Support Vector Machines (SVM)

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import pylab as pl
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:
```
# Importing the dataset

df = pd.read_csv("C:/Users/Munazzam/Downloads/nyc-rolling-sales.csv")
df.head()
```

Out[2]:

	Unnamed: 0	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUI CLA PRI
0	4	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	392	6		
1	5	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	26		
2	6	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	39		
3	7	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21		
4	8	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	404	55		

5 rows × 22 columns

Data processing & Exploration

In [3]:
```
#Removing unnecessary columns
del df['EASE-MENT']
del df['Unnamed: 0']
del df['ADDRESS']
del df['APARTMENT NUMBER']
```

In [4]:
```
#Checking for duplicates
sum(df.duplicated(df.columns))
```

Out[4]: 956

In [5]:
```
#Removing duplicate records
df = df.drop_duplicates(df.columns, keep='last')
sum(df.duplicated(df.columns))
```

Out[5]: 0

In [6]:
```
#Convert some of the columns to desired datatype
df['TAX CLASS AT TIME OF SALE'] = df['TAX CLASS AT TIME OF SALE'].astype
('category')
df['TAX CLASS AT PRESENT'] = df['TAX CLASS AT PRESENT'].astype('categor
y')
df['LAND SQUARE FEET'] = pd.to_numeric(df['LAND SQUARE FEET'], errors='c
oerce')
df['GROSS SQUARE FEET']= pd.to_numeric(df['GROSS SQUARE FEET'], errors=
'coerce')
df['SALE PRICE'] = pd.to_numeric(df['SALE PRICE'], errors='coerce')
df['BOROUGH'] = df['BOROUGH'].astype('category')
```

In [7]:

```
# Convert Sale Date to Year
#%Y-%m-%d %H:%M:%S
from datetime import datetime
for i in range(len(df)):
    if True:
        the_date = datetime.strptime(str(df['SALE DATE'][i]), '%Y-%m-%d
 %H:%M:%S')
        df.at[i,'SALE DATE'] = the_date.year
    else:
        df.at[i,'SALE DATE'] = int(df.at[i,'SALE DATE'])

# convert to integer
df['SALE DATE'] = df['SALE DATE'].astype(int)
df['SALE DATE'].head()
```

```
---------------------------------------------------------------------
----
KeyError                                  Traceback (most recent call l
ast)
<ipython-input-7-fcb26584269c> in <module>
      4 for i in range(len(df)):
      5     if True:
----> 6         the_date = datetime.strptime(str(df['SALE DATE'][i]),
'%Y-%m-%d %H:%M:%S')
      7         df.at[i,'SALE DATE'] = the_date.year
      8     else:

~\Anaconda3\lib\site-packages\pandas\core\series.py in __getitem__(sel
f, key)
   1066         key = com.apply_if_callable(key, self)
   1067         try:
-> 1068             result = self.index.get_value(self, key)
   1069
   1070             if not is_scalar(result):

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_value
(self, series, key)
   4728         k = self._convert_scalar_indexer(k, kind="getitem")
   4729         try:
-> 4730             return self._engine.get_value(s, k, tz=getattr(seri
es.dtype, "tz", None))
   4731         except KeyError as e1:
   4732             if len(self) > 0 and (self.holds_integer() or self.
is_boolean()):

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.Int64
HashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.Int64
HashTable.get_item()

KeyError: 48
```

In [8]:
```
df.head()
```

Out[8]:

	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	BUILDING CLASS AT PRESENT	ZIP CODE	RES
0	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	392	6	C2	10009	
1	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	26	C7	10009	
2	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	39	C7	10009	
3	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21	C4	10009	
4	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	404	55	C2	10009	

In [9]:
```
#checking missing values
df.columns[df.isnull().any()]
```

Out[9]:
```
Index(['LAND SQUARE FEET', 'GROSS SQUARE FEET', 'SALE PRICE'], dtype='o
bject')
```

In [10]:
```
miss=df.isnull().sum()/len(df)
miss=miss[miss>0]
miss.sort_values(inplace=True)
miss
```

Out[10]:
```
SALE PRICE            0.168365
LAND SQUARE FEET      0.310484
GROSS SQUARE FEET     0.326371
dtype: float64
```

In [11]:
```
#Convert series to column DataFrame
miss=miss.to_frame()
#Set Column Name
miss.columns=['count']
#Set Index Name
miss.index.names=['Name']
#Create Column from Index
miss['Name']=miss.index
miss
```

Out[11]:

Name	count	Name
SALE PRICE	0.168365	SALE PRICE
LAND SQUARE FEET	0.310484	LAND SQUARE FEET
GROSS SQUARE FEET	0.326371	GROSS SQUARE FEET

In [12]:
```
#Populating mean values for missing data
df['LAND SQUARE FEET']=df['LAND SQUARE FEET'].fillna(df['LAND SQUARE FEE
T'].mean())
df['GROSS SQUARE FEET']=df['GROSS SQUARE FEET'].fillna(df['GROSS SQUARE
 FEET'].mean())
```

In [13]:
```
df.head()
```

Out[13]:

	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	BUILDING CLASS AT PRESENT	ZIP CODE	RES
0	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	392	6	C2	10009	
1	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	26	C7	10009	
2	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	39	C7	10009	
3	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21	C4	10009	
4	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	404	55	C2	10009	

In [14]:
```
# Removing null observations
df = df[(df['SALE PRICE'] > 100000) & (df['SALE PRICE'] < 5000000)]
```

In [15]:
```
df.head()
```

Out[15]:

	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	BUILDING CLASS AT PRESENT	ZIP CODE	RE
3	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21	C4	10009	
6	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	406	32	C4	10009	
13	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	
15	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	
16	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	

In [16]:
```
#  Removing SALE DATE column

del df['SALE DATE']
```

In [17]:
```
df.head()
```

Out[17]:

	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	BUILDING CLASS AT PRESENT	ZIP CODE	RE
3	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21	C4	10009	
6	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	406	32	C4	10009	
13	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	
15	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	
16	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	

Encoding

In [18]:
```
X = df[['BOROUGH','NEIGHBORHOOD','BUILDING CLASS CATEGORY','TAX CLASS AT
PRESENT','BLOCK','LOT','BUILDING CLASS AT PRESENT','ZIP CODE','RESIDENTI
AL UNITS','COMMERCIAL UNITS','TOTAL UNITS','LAND SQUARE FEET','GROSS SQU
ARE FEET','YEAR BUILT','TAX CLASS AT TIME OF SALE','BUILDING CLASS AT TI
ME OF SALE']].values
X[:, 14]
```

Out[18]:
```
array([2, 2, 2, ..., 1, 1, 1], dtype=object)
```

In [19]:
```
# Getting the dependent variables and independent variables
X = df[['BOROUGH','NEIGHBORHOOD','BUILDING CLASS CATEGORY','TAX CLASS AT
PRESENT','BLOCK','LOT','BUILDING CLASS AT PRESENT','ZIP CODE','RESIDENTI
AL UNITS','COMMERCIAL UNITS','TOTAL UNITS','LAND SQUARE FEET','GROSS SQU
ARE FEET','YEAR BUILT','TAX CLASS AT TIME OF SALE','BUILDING CLASS AT TI
ME OF SALE']].values
y = df['SALE PRICE'].values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

labelencoder_X_3 = LabelEncoder()
X[:, 3] = labelencoder_X_3.fit_transform(X[:, 3])

labelencoder_X_6 = LabelEncoder()
X[:, 6] = labelencoder_X_6.fit_transform(X[:, 6])


labelencoder_X_16 = LabelEncoder()
X[:, 15] = labelencoder_X_16.fit_transform(X[:, 15])
```

In [20]:
```
X[0:5]
```

Out[20]:
```
array([[1, 1, 6, 7, 402, 21, 18, 10009, 10, 0, 10, 2272.0, 6794.0, 191
3,
        2, 17],
       [1, 1, 6, 7, 406, 32, 18, 10009, 8, 0, 8, 1750.0, 4226.0, 1920,
2,
        17],
       [1, 1, 8, 5, 373, 40, 20, 10009, 0, 0, 0, 3846.981435858288,
        3874.3228378618364, 1920, 2, 19],
       [1, 1, 8, 5, 373, 40, 20, 10009, 0, 0, 0, 3846.981435858288,
        3874.3228378618364, 1920, 2, 19],
       [1, 1, 8, 5, 373, 40, 20, 10009, 0, 0, 0, 3846.981435858288,
        3874.3228378618364, 1920, 2, 19]], dtype=object)
```

In [21]:
```
y[0:5]
```

Out[21]:
```
array([3936272., 3192840.,  499000.,  529500.,  423000.])
```

In [22]:
```
df.head()
```

Out[22]:

	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	BUILDING CLASS AT PRESENT	ZIP CODE	RE
3	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21	C4	10009	
6	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	406	32	C4	10009	
13	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	
15	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	
16	1	ALPHABET CITY	09 COOPS - WALKUP APARTMENTS	2	373	40	C6	10009	

We then set the target variable, Sale Price

In [23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 54534 entries, 3 to 84545
Data columns (total 17 columns):
BOROUGH                           54534 non-null category
NEIGHBORHOOD                      54534 non-null object
BUILDING CLASS CATEGORY           54534 non-null object
TAX CLASS AT PRESENT              54534 non-null category
BLOCK                             54534 non-null int64
LOT                               54534 non-null int64
BUILDING CLASS AT PRESENT         54534 non-null object
ZIP CODE                          54534 non-null int64
RESIDENTIAL UNITS                 54534 non-null int64
COMMERCIAL UNITS                  54534 non-null int64
TOTAL UNITS                       54534 non-null int64
LAND SQUARE FEET                  54534 non-null float64
GROSS SQUARE FEET                 54534 non-null float64
YEAR BUILT                        54534 non-null int64
TAX CLASS AT TIME OF SALE         54534 non-null category
BUILDING CLASS AT TIME OF SALE    54534 non-null object
SALE PRICE                        54534 non-null float64
dtypes: category(3), float64(3), int64(7), object(4)
memory usage: 6.4+ MB
```

In [24]:
```
# It looks like the SALES PRiCE column includes some values that are flo
ats. We can drop those rows:
cell_df = df
cell_df = cell_df[pd.to_numeric(cell_df['SALE PRICE'], errors='coerce').
notnull()]
cell_df['SALE PRICE'] = cell_df['SALE PRICE'].astype('int')
cell_df.dtypes
```

Out[24]:
```
BOROUGH                           category
NEIGHBORHOOD                        object
BUILDING CLASS CATEGORY             object
TAX CLASS AT PRESENT              category
BLOCK                                int64
LOT                                  int64
BUILDING CLASS AT PRESENT           object
ZIP CODE                             int64
RESIDENTIAL UNITS                    int64
COMMERCIAL UNITS                     int64
TOTAL UNITS                          int64
LAND SQUARE FEET                   float64
GROSS SQUARE FEET                  float64
YEAR BUILT                           int64
TAX CLASS AT TIME OF SALE         category
BUILDING CLASS AT TIME OF SALE      object
SALE PRICE                           int32
dtype: object
```

In [25]:
```
# Target variable, Sale Price

cell_df['SALE PRICE'] = cell_df['SALE PRICE'].astype('int')
y = np.asarray(cell_df['SALE PRICE'])
y [0:10]
```

Out[25]:
```
array([3936272, 3192840,  499000,  529500,  423000,  501000,  450000,
        510000,  350000,  350000])
```

Setting Up SVM

We split our dataset into Train / Test Dataset

In [26]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [27]:
```
# First we make a trial Subset
len(X)
X_b = X[0:500]
len(X_b)
```

Out[27]:
```
500
```

In [28]:
```
# We also make a subset for y

y_b = y[0:500]
len(y_b)
```

Out[28]:
```
500
```

In [29]:
```
# Split the Subset data
X_b_train, X_b_test, y_b_train, y_b_test = train_test_split( X_b, y_b, t
est_size=0.2, random_state=0)
print ('Train set:', X_b_train.shape,  y_b_train.shape)
print ('Test set:', X_b_test.shape,  y_b_test.shape)
```

```
Train set: (400, 16) (400,)
Test set: (100, 16) (100,)
```

In [30]:
```
# modeling the subset data

from sklearn import svm
clf_b = svm.SVC(kernel='rbf')
clf_b.fit(X_b_train, y_b_train)
```

```
C:\Users\Munazzam\Anaconda3\lib\site-packages\sklearn\svm\base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 's
cale' in version 0.22 to account better for unscaled features. Set gamm
a explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[30]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [31]:
```
# now we use the model to predict new values for the subset data

yhat_b = clf_b.predict(X_b_test)
yhat_b [0:5]
```

Out[31]:
```
array([2150000,  101726,  590000, 1750000, 1750000])
```

Evaluation

In [32]:
```
from sklearn.metrics import f1_score
f1_score(y_b_test, yhat_b, average='weighted')
```

```
C:\Users\Munazzam\Anaconda3\lib\site-packages\sklearn\metrics\classific
ation.py:1437: UndefinedMetricWarning: F-score is ill-defined and being
set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Munazzam\Anaconda3\lib\site-packages\sklearn\metrics\classific
ation.py:1439: UndefinedMetricWarning: F-score is ill-defined and being
set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

Out[32]:
```
0.02
```

In [33]:
```
# Jaccard Index
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_b_test, yhat_b)
```

```
C:\Users\Munazzam\Anaconda3\lib\site-packages\sklearn\metrics\classific
ation.py:635: DeprecationWarning: jaccard_similarity_score has been dep
recated and replaced with jaccard_score. It will be removed in version
0.23. This implementation has surprising behavior for binary and multic
lass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

Out[33]:
```
0.02
```

In [34]:
```
# SVM's accuracy
from sklearn import metrics
import matplotlib.pyplot as plt
print("SVM's Accuracy: ", metrics.accuracy_score(y_b_test, yhat_b))
```

```
SVM's Accuracy:  0.02
```