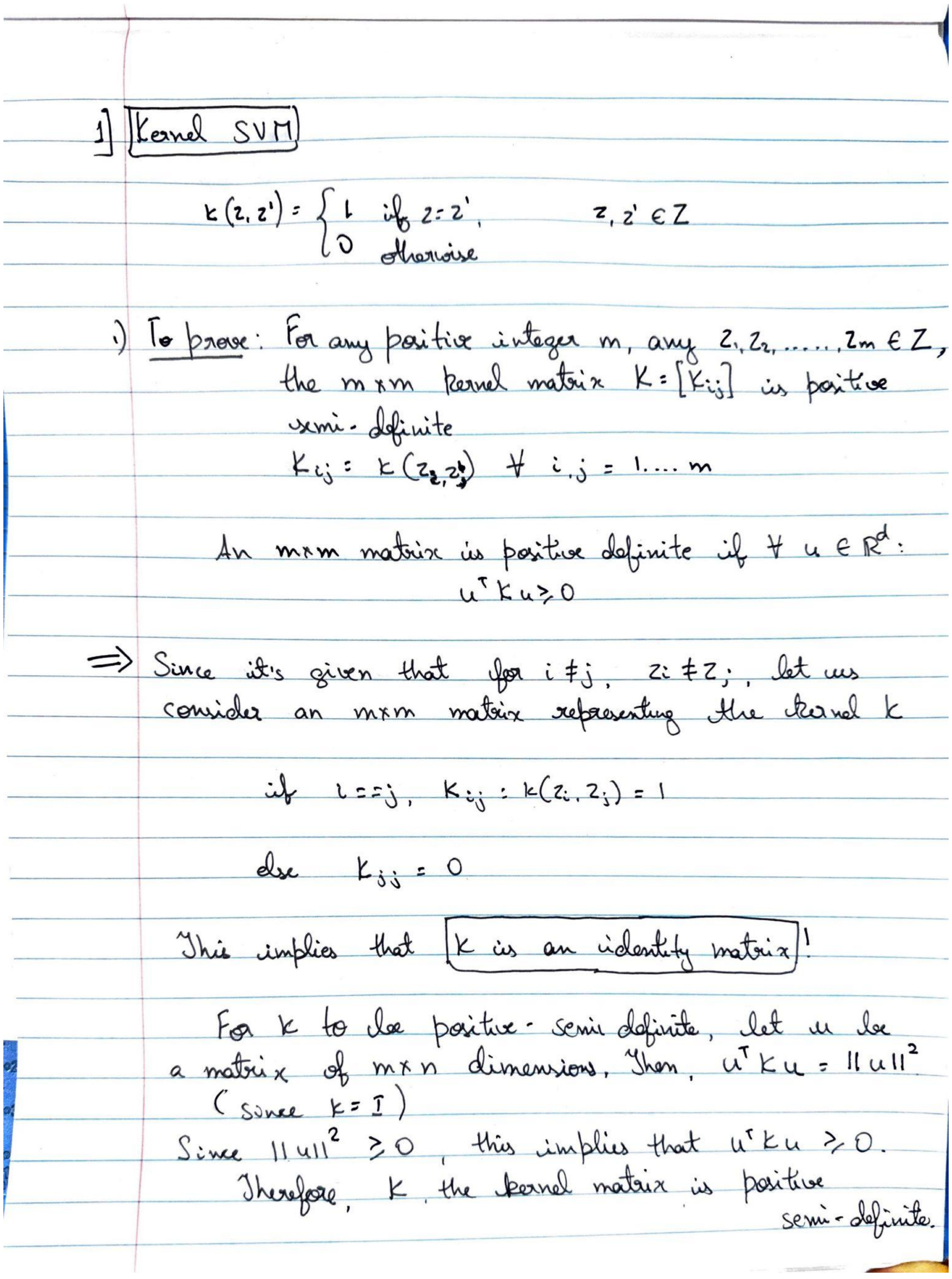
HOMEWORK 6

Hafeez Ali Anees Ali
908 460 3423

1 Kernel SVM [30pts]

1. (10 pts) Prove that for any positive integer m, any $z_1, \ldots, z_m \in Z$, the $m \times m$ kernel matrix $K = [K_{ij}]$ is positive semi-definite, where $K_{ij} = \kappa(z_i, z_j)$ for $i, j = 1 \ldots m$. (Let us assume that for $i \neq j$, we have $z_i \neq z_j$) Hint: An $m \times m$ matrix K is positive semi-definite if $\forall u \in R^d$: $u^\top K u \geq 0$.



1] $\boxed{\text{Kernel SVM}}$

$$k(z, z') = \begin{cases} 1 & \text{if } z = z', \\ 0 & \text{otherwise} \end{cases} \qquad z, z' \in Z$$

i) To prove: For any positive integer m, any $z_1, z_2, \ldots, z_m \in Z$, the $m \times m$ kernel matrix $K = [K_{ij}]$ is positive semi-definite

$K_{ij} = k(z_i, z_j) \quad \forall \; i, j = 1 \ldots m$

An $m \times m$ matrix is positive definite if $\forall \; u \in \mathbb{R}^d$:
$u^T K u \geq 0$

$\Rightarrow$ Since it's given that for $i \neq j$, $z_i \neq z_j$, let us consider an $m \times m$ matrix representing the kernel k

if $i == j$, $K_{ij} = k(z_i, z_j) = 1$

else $K_{ij} = 0$

This implies that $\boxed{\text{K is an identity matrix}}$!

For k to be positive-semi definite, let u be a matrix of $m \times n$ dimensions. Then, $u^T K u = \|u\|^2$ (since $K = I$)

Since $\|u\|^2 \geq 0$, this implies that $u^T K u \geq 0$.

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

2. (10 pts) Given a training set $(z_1, y_1), \ldots, (z_n, y_n)$ with binary labels, the dual SVM problem with the above kernel κ will have parameters $a_1, \ldots, a_n, b \in \mathbb{R}$. (Let us assume that for $i \neq j$, we have $z_i \neq z_j$) The predictor for input z takes the form

$$f(z) = \sum_{i=1}^{n} a_i y_i \kappa(z_i, z) + b .$$

Recall that the label prediction is sgn(f(z)). Prove that there exist $a_1, \ldots, a_n, b$ such that f correctly separates the training set. In other words, κ induces a feature space rich enough such that in it, any training set can be linearly separated.

2) Given training set $(z_1, y_1), \ldots (z_n, y_n)$ with binary labels, to show that the kernel k will have parameters $a_1, \ldots, a_n, b \in \mathbb{R}$.

$\forall\, i \neq j,\ z_i \neq z_j$

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

Label prediction is sign$(f(z))$

$\Rightarrow$ We know that the label $y = 1$ if sign$(f(z)) > 0$ & $y = -1$ if sign$(f(z)) < 0$.

$\forall\ y_i = 1$, we have sign$(f(z_i)) > 0$

$$\Rightarrow \sum_{i=1}^{n} a_i y_i k(z_i, z) + b > 0$$

$$a_i y_i \times 1 + b > 0$$

$$\boxed{a_i + b > 0} \quad \text{—} ①$$

IIIly $\forall\ y_j = -1$, we have sign$(f(z_j)) < 0$

$$\Rightarrow \sum_{i=1}^{n} a_i y_i k(z_i, z) + b < 0$$

$$a_j y_j \times 1 + b < 0$$

$$-a_j + b < 0$$

$$\boxed{a_j > b} \quad \text{—} ②$$

From ① & ②, $\forall\, b \in \mathbb{R}$, there can exist a_is such that $a_i > -b$ & there can exist a_js such that $a_j > b$. Hence, we can show that we have parameters $a_1, \ldots, a_n, b \in \mathbb{R}$ such that $f(z)$ correctly separates the dataset !!

- For a particular case, let's say all a_i & a_j are equal, i.e., $a_i = a_j = a$

Then from ① & ② $a+b > 0$

$a-b > 0$

$\Rightarrow$ For $b = 0$ or some infiteximally small value, we have some solution with $a > 0$! (say $a=1$ & $b = 0.1$)

3. (10 pts) How does that *f* predict input z that is not in the training set?

3) For an input data point not in the training set, we have $k(z_i, z_j) = 0$

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

$$= 0 + b = b.$$

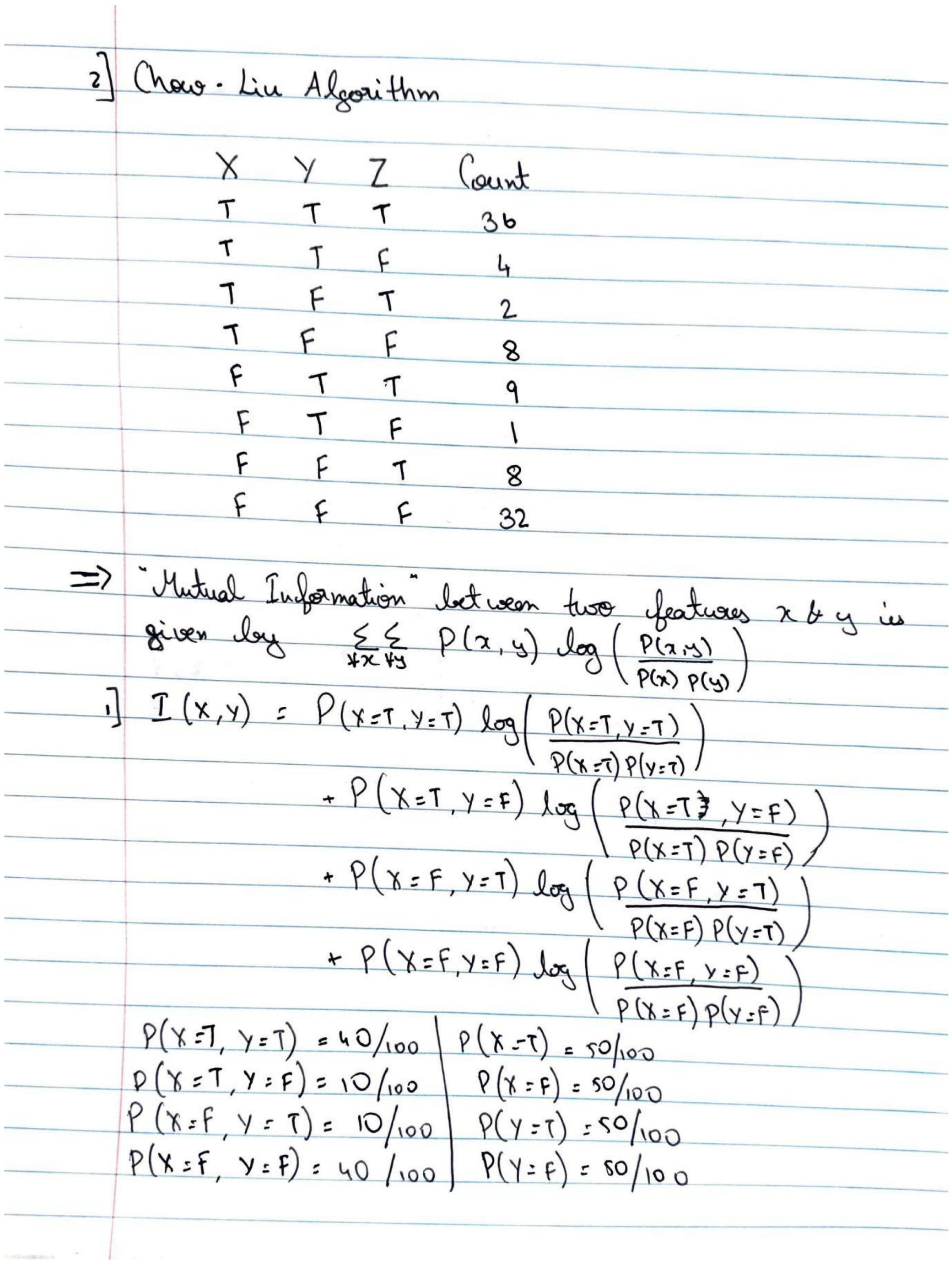
$$f(z) = b$$

This implies that label prediction for some point not in the dataset is given by $\boxed{\text{sign}(b)}$!

2 Chow-Liu Algorithm [30 pts]

Suppose we wish to construct a directed graphical model for 3 features *X*, *Y* , and *Z* using the Chow-Liu algorithm. We are given data from 100 independent experiments where each feature is binary and takes value *T* or *F*. Below is a table summarizing the observations of the experiment:

1. Compute the mutual information *I*(*X, Y*) based on the frequencies observed in the data. (5 pts)



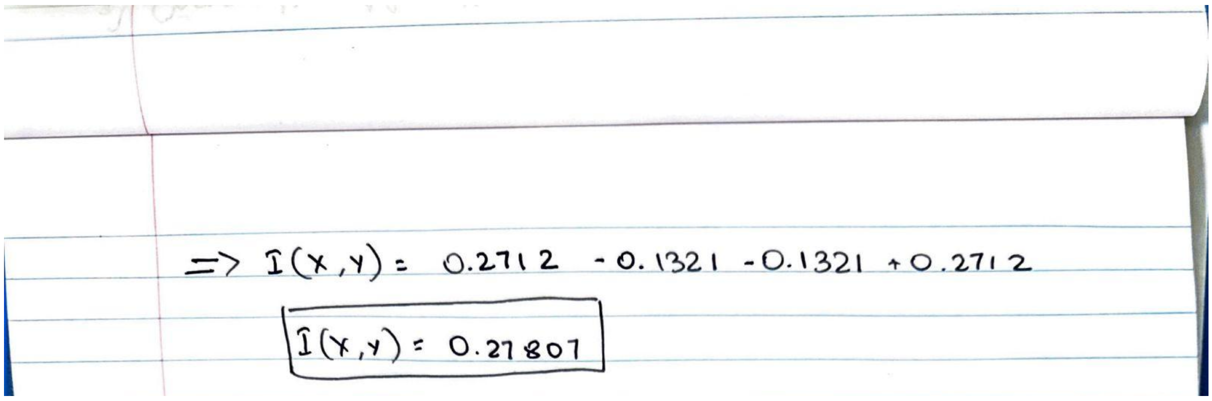
2] Chow - Liu Algorithm

X	Y	Z	Count
T	T	T	36
T	T	F	4
T	F	T	2
T	F	F	8
F	T	T	9
F	T	F	1
F	F	T	8
F	F	F	32

$\Rightarrow$ "Mutual Information" between two features x & y is given by $\sum_{\forall x}\sum_{\forall y} P(x,y) \log\left(\frac{P(x,y)}{P(x)\,P(y)}\right)$

1]
$$I(X,Y) = P(X=T,Y=T)\log\left(\frac{P(X=T,Y=T)}{P(X=T)P(Y=T)}\right)$$
$$+ P(X=T,Y=F)\log\left(\frac{P(X=T,Y=F)}{P(X=T)P(Y=F)}\right)$$
$$+ P(X=F,Y=T)\log\left(\frac{P(X=F,Y=T)}{P(X=F)P(Y=T)}\right)$$
$$+ P(X=F,Y=F)\log\left(\frac{P(X=F,Y=F)}{P(X=F)P(Y=F)}\right)$$

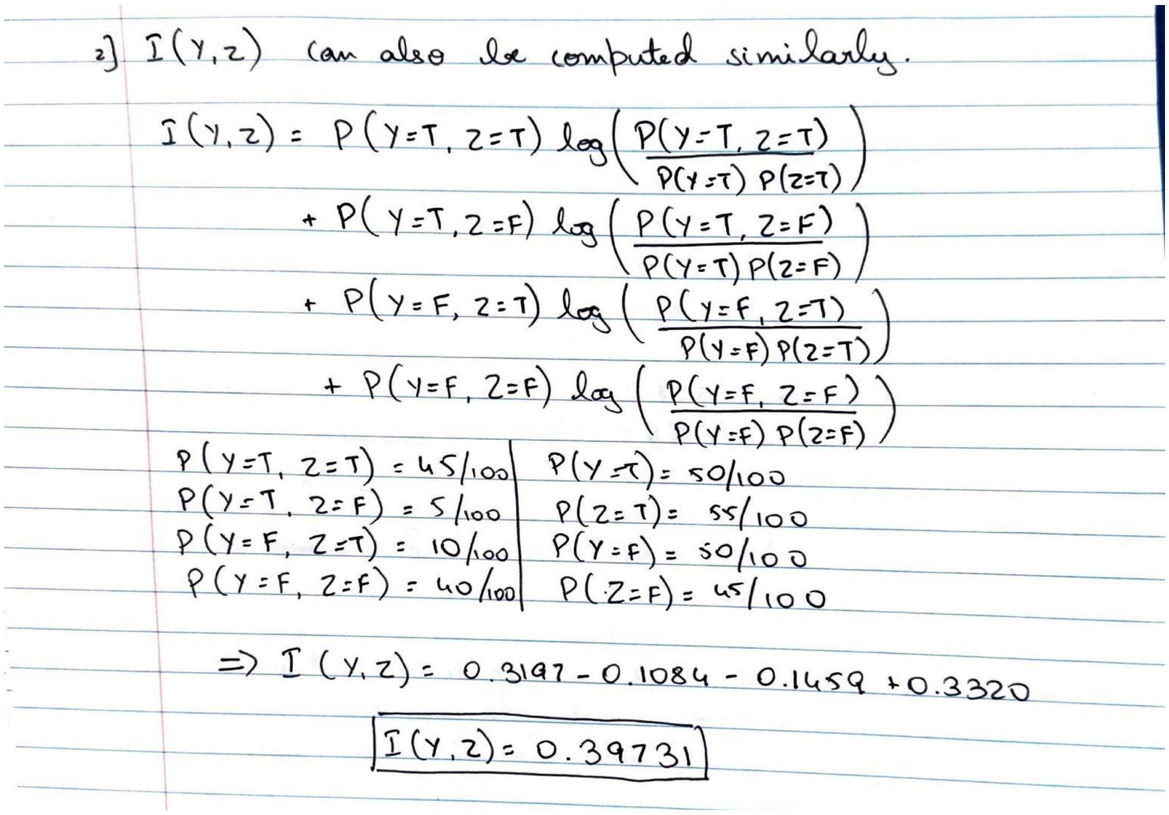
$P(X=T, Y=T) = 40/100$	$P(X=T) = 50/100$
$P(X=T, Y=F) = 10/100$	$P(X=F) = 50/100$
$P(X=F, Y=T) = 10/100$	$P(Y=T) = 50/100$
$P(X=F, Y=F) = 40/100$	$P(Y=F) = 50/100$



$$\Rightarrow I(X,Y) = 0.2712 - 0.1321 - 0.1321 + 0.2712$$

$$\boxed{I(X,Y) = 0.27807}$$

2. Compute the mutual information *I*(*X, Z*) based on the frequencies observed in the data. (5 pts)



2] $I(Y,Z)$ can also be computed similarly.

$$\begin{aligned} I(Y,Z) = {} & P(Y=T, Z=T) \log\left(\frac{P(Y=T, Z=T)}{P(Y=T)\,P(Z=T)}\right) \\ & + P(Y=T, Z=F) \log\left(\frac{P(Y=T, Z=F)}{P(Y=T)\,P(Z=F)}\right) \\ & + P(Y=F, Z=T) \log\left(\frac{P(Y=F, Z=T)}{P(Y=F)\,P(Z=T)}\right) \\ & + P(Y=F, Z=F) \log\left(\frac{P(Y=F, Z=F)}{P(Y=F)\,P(Z=F)}\right) \end{aligned}$$

$P(Y=T, Z=T) = 45/100$	$P(Y=T) = 50/100$
$P(Y=T, Z=F) = 5/100$	$P(Z=T) = 55/100$
$P(Y=F, Z=T) = 10/100$	$P(Y=F) = 50/100$
$P(Y=F, Z=F) = 40/100$	$P(Z=F) = 45/100$

$$\Rightarrow I(Y,Z) = 0.3197 - 0.1084 - 0.1459 + 0.3320$$

$$\boxed{I(Y,Z) = 0.39731}$$

3. Compute the mutual information *I*(*Z, Y*) based on the frequencies observed in the data. (5 pts)

3) $$I(X,Z) = P(X=T, Z=T)\log\left(\frac{P(X=T, Z=T)}{P(X=T)P(Z=T)}\right) + P(X=T, Z=F)\log\left(\frac{P(X=T, Z=F)}{P(X=T)P(Z=F)}\right) + P(X=F, Z=T)\log\left(\frac{P(X=F, Z=T)}{P(X=F)P(Z=T)}\right) + P(X=F, Z=F)\log\left(\frac{P(X=F, Z=F)}{P(X=F)P(Z=F)}\right)$$

$P(X=T, Z=T) = 38/100$	$P(X=T) = 50/100$
$P(X=T, Z=F) = 12/100$	$P(X=F) = 50/100$
$P(X=F, Z=T) = 17/100$	$P(Z=T) = 55/100$
$P(X=F, Z=F) = 33/100$	$P(Z=F) = 45/100$

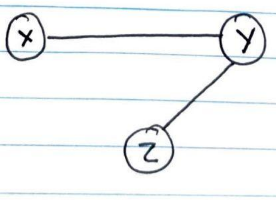
$$\Rightarrow I(X,Z) = 0.17729 - 0.10882 - 0.11796 + 0.18233$$

$$\boxed{I(X,Z) = 0.13284}$$

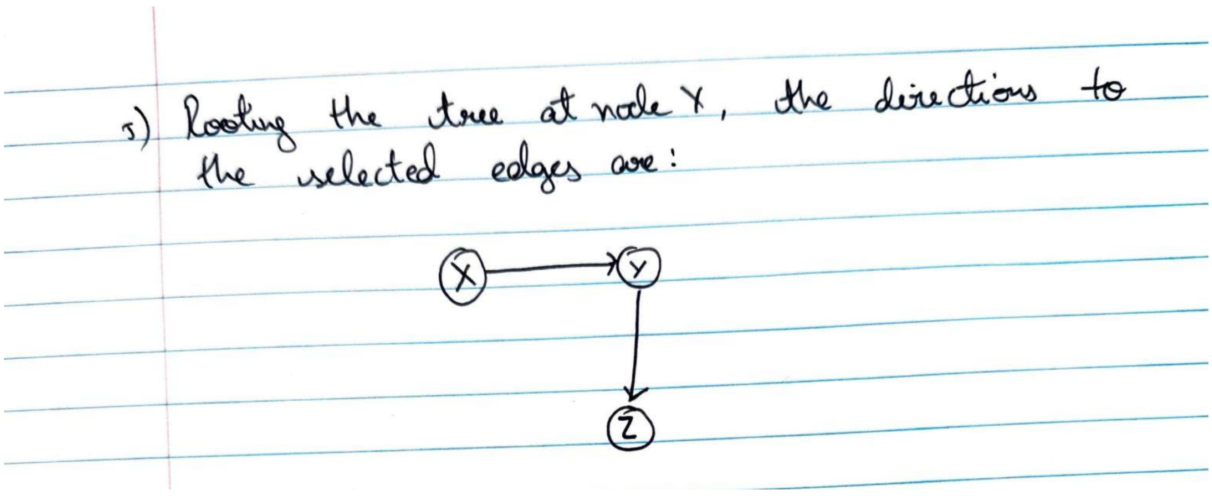
4. Which undirected edges will be selected by the Chow-Liu algorithm as the maximum spanning tree? (5 pts)

4) Info gains : $\{ I(X,Y) = 0.27807 ; I(Y,Z) = 0.39731 ; I(X,Z) = 0.13284 \}$

Constructing a maximum spanning tree in accordance with Chow-Liu algorithm, we will select the top 2 edges with the maximum information gain.



5. Root your tree at node X, and assign directions to the selected edges. (10 pts)


5) Rooting the tree at node X, the directions to the selected edges are:

$$\begin{tikzcd} X \arrow[r] & Y \arrow[d] \\ & Z \end{tikzcd}$$

3 Game of Classifiers [60pts]

3.1 Implementation

Implement the following models in the choice of your programming language. Include slack variables in SVM implementation if needed. You can use autograd features of PyTorch, TensorFlow, etc., or derive gradients on your own (but do not use inbuilt models for SVM, Kernel SVM, and Logistic Regression from libraries).

- Implement Linear SVM (without kernels).
- Implement Kernel SVM, with options for linear, rbf, and polynomial kernels. You should keep the kernel parameters tunable (e.g., do not fix the degree of polynomial kernels but keep it as a variable and play with different values of it.) Is Linear SVM a special case of Kernel SVMs?
- Implement Logistic Regression with and without kernels (use same kernels as above).

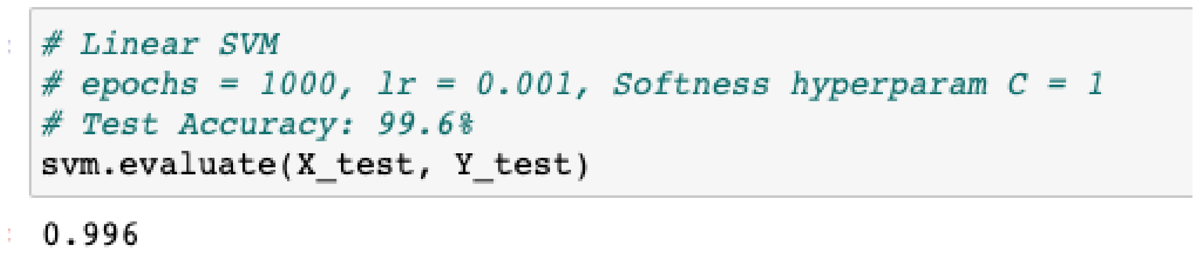
3.2 Synthetic Dataset-1 (20 pts)

Generate a 2-D dataset as follows: Let μ = 2.5 and I_2 be the 2 × 2 identity matrix. Generate points for the positive and negative classes, respectively from $N([\mu, 0], I_2)$, and $N([-\mu, 0], I_2)$. For each class, generate 750 points (1500 in total). Randomly create train, validation, and test splits of 1000, 250, and 250 points, respectively. Do the following with this dataset:

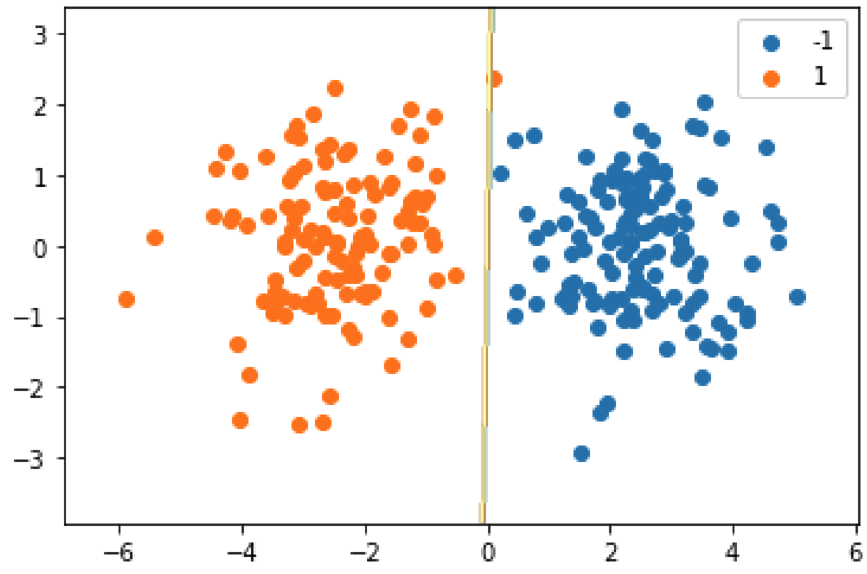
1. (5 pts) Train your Linear SVM, Logistic Regression models and report decision boundaries and test accuracies.

Sol:

Linear SVM Accuracy = 99.6%


```
# Linear SVM
# epochs = 1000, lr = 0.001, Softness hyperparam C = 1
# Test Accuracy: 99.6%
svm.evaluate(X_test, Y_test)
```
```
0.996
```

Linear SVM Decision Boundary:



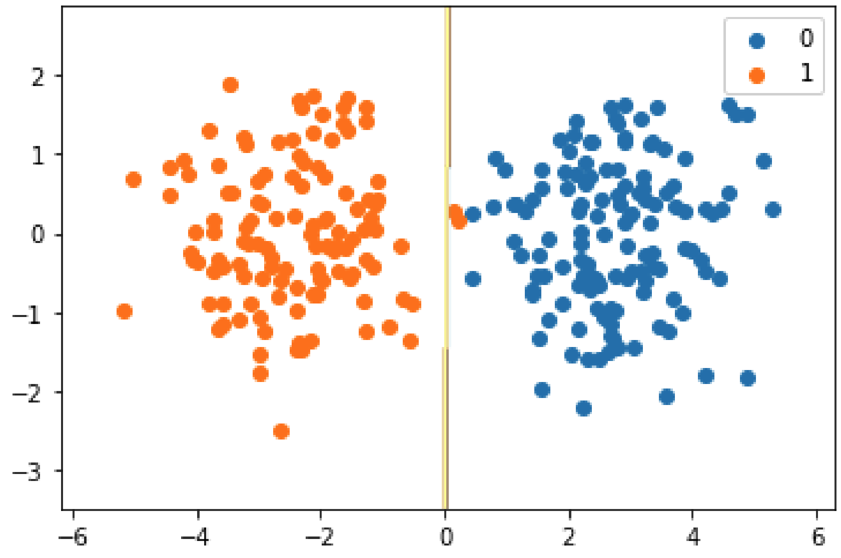
Logistic Regression Accuracy = 99.2%

```
# Logistic Regression
# alpha (learning rate = 0.1), epochs = 1000
# Test Accuracy = 99.2%
lr.evaluate(X_test, Y_test)
```

```
0.992
```

Logistic Regression Decision Boundary:

```
decision_boundary(lr, X_test, Y_test)
```



2. (5 pts) Show the decision boundaries with *k*-NN and Naive Bayes Classifiers. (You can use library implementations or implement from scratch. Figure out the hyper-parameters using the validation set.)

Sol:

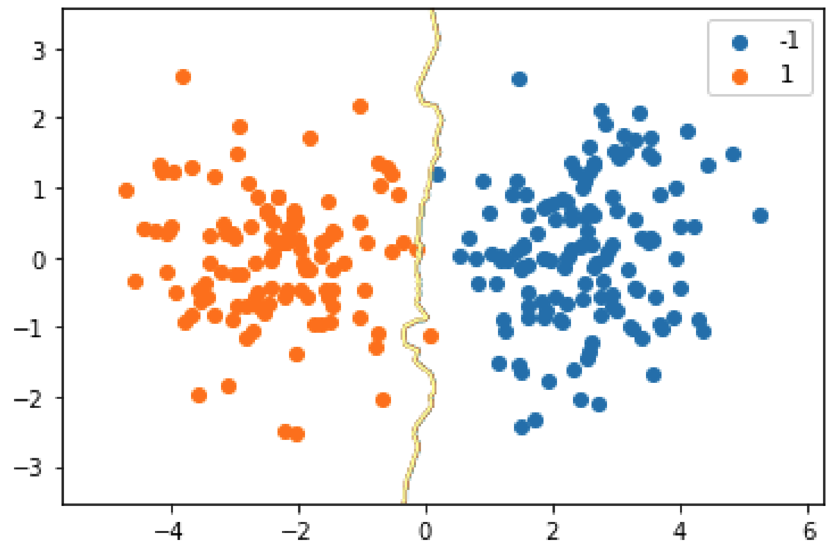
k-NN Accuracy = 99.6%

```
# K Nearest Neighbors
# Test accuracies = 99.6%
KNN = KNeighborsClassifier()
KNN.fit(X_train, Y_train)
KNN.score(X_test, Y_test)
```

```
0.996
```

k-NN Decision Boundary:

```
decision_boundary(KNN, X_test, Y_test)
```



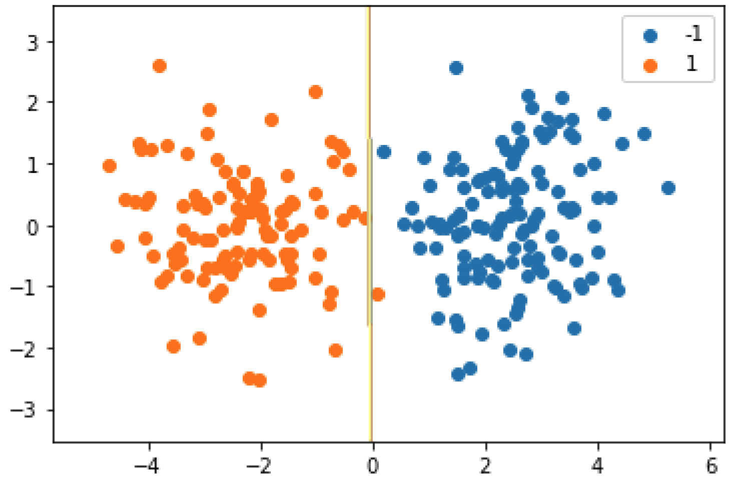
Naive-Bayes Classifier Accuracy = 99.6%

```
# Gaussian Naive Bayes
# Test accuracies = 99.6%
GNB = GaussianNB()
GNB.fit(X_train, Y_train)
GNB.score(X_test, Y_test)
```

```
0.996
```

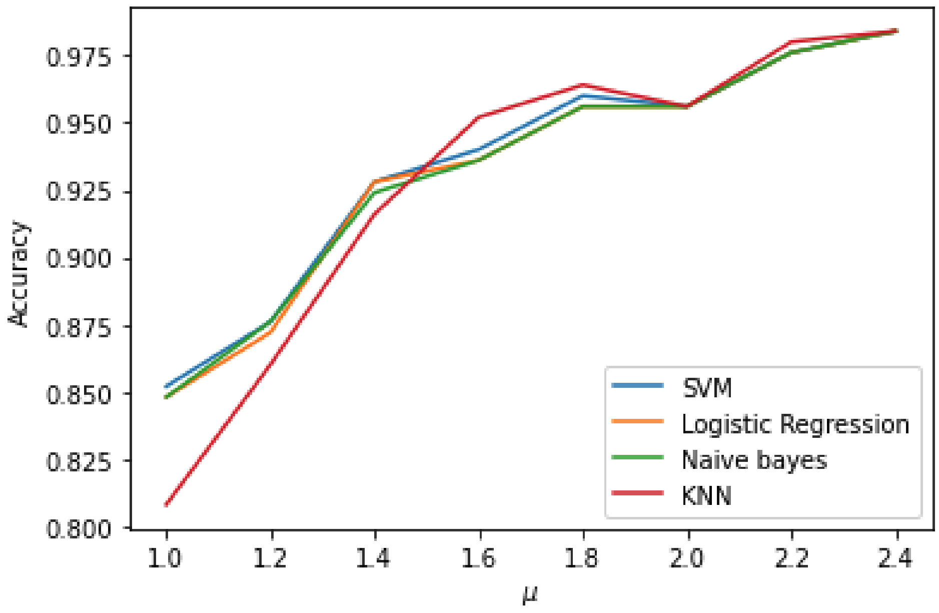
Naive-Bayes Classifier Decision Boundary:

```
decision_boundary(GNB, X_test, Y_test)
```



3. (5 pts) Repeat the process by varying μ from 1.0 to 2.4 with a step size of 0.2 for each value of μ to obtain test accuracies of the models and plot (μ on *x*-axis and test accuracy on *y*-axis). (You will have a curve for each of the 4-classifiers mentioned above.)

Sol:



4. (5 pts) What are your conclusions from this exercise?

Sol:

- Accuracy is increasing with mean because the centers are moving apart from each other and hence it's easier to linearly separate them
- K-NN is performing poorly in the beginning because the data points from opposing class labels are closer to each other with smaller mean
- The accuracies start to merge as mean increases because the two clusters are easier to linearly separate for all models

3.3 Synthetic Dataset-2 (20 pts)

Generate 1500 data points from the 2-D circles dataset of sklearn:

sklearn.datasets.make_circles

Randomly create train, validation, and test splits of 1000, 250, and 250 points, respectively. Evaluate the above classifiers in this setting.

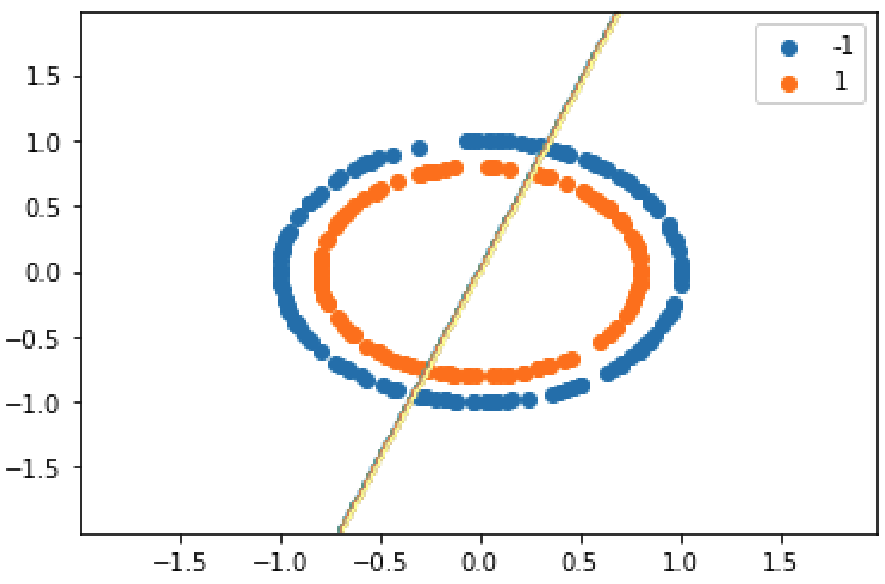
1. (5 pts) Show decision boundaries for Linear SVM and Logistic Regression classifiers.

Sol:

Linear SVM Decision Boundary:

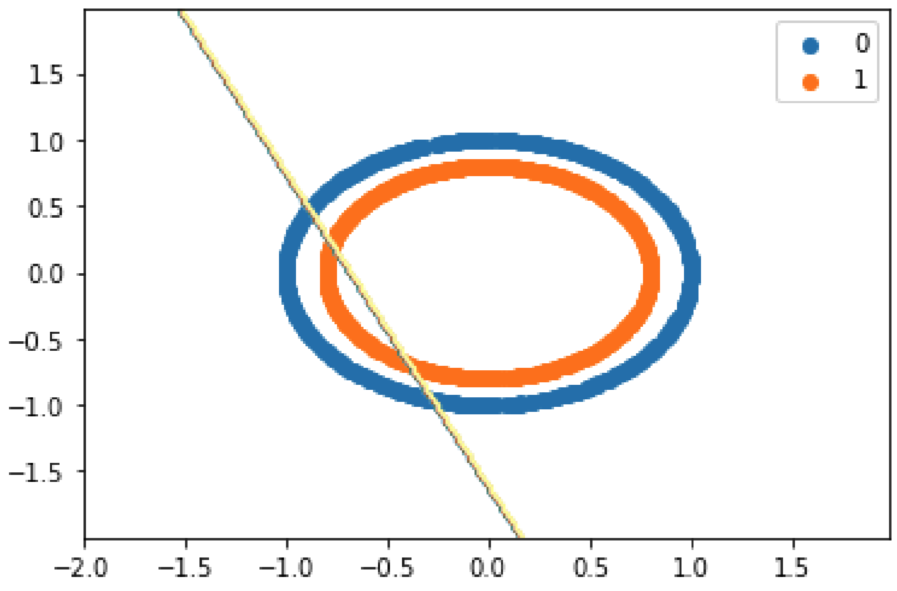
```
decision_boundary(svm, X_test, Y_test)
```

```
[-1 -1 -1 ...  1  1  1]
```



Logistic Regression Decision Boundary:

```
decision_boundary(lr, X_train, Y_train)
```



2. (5 pts) Show decision boundaries for Kernel SVM and Kernel Logistic Regression (use rbf, polynomial kernels). Try different values of hyperparameters, and report results with whichever works best.

Sol:

Kernel Logistic Regression:

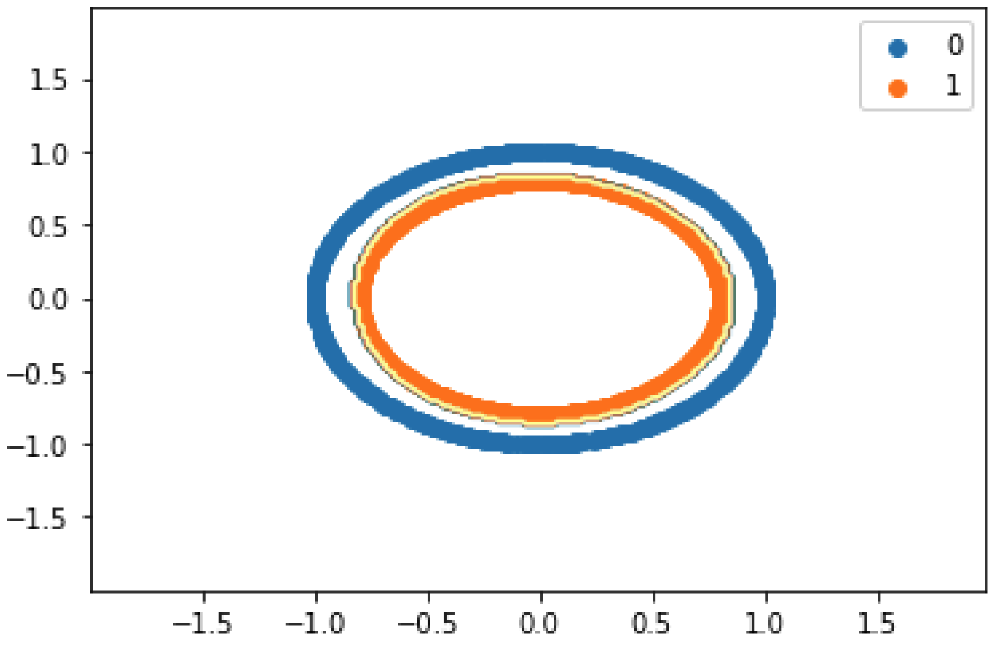
Polynomial kernel accuracy = 100%

```
# Kernel Logistic Regression
# Kernel = "Polynomial", degree=3, lr = 0.01, epochs = 2000
# Accuracy = 100%
klr.evaluate(X_test, Y_test)
```

```
1.0
```

Polynomial kernel decision Boundary:

```
decision_boundary(klr, X_train, Y_train)
```



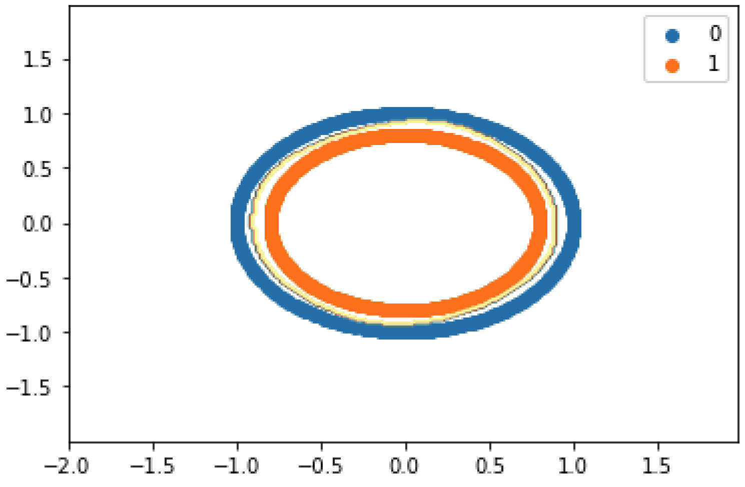
RBF kernel accuracy = 100%

```
# Kernel Logistic Regression
# Kernel = "RBF", gamma = 5, lr = 0.1, epochs = 100
# Accuracy = 100%
klr.evaluate(X_test, Y_test)
```

```
1.0
```

RBF kernel decision boundary:

```
decision_boundary(klr, X_train, Y_train)
```



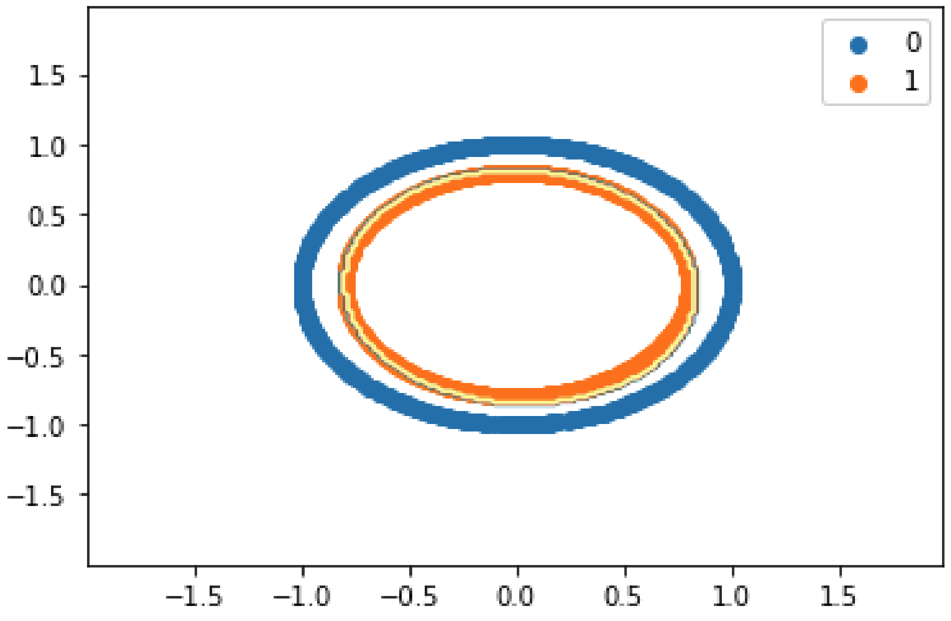
Kernel SVM:

Polynomial Kernel Accuracy = 100%

```
# Kernel SVM
# Kernel = "Polynomial", degree=2, lr = 0.01, epochs = 1000
# Accuracy = 100%
svm.evaluate(X_test, Y_test)
```

```
1.0
```

Polynomial Kernel Decision Boundary:



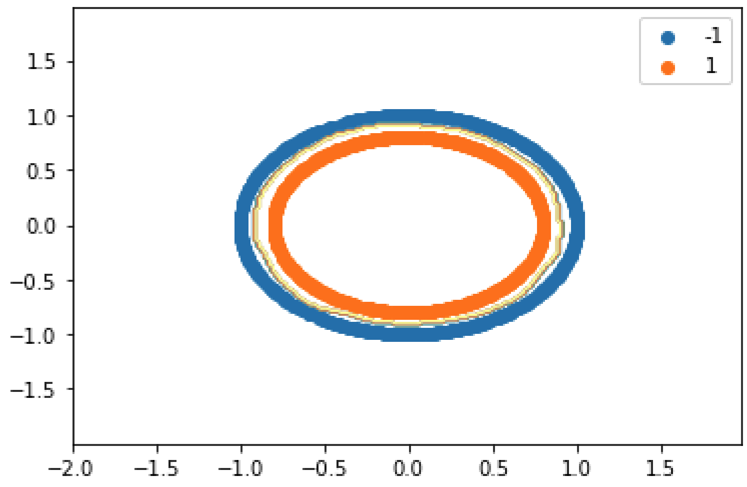
RBF Kernel Accuracy = 100%

```
# RBF kernel in SVM
# C=1, gamma = 100, epochs=1000, lr=0.001
# Accuracy = 100%
svm.evaluate(X_test, Y_test)
```

```
1.0
```

RBF Kernel Decision Boundary:

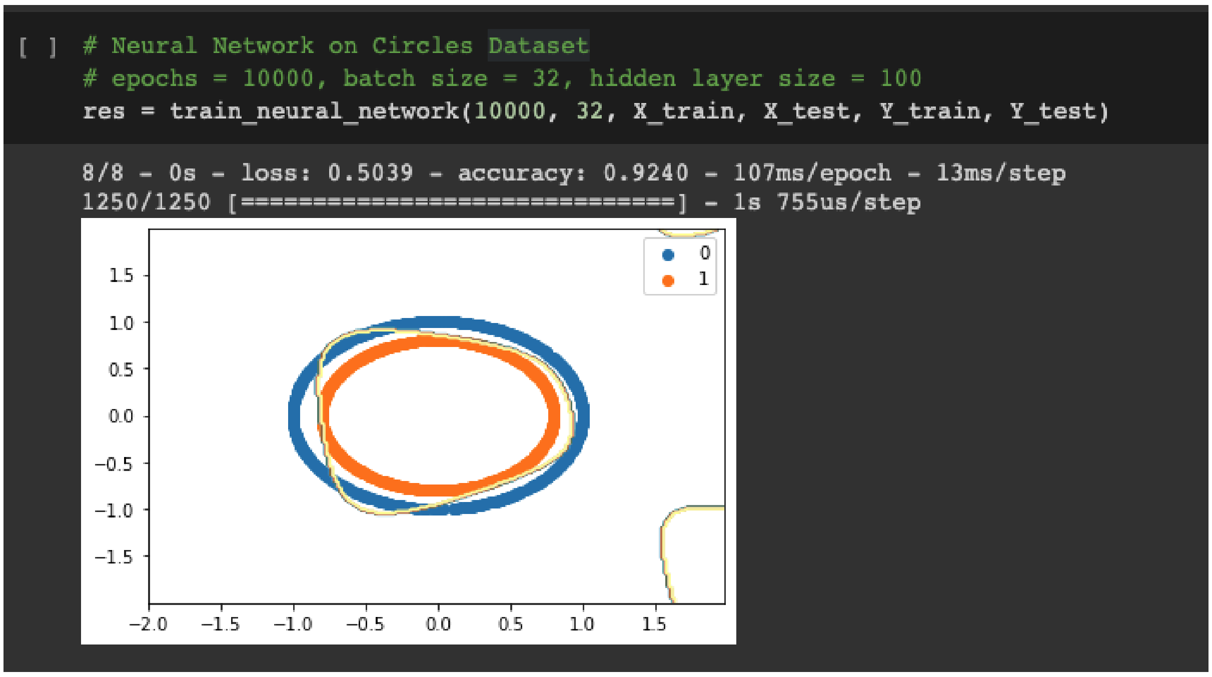
```
decision_boundary(svm, X_train, Y_train)
```



3. (5 pts) Train Neural Network from HW4, and *k*-NN classifiers on this dataset and show decision bound aries. (You can use library implementation for these classifiers).

Sol:

Neural Network Decision Boundary:

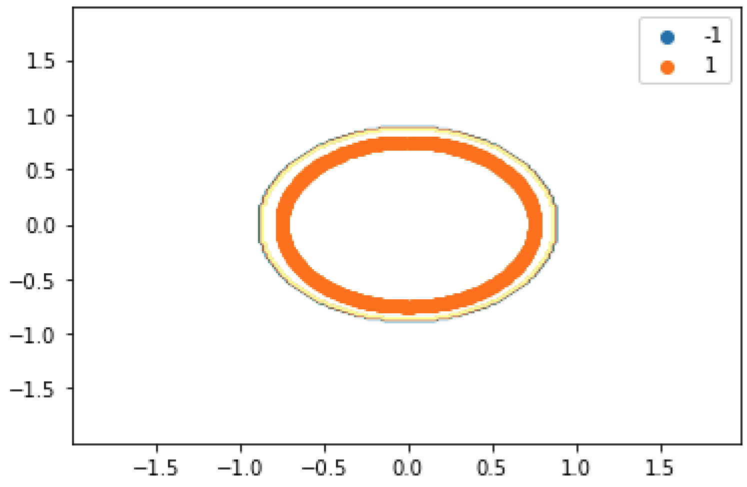


```
[ ] # Neural Network on Circles Dataset
    # epochs = 10000, batch size = 32, hidden layer size = 100
    res = train_neural_network(10000, 32, X_train, X_test, Y_train, Y_test)

    8/8 - 0s - loss: 0.5039 - accuracy: 0.9240 - 107ms/epoch - 13ms/step
    1250/1250 [==============================] - 1s 755us/step
```

KNN Decision Boundary:

```
decision_boundary(KNN, X_train, Y_train)
```



KNN Accuracy = 100%

```
# K Nearest Neighbors on circles dataset
# Test accuracies = 100%
KNN.score(X_test, Y_test)
```

```
1.0
```

4. (5 pts) What are your conclusions from this exercise?

Sol:

- Both kernelized logistic regression and SVM using RBF and polynomial kernels give 100% accuracy due to easy overfitting on this dataset
- Logistic Regression and Linear SVM without kernels perform poorly because the data is not linearly separable
- By adding the kernel trick and hence transforming the input space to higher dimensions, we are able to achieve much higher accuracies
- The RBF and polynomial kernels very closely approximate to KNN behavior with increasing dimensions

3.4 Evaluation on Real Dataset (20 pts)

Let's put all this to some real use. For this problem, use the Wisconsin Breast Cancer dataset. You can download it from the sklearn library:

sklearn.datasets.load_breast_cancer

1. (10 pts) Do all the points of Section 3.3 in this dataset. Since these are high-dimensional data, you do not have to show the decision boundaries. Report test accuracies for these classifiers and discuss your findings.

Sol:

Linear SVM, Acc = 40%

```
# Linear SVM on Breat Cancer Dataset
# Accuracy = 40%, epochs = 10000, lr=0.01, C=2
svm.evaluate(X_test, Y_test)
```

```
0.4
```

Logistic Regression, Acc = 93.6%

```
# Logistic Regression - Linear
# epochs=1000, alpha (lr) = 0.01
# Accuracy = 93.68%
lr.evaluate(X_test, Y_test)
```

```
/var/folders/v0/g401c4x979d___jdl5ddw
in exp
  return 1 / (1 + np.exp(-X))
```

```
0.9368421052631579
```

Kernel SVM using Polynomial, Acc = 40%

```
# Polynomial kernel SVM on breast cancer dataset
# Softness parameter C=1, degree = 2, lr = 0.1, epochs = 1000
# Accuracy = 40%
svm.evaluate(X_test, Y_test)
```

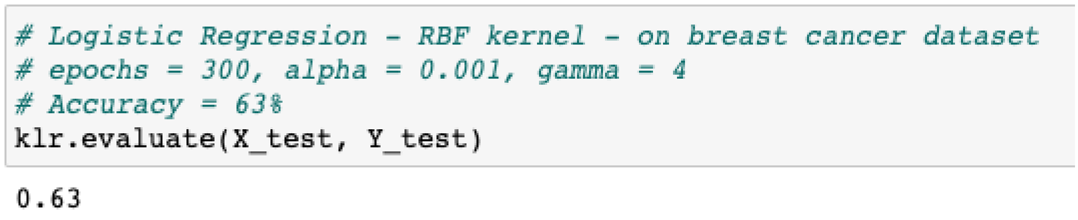
```
0.4
```

Kernel SVM using RBF, Acc = 94.7%

```
# RBF kernel SVM
# Softness parameter C = 1, gamma = 1, epochs = 1000, lr = 0.001
svm.evaluate(X_test, Y_test)
```

```
0.9473684210526315
```

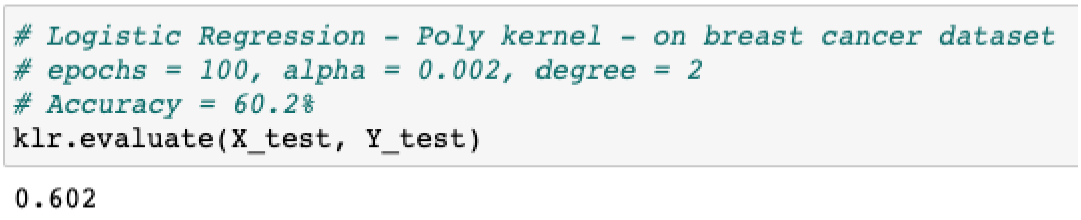
Logistic Regression using RBF kernel: Acc=63%



```
# Logistic Regression - RBF kernel - on breast cancer dataset
# epochs = 300, alpha = 0.001, gamma = 4
# Accuracy = 63%
klr.evaluate(X_test, Y_test)
```

```
0.63
```

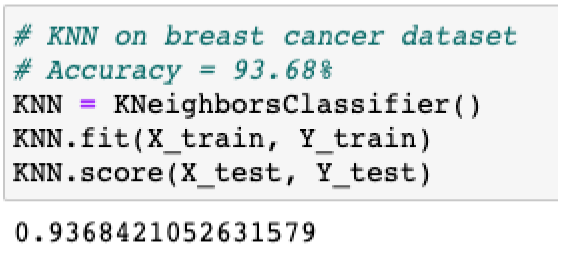
Logistic Regression using Polynomial kernel: Acc=60.2%



```
# Logistic Regression - Poly kernel - on breast cancer dataset
# epochs = 100, alpha = 0.002, degree = 2
# Accuracy = 60.2%
klr.evaluate(X_test, Y_test)
```

```
0.602
```

KNN Accuracy: Acc = 93.68%



```
# KNN on breast cancer dataset
# Accuracy = 93.68%
KNN = KNeighborsClassifier()
KNN.fit(X_train, Y_train)
KNN.score(X_test, Y_test)
```

```
0.9368421052631579
```

Neural Network: Acc = 90.34%

2. (10 pts) In addition, you also want to figure out the important features which determine the class. Which regularization will you use for this? Upgrade your SVM, Kernel SVM implementation to include this regularization. Discuss the important features that you obtain by running your regularized SVM on this dataset. (You might need to normalize this dataset before training any classifier.)

Sol:

- To induce sparsity, I tried adding L1 regularization to the objective function and upgrading the gradients accordingly. This will try to reduce weights for correlated features to 0. (The code changes are in Kernel SVM - Regularized and Linear SVM - Regularized)
- Another approach was to replace the 1/2 W**2 term by W term in the loss/objective function, effectively changing L2 with L1 norm.
- The most important features found for this dataset are as follows: (after maxmin regularization)
    - radius _worst
    - concave.points_mean
    - area_worst
    - area_mean
    - concave.points_worst
    - perimeter_mean
    - area_se
    - concavity_worst