Aadith Lasar

LY CORE-2

2203262

MACHINE LEARNING LAB-8

Title: Implement SVM Classifier or Regression for given dataset After completion of this experiment students will be able to:

1.To learn SVM and kernel functions

2.To implement SVM classifier

Aim: Implement SVM Classifier or Regression for given dataset

Theory:

SVM: support-vector machines are supervised learning models with associated learning algorithms that analyzedata used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mappingtheir inputs into high-dimensional feature spaces.

Kernel function

In machine learning, kernel methods are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principalcomponents, correlations, classifications) in datasets. For many algorithms thatsolve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified feature map: in contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation.

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply

computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicitcomputation of the

coordinates. This approach is called the "kernel trick".[1] Kernel functions have been introduced for sequence data, graphs, text,images, as well as vectors.

Kernel trick

The kernel trick seems to be one of the most confusing concepts in statistics and machine learning; it first appears to be genuine mathematical sorcery, not to mention the problem of lexical ambiguity (does kernel refer to: a non-parametricway to estimate a probability density (statistics), the set of vectors v for which alinear transformation T maps to the zero vector — i.e. T(v) = 0 (linear algebra), the set of elements in a group G that are mapped to the identity element by a homomorphism between groups (group theory), the core of a computer operating system (computer science), or something to do with the seeds of nutsor fruit?).

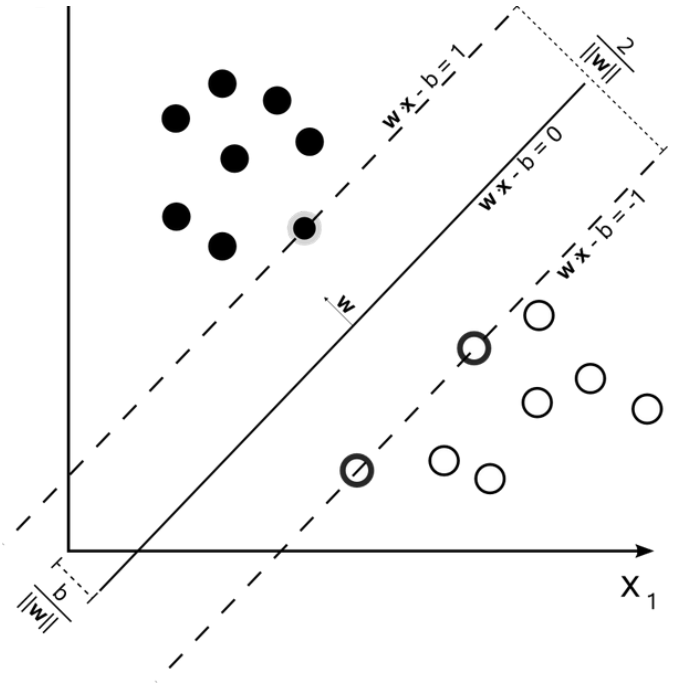
The kernel trick also illustrates some fundamental ideas about different ways torepresent data and how machine learning algorithms “see” these different datarepresentations. And finally, the seeming mathematical sleight of hand in the kernel trick just begs one to further explore what it actually means.

Significance of SVM

it capable of doing both classification and regression. In this post I'll focus on using SVM for classification. In particular I'll be focusing on non-linear SVM, or SVM using a non-linear kernel. Non-linear SVM means that the boundary that the algorithm calculates doesn't have to be a straight line. The benefit is that you can capture much more complex relationships between your datapoints without having to perform difficult transformations on your own. The downside is that the training time is much longer as it's much more computationally intensive.

Originally Answered: what is the purpose of support vector in SVM?

A support vector machine attempts to find the line that "best" separates two classes of points. By "best", we mean the line that results in the largest margin between the two classes. The points that lie on this margin are the support vectors.



Here we have three support vectors.

The nice thing about acknowledging these support vectors is that we can then formulate the problem of finding the "maximum-margin hyperplane" (the line that best separates the two classes) as an optimization problem that only considers these support vectors. So we can effectively throw out the vast majority of our data, which makes the classification process go much faster than,say, a neural network.

More importantly, by presenting the problem in terms of the support vectors (the so-called *dual form*), we can apply what's called the *kernel trick* to effectively transform the SVM into a non-linear classifier.

Code:-

```
# import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore') from sklearn import svm
from sklearn.svm import SVC

from sklearn.model_selection
import GridSearchCV


from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.model_selection import train_test_split, cross_val_score
data = pd.read_csv('forestfires.csv')
data.head()
data.shape
data.columns
data1 = data.drop('month', axis = 1)
data2 = data1.drop('day', axis = 1)
data2.head()
data2.shape

from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
#we have used labelencoder for the column 28, i.e size_category data2.iloc[:, 28] =
labelencoder.fit_transform(data2.iloc[:,28]) data2.head()
X = data2.iloc[:,0:28] Y = data2.iloc[:,28]
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3)

X_train.shape, y_train.shape, X_test.shape, y_test.shape clf = SVC()
clf.fit(X_train, y_train) y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100 print("Accuracy =", acc) c
onfusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))


clf = SVC()
```

param_grid = [{'kernel':['rbf', 'poly'], 'gamma':[100, 5, 15, 0.5], 'C':[12, 11, 10, 0.1, 0.001] }] gsv = GridSearchCV(clf, param_grid, cv=10)
gsv.fit(X_train, y_train) gsv.best_params_ , gsv.best_score_
clf = SVC(C= 12, gamma = 100,kernel='poly') clf.fit(X_train , y_train)
y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred) * 100 print("Accuracy =", acc)
confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))

Dataset:-

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthfeb	monthjan	monthjul	monthjun	monthmar	monthmay	monthnov	monthoct	mor
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	...	0	0	0	0	1	0	0	0	
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	...	0	0	0	0	0	0	0	1	
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	...	0	0	0	0	0	0	0	1	
3	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	...	0	0	0	0	1	0	0	0	
4	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	...	0	0	0	0	1	0	0	0	

5 rows × 31 columns

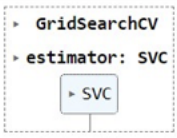
In [16]:
```
clf = SVC()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
print("Accuracy =", acc)
confusion_matrix(y_test, y_pred)
```

```
Accuracy = 76.28205128205127
```

Out[16]:
```
array([[  1,  37],
       [  0, 118]], dtype=int64)
```

In [18]:
```
clf = SVC()
param_grid = [{'kernel':['rbf', 'poly'], 'gamma':[100, 5, 15, 0.5], 'C':[12, 11, 10, 0.1, 0.001] }]
gsv = GridSearchCV(clf, param_grid, cv=10)
gsv.fit(X_train, y_train)
```

Out[18]:



In [19]:
```
gsv.best_params_ , gsv.best_score_
```

Out[19]:
```
({'C': 12, 'gamma': 100, 'kernel': 'poly'}, 0.9805555555555554)
```

In [20]:
```
clf = SVC(C= 12, gamma = 100,kernel='poly')
clf.fit(X_train , y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
print("Accuracy =", acc)
confusion_matrix(y_test, y_pred)
```

```
Accuracy = 98.07692307692307
```

Out[20]:
```
array([[ 36,   2],
       [  1, 117]], dtype=int64)
```

In [21]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.95      0.96        38
           1       0.98      0.99      0.99       118

    accuracy                           0.98       156
   macro avg       0.98      0.97      0.97       156
weighted avg       0.98      0.98      0.98       156
```

Results:-

In [4]:
```
test
```

Out[4]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	native	Salary
0	25	Private	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	34	Private	10th	6	Never-married	Other-service	Not-in-family	White	Male	0	0	30	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15055	33	Private	Bachelors	13	Never-married	Prof-specialty	Own-child	White	Male	0	0	40	United-States	<=50K
15056	39	Private	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	<=50K
15057	38	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	<=50K
15058	44	Private	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	<=50K
15059	35	Self-emp-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	>50K

15060 rows × 14 columns

In [23]:
```
acc = accuracy_score(Ytest, y_pred) * 100
print("Accuracy =", acc)
confusion_matrix(Ytest, y_pred)
print('\n')
print(classification_report(Ytest,y_pred))
```

```
Accuracy = 76.22177954847278
```

```
              precision    recall  f1-score   support

       <=50K       0.77      0.98      0.86     11360
        >50K       0.61      0.09      0.16      3700

    accuracy                           0.76     15060
   macro avg       0.69      0.54      0.51     15060
weighted avg       0.73      0.76      0.69     15060
```

In [5]:
```
train
```

Out[5]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	native	Salary
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
30156	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
30157	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
30158	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
30159	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
30160	52	Self-emp-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

30161 rows × 14 columns

In [26]:
```
acc = accuracy_score(Ytest, y_pred) * 100
print("Accuracy =", acc)
confusion_matrix(Ytest, y_pred)
print('\n')
print(classification_report(Ytest,y_pred))
```

```
Accuracy = 79.64143426294821
```

	precision	recall	f1-score	support
<=50K	0.80	0.97	0.88	11360
>50K	0.73	0.27	0.39	3700
accuracy			0.80	15060
macro avg	0.77	0.62	0.64	15060
weighted avg	0.79	0.80	0.76	15060

In [29]:
```
acc = accuracy_score(Ytest, y_pred) * 100
print("Accuracy =", acc)
confusion_matrix(Ytest, y_pred)
print('\n')
print(classification_report(Ytest,y_pred))
```

```
Accuracy = 76.22177954847278
```

	precision	recall	f1-score	support
<=50K	0.77	0.98	0.86	11360
>50K	0.61	0.09	0.16	3700
accuracy			0.76	15060
macro avg	0.69	0.54	0.51	15060
weighted avg	0.73	0.76	0.69	15060

Conclusion:

Thus we have successfully completed the implementation of SVM.