DS-630 Assignment 2

1. Try a Support Vector Machine regressor (sklearn.svm.SVR) with various hyperparameters such as kernel="linear" (with various values for the C hyperparameter) or kernel="rbf" (with various values for the C and gamma hyperparameters). Don't worry about what these hyperparameters mean for now. How does the best SVR predictor perform? (based on the housing dataset. i.e., "..\handson-ml2\datasets\housing\housing.csv")

Import pandas to read the CSV file of housing data.

```
# Importing Library
import pandas as pd
import numpy as np
```
✓ 3.6s Python

```
# faching data from housing csv
housing_data = pd.read_csv('housing/housing.csv')
```
✓ 0.1s Python

```
# printing top 5 records from the datafarmes
housing_data.head()
```
✓ 0.1s Python

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

```
housing_data.info()
```
✓ 0.1s Python

using shape to get the no of rows and columns of the data frame.

And using describe get the count, average, and median of the column's data.

```
# chaking shape of data from output no of row and column
housing_data.shape
```
✓ 0.1s

```
(20640, 10)
```

```
# return count, mean, standard deviation
housing_data.describe()
```
✓ 0.2s

Here importing a matplotlib for plotting the histogram of the data.

```
housing_data['ocean_proximity'].value_counts()
```

✓ 0.1s

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

```
# importing Matpolt lib for ploting a graph
%matplotlib inline
import matplotlib.pyplot as plt
housing_data.hist(bins=50, figsize=(20,15))
plt.show()
```

✓ 2.9s

Using this code check the correlation of the column using heatmap.

```
# import seaborn for correlation heat map
import seaborn as sns
corr = housing_data.corr()
corr.shape
plt.figure(figsize=(10,10))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'size':15}, cmap='Greens')
```

Here making a group Based on Median Income.

```
# here using cuts making a group based on median income.
housing_data['income_cat'] = pd.cut(x = housing_data['median_income'],
                                    bins = [0, 1.5, 3.0, 4.5, 6.0, np.inf],
                                    labels = [1, 2, 3, 4, 5])
```

✓ 0.1s

Splitting data into Train(80%) and test(20%) data set using sklearn.

```
from sklearn.model_selection import StratifiedShuffleSplit
```
✓ 0.1s

```
# Spliting record for training and testing set

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train_index, test_index in split.split(housing_data, housing_data["income_cat"]):
    strat_train_set = housing_data.loc[train_index]

    strat_test_set = housing_data.loc[test_index]
```
✓ 0.1s

Using this code-making feature engineering for finding how many rooms are needed per household based on total rooms and households. Same bedrooms per room and population pre-household.

```
def feature_engineering(data):
    data["rooms_per_household"] = data["total_rooms"]/data["households"]
    data["bedrooms_per_room"] = data["total_bedrooms"]/data["total_rooms"]
    data["population_per_household"]=data["population"]/data["households"]
    return data
```
✓ 0.1s

Here I am checking for the null values try to fill the missing value or null value using the median of the data

```python
def data_transformation(data):
    # Seperate Labels
    if 'median_house_value' in data.columns:
        labels = data['median_house_value']
        data = data.drop('median_house_value', axis = 1)
    else:
        labels = None
    # Feature Engineering
    fed = feature_engineering(data)
    features = fed.columns.to_list()
    features.remove('ocean_proximity')
    fedc = fed['ocean_proximity'].copy()
    fed = fed.drop('ocean_proximity', axis = 1)
    # Handling Missing Values
    from sklearn.impute import SimpleImputer
    imp = SimpleImputer(strategy='median')
    data_imp = imp.fit_transform(fed)
    # Scaling/Normalizing Data
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    scaled_data = sc.fit_transform(data_imp)

    # Encoding Categorical Variables
    from sklearn.preprocessing import OneHotEncoder
    enc = OneHotEncoder(sparse=False)
    arr = fedc.values
    arr = arr.reshape(-1, 1)
    data_e = enc.fit_transform(arr)
    features = features + enc.categories_[0].tolist()

    output = np.hstack((scaled_data, data_e))

    return output, labels, features
```

✓ 0.1s

Performing Grid search.

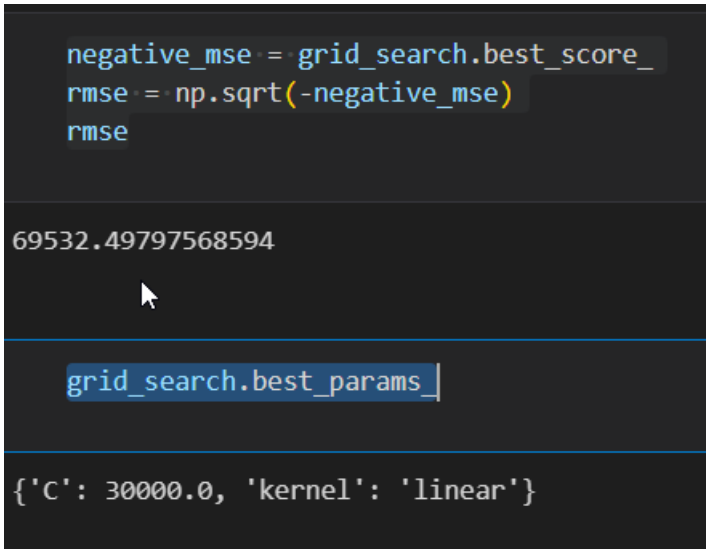
```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = [
        {'kernel': ['linear'], 'C': [10., 30., 100., 300., 1000., 3000., 10000., 30000.0]},
        {'kernel': ['rbf'], 'C': [1.0, 3.0, 10., 30., 100., 300., 1000.0],
         'gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0]},
    ]

svm_reg = SVR()
grid_search = GridSearchCV(svm_reg, param_grid, cv=5, scoring='neg_mean_squared_error', verbose=2)
grid_search.fit(train_data, train_labels)
```

53m 12.6s

Here is the grid search result in its shows the linear kernel is batter performed batter then RBF



Support vector machine regression works on the principle of the Support vector machine with a few differences. SVR tries to find the curve from a given data point. A curve is used to find the march between the vector and the position of the curve instead of the decision boundary.

2. Try to build a classifier for the MNIST dataset that achieves over 97% accuracy on the test set. Hint: the KNeighborsClassifier works quite well for this task; you just need to find good hyperparameter values (try a grid search on the weights and n_neighbors hyperparameters).

Code:

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.keys()
```

✓ 35.2s Python

```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
```

```
# seting up X and y as input and target data.
X, y = mnist["data"], mnist["target"]
```

✓ 0.1s Python

```
X.shape
```

✓ 0.2s Python

```
(70000, 784)
```

```
y.shape
```

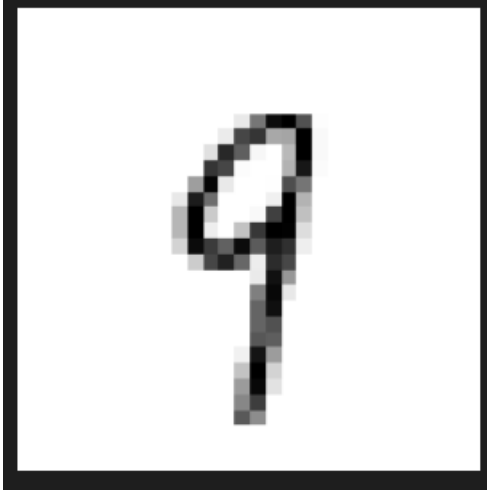
✓ 0.2s Python

```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

img = X[900]
image_shape = img.reshape(28, 28)
plt.imshow(image_shape, cmap=mpl.cm.binary)
plt.axis("off")

plt.show()
```

✓ 1.8s



```
y[900]
```

✓ 0.1s

```
'9'
```

```
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    n_rows = (len(instances) - 1) // images_per_row + 1
    n_empty = n_rows * images_per_row - len(instances)
    padded_instances = np.concatenate([instances, np.zeros((n_empty, size * size))], axis=0)
    image_grid = padded_instances.reshape((n_rows, images_per_row, size, size))
    big_image = image_grid.transpose(0, 2, 1, 3).reshape(n_rows * size,images_per_row * size)
    plt.imshow(big_image, cmap = mpl.cm.binary, **options)
    plt.axis("off")
```

✓ 0.1s

```
import numpy as np
plt.figure(figsize=(12,12))
example_images = X[:200]
plot_digits(example_images, images_per_row=10)
plt.show()
```

✓ 0.3s

```
# seting up the training and testing data set
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

✓ 0.2s Python

```
# using kNeighborsClassifier method for train data
from sklearn.neighbors import KNeighborsClassifier
train_data = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None,
n_jobs=None)
```

✓ 2.2s Python

+ Code + Markdown

```
# fitting up with model
train_data.fit(X_train,y_train)
```

✓ 0.2s Python

```
KNeighborsClassifier()
```

```
# pridicting the model with test data.
y_model=train_data.predict(X_test[:60000])
y_model
```

✓ 55.8s Python

```
array(['7', '2', '1', ..., '4', '5', '6'], dtype=object)
```

```
y_test[:60000]
```

✓ 0.7s

```
array(['7', '2', '1', ..., '4', '5', '6'], dtype=object)
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix( y_test[:60000], y_model )
```

✓ 0.1s

```
array([[ 974,    1,    1,    0,    0,    1,    2,    1,    0,    0],
       [   0, 1133,    2,    0,    0,    0,    0,    0,    0,    0],
       [  11,    8,  991,    2,    1,    0,    1,   15,    3,    0],
       [   0,    3,    3,  976,    1,   13,    1,    6,    3,    4],
       [   3,    7,    0,    0,  944,    0,    4,    2,    1,   21],
       [   5,    0,    0,   12,    2,  862,    4,    1,    2,    4],
       [   5,    3,    0,    0,    3,    2,  945,    0,    0,    0],
       [   0,   22,    4,    0,    3,    0,    0,  988,    0,   11],
       [   8,    3,    5,   13,    6,   12,    5,    5,  913,    4],
       [   5,    7,    3,    9,    7,    3,    1,   10,    2,  962]],
      dtype=int64)
```

```
# finding the accuracy score suing Confusion matrix
from sklearn.metrics import accuracy_score
accuracy_score(y_test[:60000], y_model, normalize=True, sample_weight=None)
```

✓ 0.9s

```
0.9688
```