Name:- Aniket Kumar

SR.NO:- 30

RollNo:- PF45

PRN No:- 1032171203.

AML LAB ASSIGNMENT - 4.

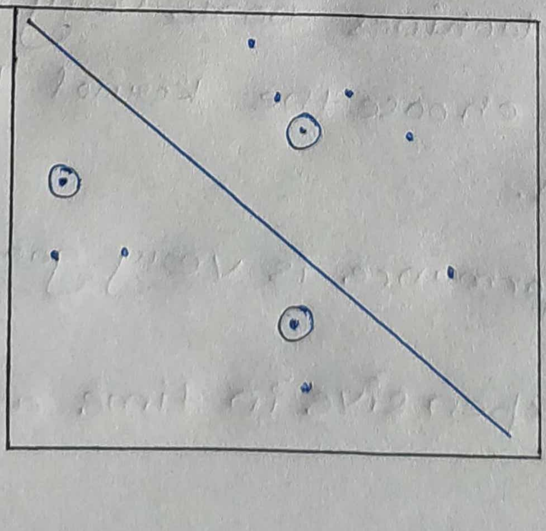
Problem Statement:- To study / implement SVM

Algorithm:- SVM.

SVM:-

It is highly preferred by many as it produces significant accuracy with less computational power. It can be used for both regression and classification

- The line that maximizes the minimum margin is a good best.
- The maximum margin separator is determined by a subset of datapoints.



- Data points in this subset are called support vectors

- It will be useful computionally if only a small fraction of the datapoints are support vectors.

- The support vectors are used to decide which side of the separator a test case is on.

Hyper plane :-

Hyper plane are decision boundaries that help classify the data points. Data points failing on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyper -plane depends upon the number of features.

Main Ideas behind the kernel function.

1. Starts with data in low dimension
2. Moves the data into a higher dimension if it is not linearly separates the hyperplane

- Support Vector Machines work very well in practice.

  The user must choose the kernel function and its parameters.

  The test performance is very good.

- They can be expensive in time and space for big dataset.

- The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space.

Conclusion

SVM classifier was studied and the implementation was performed for the kernel functions.

NAME : Aniket Kumar

S.no : 30

PRN : 1032171203

ROLL : PF45

Subject : AML Lab Assignment 4

Problem Statement : Implement SVM

Import the required libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

import warnings
```

```
warnings.filterwarnings("ignore")
SEED = 42
```

Import the data

In [2]:
```
column_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
'pedigree', 'age', 'label']
df = pd.read_csv(r"C:\Users\Aniket\Desktop\Aml\3\pima-indians-diabetes.
csv", header = None, names = column_names)
```

Display the info about dataset

In [3]:
```
df.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
df.head()
```

Out[4]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnant    768 non-null int64
glucose     768 non-null int64
```

```
bp          768 non-null int64
skin        768 non-null int64
insulin     768 non-null int64
bmi         768 non-null float64
pedigree    768 non-null float64
age         768 non-null int64
label       768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
df.describe()
```

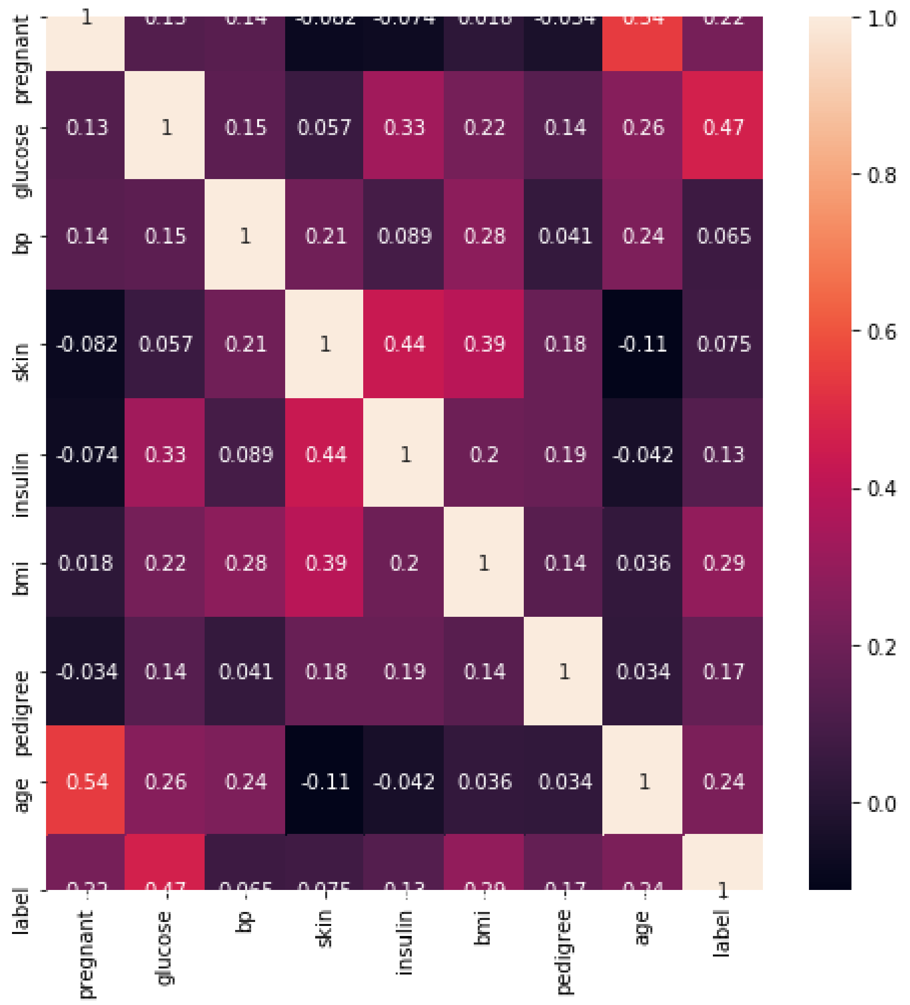
Out[6]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.2
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.7
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.0

Correlation Matrix

In [7]:
```
fig = plt.figure(figsize = (8, 8))
correlationmatrix = df.corr()
sns.heatmap(correlationmatrix, annot = True)
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15b39bf59c8>
```



Data Preprocessing

```
In [8]: df[['glucose', 'bp', 'skin', 'insulin', 'bmi']] = df[['glucose', 'bp',
        'skin', 'insulin', 'bmi']].replace(0, np.NaN)
        df.isnull().sum()
```

```
Out[8]: pregnant      0
```

```
glucose       5
bp           35
skin        227
insulin     374
bmi          11
pedigree      0
age           0
label         0
dtype: int64
```

In [9]:
```
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
df[['glucose', 'bp']] = imputer.fit_transform(df[['glucose', 'bp']])
imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')
df[['skin', 'insulin', 'bmi']] = imputer.fit_transform(df[['skin', 'ins
ulin', 'bmi']])
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
pregnant    0
glucose     0
bp          0
skin        0
insulin     0
bmi         0
pedigree    0
age         0
label       0
dtype: int64
```

Dependant and Independant variables

In [11]:
```
# Independant variables
feature_columns = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bm
i', 'pedigree', 'age']
X = df[feature_columns]
print("FEATURES: ", X.shape)
```

```
# Dependant variables
y = df.label
print("LABEL: ", y.shape)
```

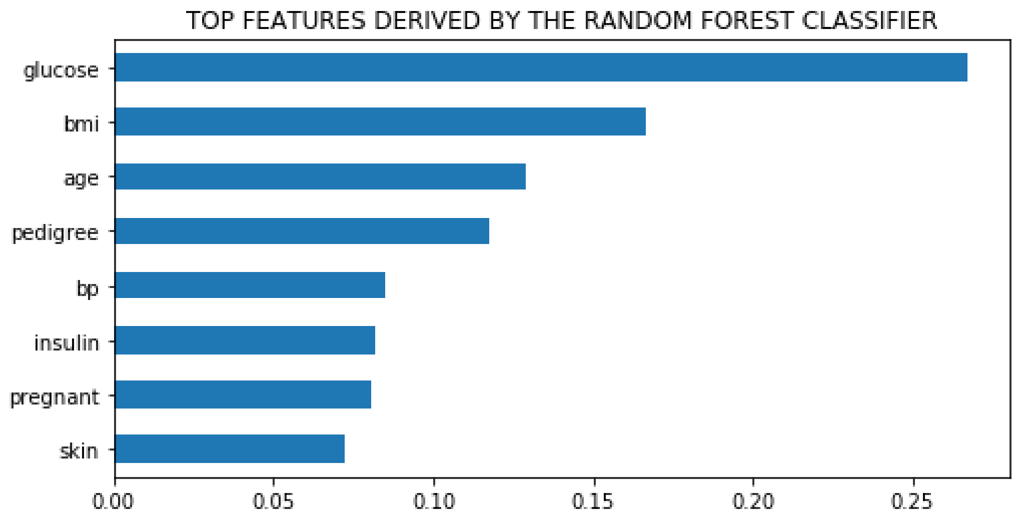
```
FEATURES:  (768, 8)
LABEL:  (768,)
```

Feature Selection

In [12]:
```
rfc = RandomForestClassifier(random_state = SEED, n_estimators = 100)
model = rfc.fit(X, y)

(pd.Series(model.feature_importances_, index = X.columns)
   .nlargest(10)
   .plot(kind = 'barh', figsize = [8, 4])
   .invert_yaxis())
plt.title('TOP FEATURES DERIVED BY THE RANDOM FOREST CLASSIFIER')
```

Out[12]:
```
Text(0.5, 1.0, 'TOP FEATURES DERIVED BY THE RANDOM FOREST CLASSIFIER')
```



In [13]:
```
feature_columns = ['glucose', 'bmi']
```

```
X = X[feature_columns]
X.head()
```

Out[13]:

	glucose	bmi
0	148.0	33.6
1	85.0	26.6
2	183.0	23.3
3	89.0	28.1
4	137.0	43.1

Feature Scaling

In [14]:
```
scaler = StandardScaler()
standardized_X = pd.DataFrame(scaler.fit_transform(X), columns = featur
e_columns)
X = standardized_X
X.head()
```

Out[14]:

	glucose	bmi
0	0.865108	0.166619
1	-1.206162	-0.852200
2	2.015813	-1.332500
3	-1.074652	-0.633881
4	0.503458	1.549303

In [15]:
```
X.describe()
```

Out[15]:

	glucose	bmi

	glucose	bmi
count	7.680000e+02	7.680000e+02
mean	-3.301757e-16	2.815312e-16
std	1.000652e+00	1.000652e+00
min	-2.554131e+00	-2.074783e+00
25%	-7.212214e-01	-7.212087e-01
50%	-1.540881e-01	-2.258989e-02
75%	6.103090e-01	6.032562e-01
max	2.541850e+00	5.042397e+00

Split the data into Train and Test sets

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 
0.2, random_state = SEED, stratify = y)
print('The shape of the of the resultant data sets are as follows...')
print('X_train: ', X_train.shape)
print('y_train: ', y_train.shape)
print('X_test : ', X_test.shape)
print('y_test : ', y_test.shape)
```

```
The shape of the of the resultant data sets are as follows...
X_train:  (614, 2)
y_train:  (614,)
X_test :  (154, 2)
y_test :  (154,)
```

Linear SVM

In [17]:
```
svc_linear = SVC(kernel = 'linear', random_state = SEED)
model = svc_linear.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

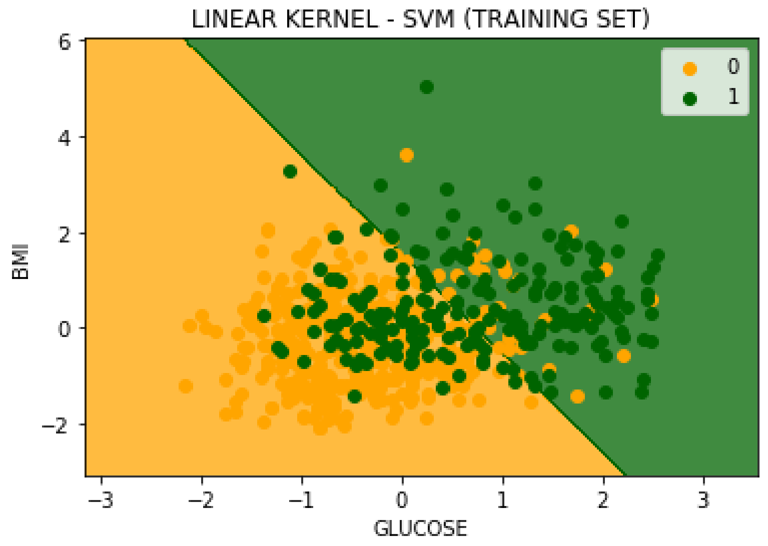
```
cm = confusion_matrix(y_test, y_pred)
print('CONFUSION MATRIX: \n', cm)
```

```
CONFUSION MATRIX: 
 [[85 15]
 [29 25]]
```

Vizualising Training dataset

In [18]:
```
X_set = X_train.to_numpy()
y_set = y_train.to_numpy()
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_
set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_
set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('orange', 'darkgreen'
)))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('orange', 'darkgreen'))(i), label =
 j)
plt.title('LINEAR KERNEL - SVM (TRAINING SET)')
plt.xlabel('GLUCOSE')
plt.ylabel('BMI')
plt.legend()
plt.show()
```

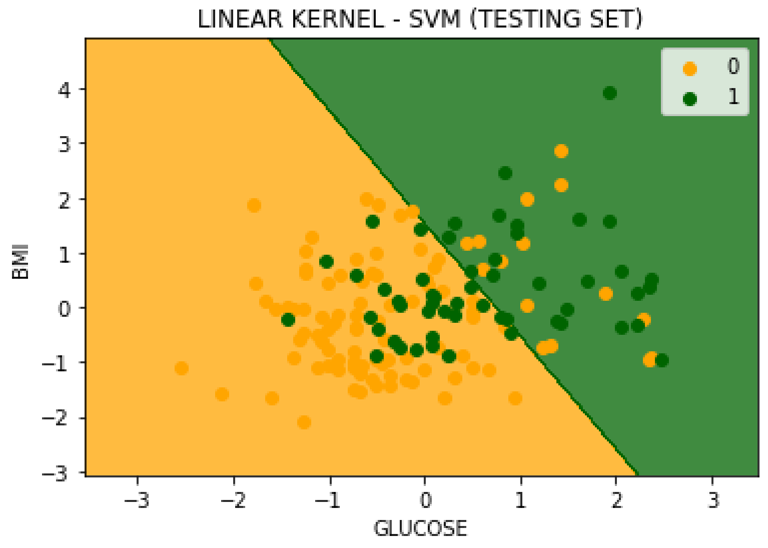
```
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
you really want to specify the same RGB or RGBA value for all points.
```



Vizualising Test dataset

In [19]:
```
X_set = X_test.to_numpy()
y_set = y_test.to_numpy()
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_
set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_
set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('orange', 'darkgreen'
)))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('orange', 'darkgreen'))(i), label =
 j)
plt.title('LINEAR KERNEL - SVM (TESTING SET)')
plt.xlabel('GLUCOSE')
plt.ylabel('BMI')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
```



RBF SVM

```
In [20]: svc_rbf = SVC(kernel = 'rbf', random_state = SEED)
         model = svc_rbf.fit(X_train, y_train)
         y_pred = model.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred)
print('CONFUSION MATRIX: \n', cm)
```

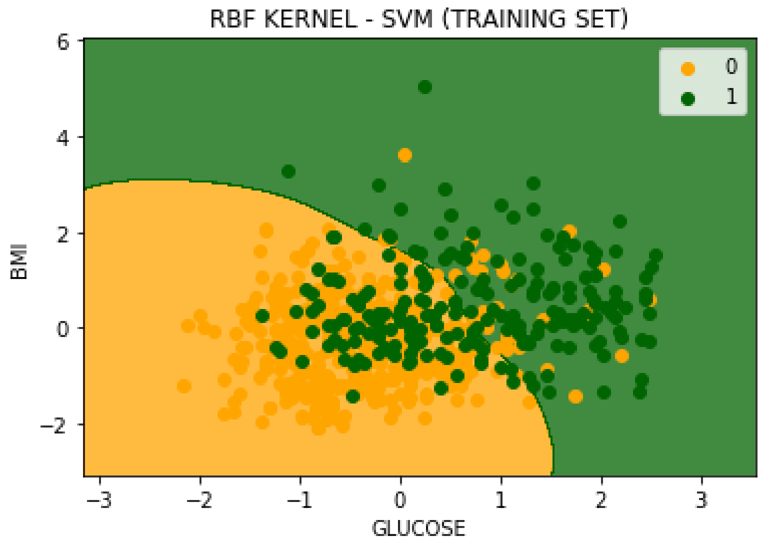
```
CONFUSION MATRIX: 
 [[84 16]
 [31 23]]
```

Vizualising Training dataset

In [21]:

```
X_set = X_train.to_numpy()
y_set = y_train.to_numpy()
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_
set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_
set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('orange', 'darkgreen'
)))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('orange', 'darkgreen'))(i), label =
 j)
plt.title('RBF KERNEL - SVM (TRAINING SET)')
plt.xlabel('GLUCOSE')
plt.ylabel('BMI')
plt.legend()
plt.show()
```

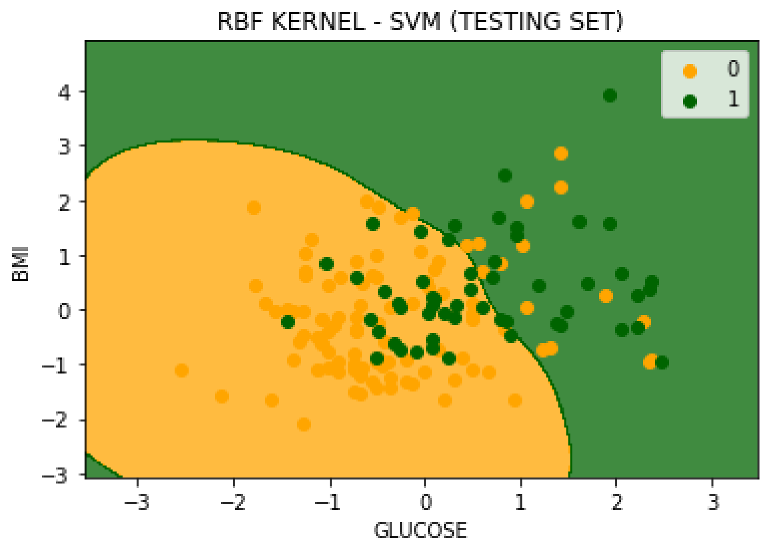
```
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
you really want to specify the same RGB or RGBA value for all points.
```



Vizualising Test dataset

In [22]:
```
X_set = X_test.to_numpy()
y_set = y_test.to_numpy()
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_
set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_
set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(), X2.ravel()]).T
).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('orange', 'darkgreen'
)))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('orange', 'darkgreen'))(i), label =
 j)
plt.title('RBF KERNEL - SVM (TESTING SET)')
plt.xlabel('GLUCOSE')
plt.ylabel('BMI')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
```


RBF KERNEL - SVM (TESTING SET)
BMI
GLUCOSE