finding out cause and effect relationship between variables. If the mean-square error is chosen as an error metric and a linear relationship between the features and target value is assumed, the objective function is as described in (1).

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 \tag{1}$$

where $\mathbf{y} \in \mathbb{R}^m$ is the target value, $\mathbf{X} \in \mathbb{R}^{m \times n}$ is a matrix with rows being the observed values of the features and $\beta \in \mathbb{R}^n$ is the weight vector that we intend to find. To avoid overfitting, often a regularization term is added. In case of *ridge regression*, the regularization term added is the ℓ_2-norm of the weight vector β. The objective function is as described in (2).

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_2^2 \tag{2}$$

where λ is a positive parameter determining the trade-off between empirical square loss and norm of weight vector. Such a minimization function has a closed form solution which is given in (3).

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} \tag{3}$$

1.1 The Kernel Trick

The kernel trick avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary. For all $\mathbf{x}$ and $\mathbf{x}'$ in the input space $\mathcal{X}$, certain functions $k(\mathbf{x}, \mathbf{x}')$ can be expressed as an inner product in another space $\mathcal{V}$. Mathematically, there exists a function $\phi \colon \mathcal{X} \to \mathcal{V}$ such that

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{V}} \tag{4}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{V}}$ refers to the inner product in space $\mathcal{V}$. This function $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is often referred to the kernel function. Kernel Ridge Regression (KRR) is performing ridge regression using the kernel trick.

1.2 Linear Kernel and Polynomial Kernel

In this problem, we apply the kernel trick for a regression task. To this end, two different kernels namely, *the linear kernel* and *the polynomial kernel* are considered for comparison. The expression for linear and polynomial kernel is given in (5).

$$\begin{aligned} K_l(\mathbf{x}, \mathbf{x}') &= \mathbf{x} \cdot \mathbf{x}' \\ K_p(\mathbf{x}, \mathbf{x}') &= (\gamma(\mathbf{x} \cdot \mathbf{x}') + a)^d \end{aligned} \tag{5}$$

where γ is the kernel factor[1], d is the degree of the polynomial and a is the independent term. The default values of $\gamma = 1/(\#\text{of features})$ and $a = 1$ are always considered unless stated otherwise. In this case, the optimization objective similar to the case of regular linear ridge regression. The exact expression is given in (6).

$$\min_{\mathbf{w}} \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^{m} (\mathbf{w} \cdot \phi(\mathbf{x}_i) - y_i)^2 \tag{6}$$

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Solving the dual of the convex optimization problem in (6), we get the solution as mentioned in (7).

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i) = \mathbf{X}\boldsymbol{\alpha} = \mathbf{X}(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} \tag{7}$$

where $\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}$ and $\mathbf{K} = \mathbf{X}^\top \mathbf{X}$ is the kernel matrix. So, for a any given $\mathbf{x}$, the prediction of the model is given in (8).

$$\forall \mathbf{x} \in \mathcal{X},\ h(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \tag{8}$$

where $K(\cdot, \cdot)$ is the kernel function.

1.3 Experiments and Inferences

The entire dataset $\mathbf{X} \in \mathbb{R}^{506 \times 13}$ is split in 80:20 ratio and KRR is performed. We conducted extensive experiments for choosing the ideal values of λ and the degree, d, of the polynomial kernel. The best performance was observed with $d = 2$ and $\lambda = 10^{-3}$. The results are mentioned in Table 1. From this experiment, we made the following observations:

Table 1: Mean squared error on the train and test data for linear and polynomial kernel

Linear		**Polynomial**	
Train	**Test**	**Train**	**Test**
23.1164	29.2940	**5.8153**	**11.6164**

where the values in bold represent the kernel with best performance i.e. least mean squared error.

- The kernel trick maps the input features into a much higher dimensional space in which linear regression is favourable. This is clearly evident from the fact the second-degree kernel has almost a three times better mean squared error as compared to the linear kernel.
- It is to be noted that even though some higher degree polynomial kernel has much less training error, the generalisation gap is too high. When $d = 2$, the train and test error are much close, i.e., the generalization error is less.
- KRR has a closed form solution due to which many of its properties can be analyzed. However, as the closed form solution has a matrix inversion step, it becomes infeasible to use when the matrix sizes are large.

2 Kernel Logistic Regression and Kernel SVM

2.1 Kernel Logistic Regression

In this section, we perform logistic regression (LR) using the kernel trick. In vanilla logistic regression, the empirical loss to be minimized is given as described in (9).

$$\widehat{\mathfrak{R}}(\mathbf{w}) = -\sum_{i=1}^{m} \log\left(1 + e^{-y_i \mathbf{w} \cdot \mathbf{x}_i}\right) \tag{9}$$

where $(\mathbf{x}_i, y_i)_{i=1}^m$ are the data and label pair. Using the kernel trick, the loss function becomes:

$$\widehat{\mathfrak{R}_K}(\mathbf{w}) = -\sum_{i=1}^{m} \log\left(1 + e^{-y_i \mathbf{w}\cdot\phi(\mathbf{x}_i)}\right) \tag{10}$$

Let $\Phi = [\phi(\mathbf{x}_1)^\top, \ldots, \phi(\mathbf{x}_m)^\top]^\top$ and $\mathbf{K} = \Phi\Phi^\top$, then $\mathbf{w} = \Phi^\top\alpha$, so (10) becomes:

$$\widehat{\mathfrak{R}_K}(\alpha) = -\sum_{i=1}^{m} \log\left(1 + e^{-y_i \alpha^\top \Phi\phi(\mathbf{x}_i)}\right) = -\sum_{i=1}^{m} \log\left(1 + e^{-y_i \alpha^\top \mathbf{K}_{:,i}}\right) \tag{11}$$

where $\mathbf{K}_{:,i}$ is the *ith* column of the kernel matrix. We use gradient descent method to minimize (11). The gradient of the loss function w.r.t. α is given in (12).

$$\nabla_\alpha \widehat{\mathfrak{R}_K}(\alpha) = \sum_{i=1}^{m} y_i \log\left(1 + e^{y_i \alpha^\top \mathbf{K}_{:,i}}\right) \cdot \mathbf{K}_{:,i} \tag{12}$$

We use exponentially decaying learning rate for faster convergence. The gradient descent updates are given in (13).

$$\alpha_{t+1} = \alpha_t - \eta_t \nabla_\alpha \widehat{\mathfrak{R}_K}(\alpha_t) \tag{13}$$

where $\eta_t = \eta_0 \cdot (decay)^t$, $\eta_0 = 0.01$ is the initial learning rate, η_t is the learning rate for t^{th} iteration, $decay = 0.9999$ is the decay rate. We choose $\alpha_0 = \mathbf{0}$ and descent is carried out for 1000 iterations. For faster optimization, we use `JAX` library (Bradbury et al., 2018) for gradient computation.

2.2 Kernel Support Vector Machine

We implement the kernel SVM using `sklearn.svm.SVC` class from `scikit-learn` library (Pedregosa et al., 2011) for both the linear and polynomial kernel. For polynomial kernel, we tested from $d = 0$ to $d = 10$ and chose the best performing value of d. The best performing value of d is 2 and the rest of the parameters use their default values.

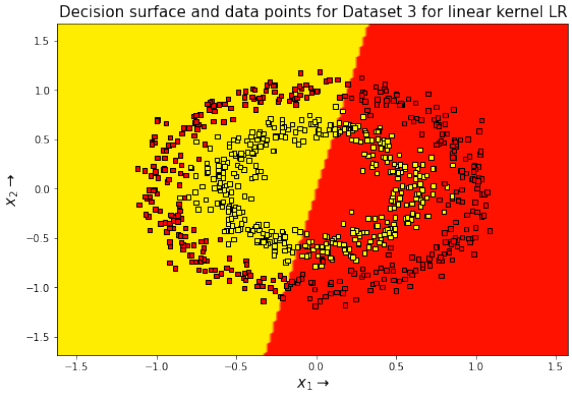
2.3 Experiments and Results

The performance of kernel logistic regression and kernel SVM for both linear and polynomial kernel is mention in Table 2. The decision boundaries and the data points for linear SVM and linear kernel LR is plotted in Figure 1 and for polynomial kernel SVM and LR is given in Figure 2. We plot the decision boundaries and the support vectors of the SVM in Figure 3.

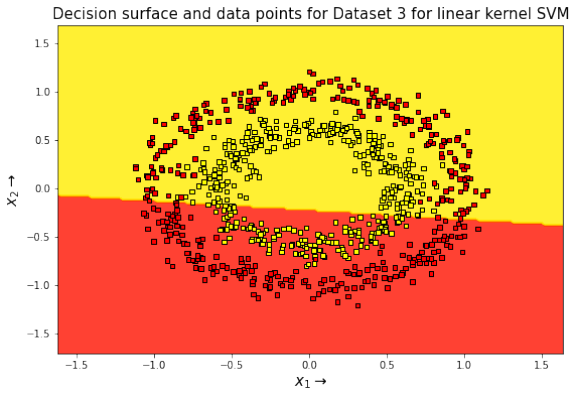
Table 2: Comparison of kernel logistic regression and kernel SVM for different kernels

	Kernel LR		**Kernel SVM**	
Kernel	**Train**	**Test**	**Train**	**Test**
Linear	0.5088	0.4500	**0.5100**	**0.4600**
Polynomial	0.9838	0.9750	**0.9838**	**0.9800**

where the values in bold represent the models with best performance for each kernel. The numbers represent accuracy as a fraction of 1. For polynomial kernel, $d = 2$ for both the models.

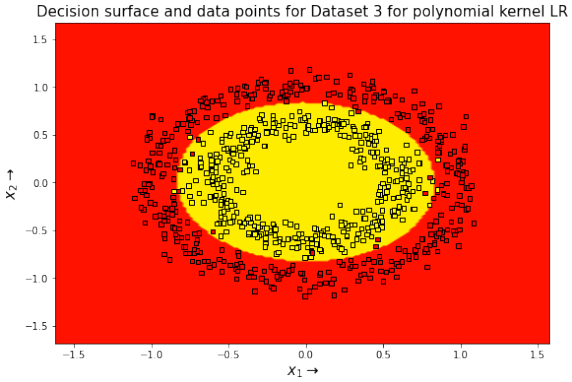


(a) Linear kernel LR

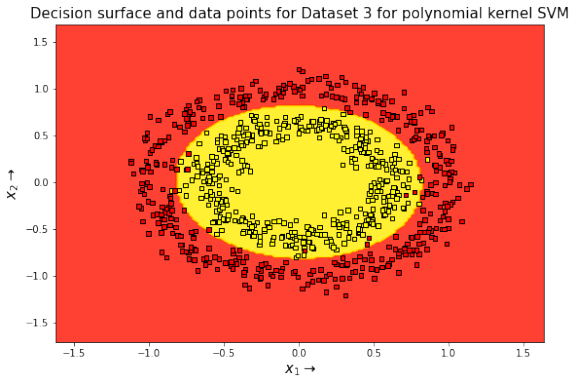


(b) Linear kernel SVM

Figure 1: Decision boundary for linear kernel SVM and kernel LR



(a) Polynomial kernel LR



(b) Polynomial kernel SVM

Figure 2: Decision boundary for polynomial kernel SVM and kernel LR

2.4 Inferences

We note the following inferences from the experiments conducted in this section:

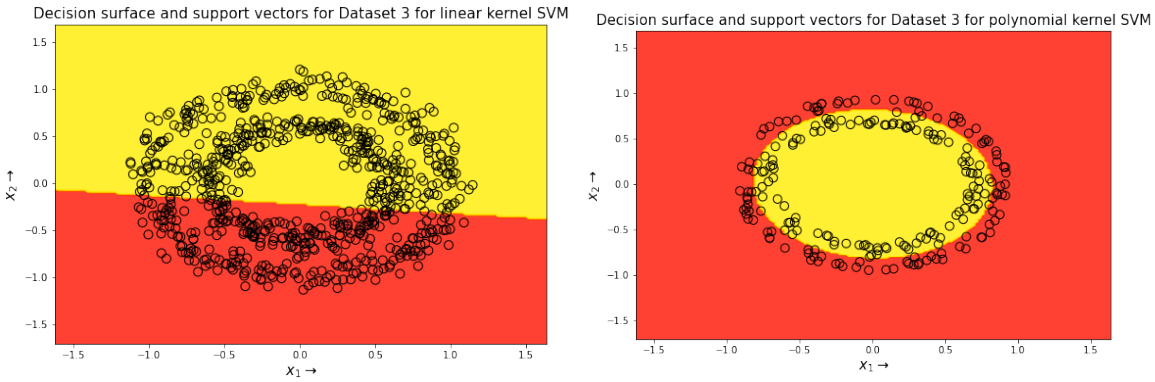
- From Table 2, we observe that linear kernel performs substantially poor (46%) as compared to the polynomial kernel (98%). This can be explained because the dataset is not linearly separable as seen in Figure 1.
- The differences in the performance of SVM and kernel LR are not very significant (45% v 46% in linear kernel and 97% v 98% in polynomial kernel), however, SVM consistently performs marginally better.

3 Soft Margin Support Vector Machine

In this question, a soft-margin SVM is implemented. The dataset is split in the 80:20 ratio and soft-margin SVM is implemented using the `sklearn.svm.SVC` class.

3.1 Effect of C on Train and Test Accuracy

The hyperparmeter C controls the margin width in SVMs. The value of C is varied logarithmically from 10^{-4} to 10^{4}. The train and test accuracies as a function of C is presented



(a) Linear kernel SVM

(b) Polynomial kernel SVM

Figure 3: Decision boundary and support vectors for linear and polynomial kernel SVM

in Table 3. From Table 3, we can observe that the test accuracy saturates for $C \geqslant 1$. However, we consider the *model with* $C = 1$ *as the best performing model* as it has the highest train accuracy for all the discussion hereafter. The confusion matrix for the best performing model is given in Figure 4.

Table 3: Table of train and test accuracy as C is varied in a linear SVM

Value of C	**Train**	**Test**
0.0001	0.5013	0.4950
0.001	0.7988	0.7750
0.01	0.8425	0.8350
0.1	0.8650	0.8450
1	**0.8688**	**0.8550**
10	0.8675	0.8550
100	0.8663	0.8550
1000	0.8663	0.8550
10000	0.8675	0.8550

where the accuracies are presented as a fraction of one and the values in bold represent the SVM model with best performance.

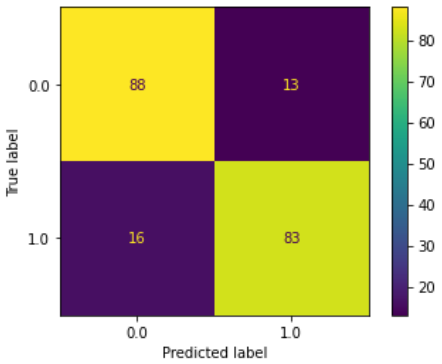


Figure 4: Confusion Matrix for SVM model with $C = 1$

3.2 Decision Boundary Plot and Support Vectors

The decision boundary plots for the best performing model is presented in Figure 5. A plot depicting only the support vectors and margin hyperplanes is further provided in Figure 6.

In case of a hard-margin SVM, all the support vectors lie on the margin hyperplanes, i.e., if $\mathcal{S}$ is the set of support vectors, then, $|\mathbf{w} \cdot \mathbf{x} + b| = 1, \forall \mathbf{x} \in \mathcal{S}$ where $\mathbf{w}$ is the weight vector and b is the intercept. However, for a soft-margin SVM, the above concept fails due to introduction of the slack parameter C. So, we need to explicitly compute the number of support vector lying on margin hyperplanes. Due to finite precision of computers, we consider $\mathbf{x}$ to lie on margin hyperplane if $|\mathbf{w} \cdot \mathbf{x} + b| \in [1 - \epsilon, 1 + \epsilon]$ for some $\epsilon > 0$. For this question, $\epsilon = 0.05$ is chosen. Doing the computation, we find $\boxed{n_+ = 17}$ support vectors for the positive class hyperplane and $\boxed{n_- = 13}$ for the negative class hyperplane.

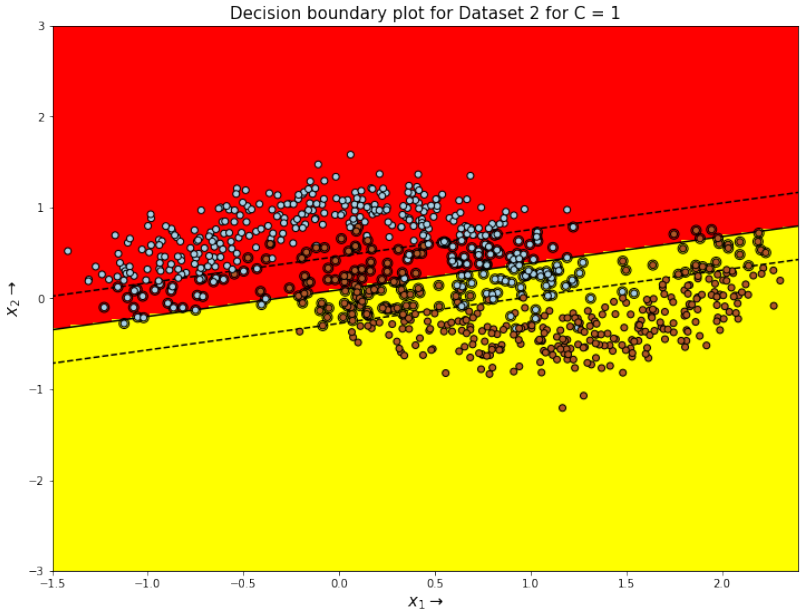


Figure 5: Decision boundary plot along with margin hyperplanes for SVM model with $C = 1$. The support vectors are the circles with double boundaries.

3.3 Weighted Loss Support Vector Machine

In this experiment, we penalize the loss associated with the negative class k times more than the positive class. The plot showing the effect of k on train and test accuracy is shown in Figure 7.

3.4 Inferences

The inferences for the experiments conducted in this section is as follows:

- In the first experiment when we vary C, we observe that for extremely low values of C, the accuracies are extremely poor. This is expected because as $C \to 0$, the model tries to have extremely large margin even if it comes at the cost of high misclassfication error. On the other hand, for high values of C, the margin is extremely small (a hard-margin SVM approximation). As the data is not linearly separable and we are using a linear kernel, after a certain limit, the test accuracy saturates.

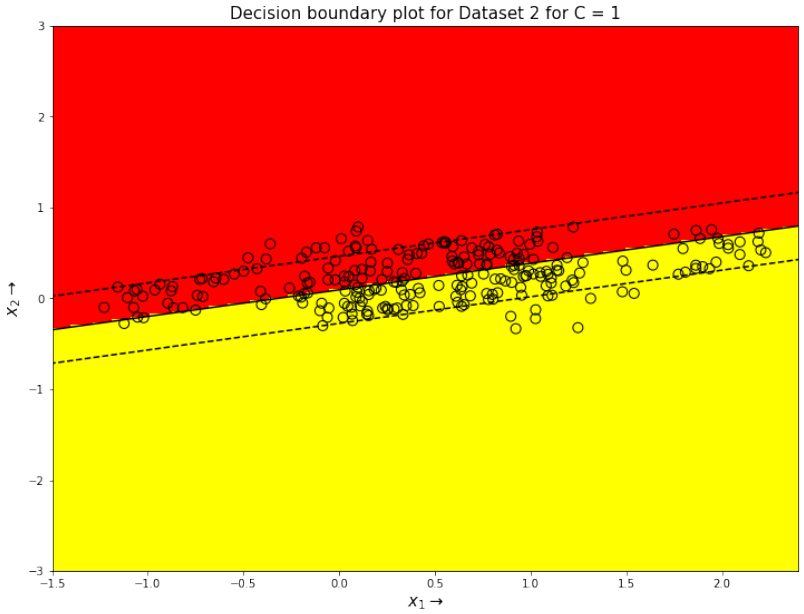


Figure 6: Support vectors along with margin hyperplanes for SVM model with $C = 1$

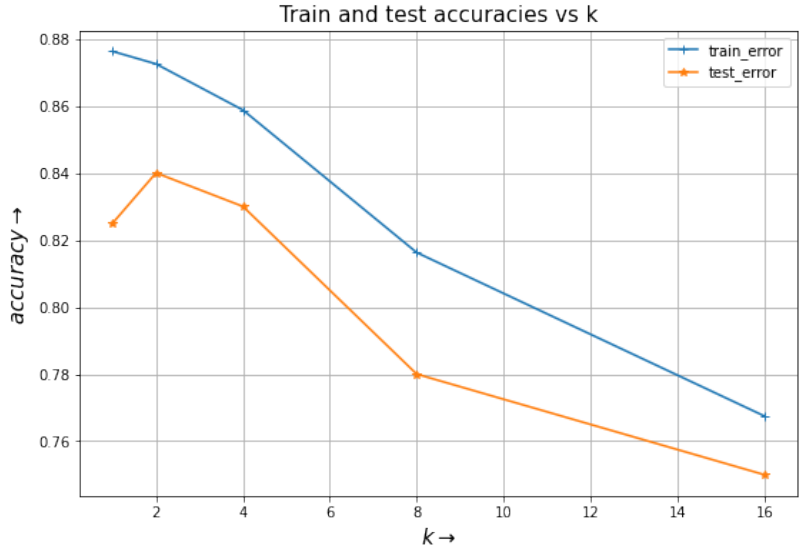


Figure 7: Effect of k on train and test accuracy

- For a soft-margin there are two types of support vectors: 1) the ones that lie on the margin hyperplanes as in case of hard margin SVMs and 2) the vectors satisfying the complementarity conditions $\alpha_i = 0 \vee y_i(\mathbf{w} \cdot \mathbf{x} + b) = 1 - \xi_i$ where $\alpha_i \geqslant 0$ is a Lagrangian variable and ξ_i controls the slack penalty (Mohri et al., 2018, Section 5.3.2).
- In the second experiment, we compute the support vectors of the former type. If we had a linearly separable dataset (such as Dataset 1 used in Section 4) or a non-linear kernel, then the disparity in the total number of support vectors and those lying on margin hyperplanes can reduce.
- In the third experiment, we observe that test accuracy peaks for $k = 2$ after which it steadily decreases (Figure 7). Weighted loss functions are particularly helpful in real-world situations such where weightage of the classes are substantially different.

4 Kernel Perceptron

The perceptron is one the earliest machine learning algorithms. It is an online classification method. In this method, the data points are processed one at a time. The equivalent dual of the problem, also called the dual Perceptron algorithm involves only the inner products of the training data points. Hence, any positive definite symmetric (PDS) kernel can be used for the same which gives rise to Kernel Perceptron.

4.1 The Kernel Perceptron Algorithm

The kernel preceptron algorithm is detailed in Algorithm 1. Here $(x_t, y_t)_{i=1}^T$ are the training data and label pairs and $K(\cdot, \cdot)$ is the PDS kernel.

Algorithm 1: Kernel Perceptron

$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha_0}$; ▷ $\boldsymbol{\alpha_0} = 0$
for $t \leftarrow 1$ *to* T **do**
  RECEIVE(x_t);
  $\hat{y}_t \leftarrow sgn\left(\sum_{s=1}^T \alpha_s y_s K(x_s, x_t)\right)$;
  RECEIVE(y_t);
  if $\hat{y}_t \neq y_t$ **then**
    $\alpha_t \leftarrow \alpha_t + 1$;
  else
    $\alpha_t \leftarrow \alpha_t$;
  end
end

4.2 Experiments and Results

In this question, we experiment with linear and polynomial kernels. The datasets are split into 80:20 train-test split. We initialize $\boldsymbol{\alpha}_0 = \mathbf{0}$, the learning rate is chosen to be 1 and we continue for a maximum of 50 iterations. The train and test accuracies for both the kernels and both the datasets is mentioned in Table 4. The decision boundary along with the training data points are plotted for Dataset 1 in Figure 8 and for Dataset 3 in Figure 9.

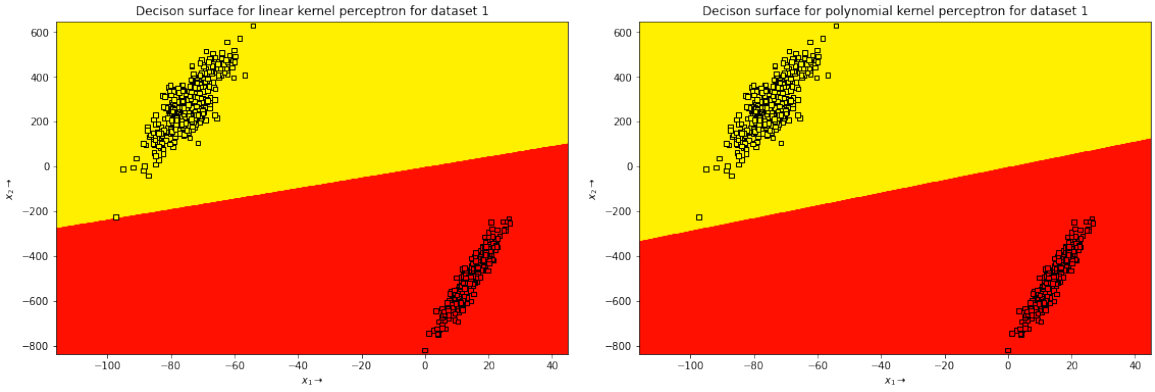
Table 4: Train and test accuracies for Dataset 1 and 3 for Kernel Perceptron algorithm

	Dataset 1		**Dataset 3**	
Kernel	**Train**	**Test**	**Train**	**Test**
Linear	**1.0000**	**1.0000**	0.4913	0.5100
Polynomial	1.0000	1.0000	**0.9813**	**0.9700**

where the values in bold represent the models with best performance for each kernel. The numbers represent accuracy as a fraction of 1. For polynomial kernel, $d = 3$ for both the models.

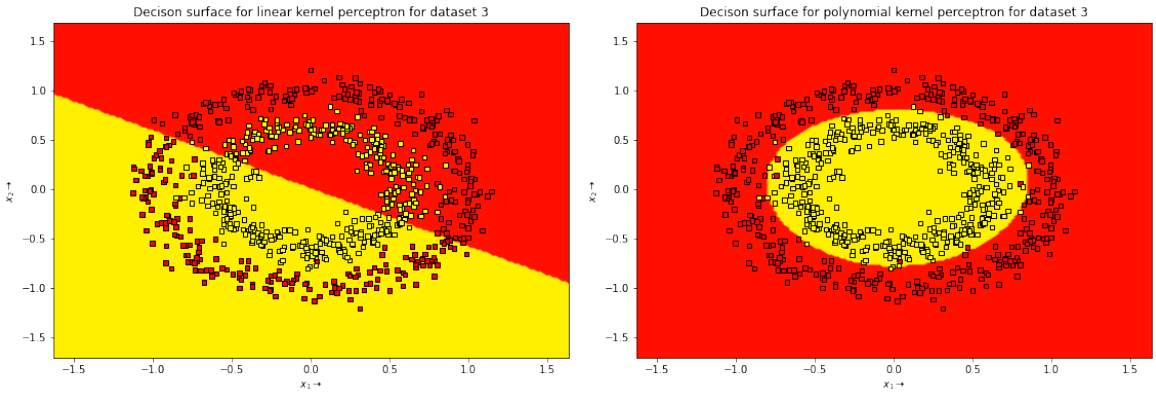
4.3 Convergence analysis of Kernel Perceptron

As described in Mohri et al. (2018), Perceptron converges only for a linearly separable dataset. Hence, we restrict our discussion to Dataset 1 only. If we have *finite size input*, i.e. $\|\mathbf{x}_t\| \leqslant R$, and the data is *linearly separable*, i.e., $\exists \mathbf{w}^* \; s.t. \; y_t(\mathbf{w}^* \cdot \mathbf{x}_t) \geqslant \gamma, \forall t \in [T]$, where γ



(a) Linear kernel perceptron (b) Polynomial kernel perceptron

Figure 8: Decision boundary for Dataset 1 for linear and polynomial kernel perceptron



(a) Linear kernel perceptron (b) Polynomial kernel perceptron

Figure 9: Decision boundary for Dataset 3 for linear and polynomial kernel perceptron

is the margin and the maximum number of mistakes (updates) performed by perceptron is M, then $M \leqslant \lceil (R/\gamma)^2 \rceil$.

We numerically evaluate R for the dataset and γ is computed by using hard margin SVM with linear kernel. We get $R = 818.74$ and $\gamma = 56.69$. So, $M \leqslant 209$. *In our experiments we observe that* $\widehat{M} = 1$ *for linear kernel and* $\widehat{M} = 2$ *for polynomial kernel.* A plausible explanation for such wide disparity could be that the dataset is linearly separable with very large margin as seen in Figure 5, hence the algorithm converges to an optimal solution in very few iterations.

4.4 Comparing Kernel Perceptron with hard margin SVM

The results for kernel perceptron for both the datasets is compared with a hard margin SVM. For implementing SVM, we use the `sklearn.svm.SVC` class. The value of C is set very high to approximate a hard-margin SVM. For all the experiments in this section, $C = 1000$. The comparison results for Dataset 1 and 3 is given in Table 5 and 6 respectively.

4.5 Inferences

We deduce the following inferences in this experiment:

- The model performance is invariant of the kernel chosen for Dataset 1 as it is linearly

Table 5: Comparison of Kernel Perceptron and hard margin SVM for Dataset 1

	Kernel Perceptron		**Hard margin SVM**	
Kernel	**Train**	**Test**	**Train**	**Test**
Linear	**1.0000**	**1.0000**	1.0000	1.0000
Polynomial	**1.0000**	**1.0000**	1.0000	1.0000

where the values in bold represent the models with best performance for each kernel. The numbers represent accuracy as a fraction of 1. For polynomial kernel, $d = 3$ for both the models.

Table 6: Comparison of Kernel Perceptron and hard margin SVM for Dataset 3

	Kernel Perceptron		**Hard margin SVM**	
Kernel	**Train**	**Test**	**Train**	**Test**
Linear	**0.4913**	**0.5100**	0.4088	0.4050
Polynomial	**0.9813**	**0.9700**	0.6275	0.5300

where the values in bold represent the models with best performance for each kernel. The numbers represent accuracy as a fraction of 1. For polynomial kernel, $d = 3$ for both the models.

separable. On the other hand, linear kernel performs very poorly for Dataset 3 (not linearly separable) as seen in Table 4 and Figure 9.

- The perceptron algorithm converges well within the maximum number of iterations in case of Dataset 1. For Dataset 3, the algorithm does not converge as anticipated. However, the number of updates in each iteration for linear kernel is much higher than that of polynomial kernel.
- The performance of perceptron is clearly superior to the hard margin SVM for Dataset 3 (Table 6). For dataset 1, even if the accuracies are exactly same, perceptron takes *much less time to converge* as compared to SVM.

Code Availability

All the experiments in this assignment is done in Python and made available here.

References

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.