# Part 8 – Support Vector Machines - Classification

## Hyperparameters

**C** = regularization parameter, inversely proportional to allowed error – default = 1.0. It determines the penalty for misclassifying a training point. It’s essentially a regularization parameter that controls the balance between maximizing the margin and minimizing classification error.

**kernel** = linear, rbf, sigmoid, poly – default = rbf

**gamma** = for rbf kernel only, defines the reach or spread of the kernel function. Specifically, it controls how much influence each training point has on the model.

* **High gamma**: A large gamma value means that each training point has a small radius of influence. This leads to a more complex, "overfitting" model because it tries to fit the data more closely, resulting in a decision boundary that is very wiggly and specific to the training data.
* **Low gamma**: A small gamma value means that each training point has a large radius of influence. This results in a simpler, "underfitting" model with a smoother decision boundary that may not capture the data's complexities effectively.

**degree** = for poly kernel only. The **degree** parameter in the polynomial kernel of Support Vector Machines (SVM) controls the degree of the polynomial used to compute the similarity between data points in the feature space.

### SVR – Support Vector Machine Regression

Include other hyperparameters

# Support Vector Regression - Cement Slump Test

#### Base Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

#### Read in Data

df = pd.read\_csv('../Data/cement\_slump.csv')

df.head()

#### Heatmap to examine correlation

plt.figure(figsize=(8,8), dpi=200)

sns.heatmap(df.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1);

#### Split in variables

df.columns # Examine columns and correct spelling

X = df.drop('Compressive Strength (28-day)(Mpa)', axis=1)

y = df['Compressive Strength (28-day)(Mpa)']

#### Train Test Split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)

#### Scale Data

Support Vector Machines require data to be scaled

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_X\_train = scaler.fit\_transform(X\_train)

scaled\_X\_test = scaler.transform(X\_test)

#### Import Models

from sklearn.svm import SVR, LinearSVR

#### Build Base SVR Model

base\_model = SVR()

base\_model.fit(scaled\_X\_train, y\_train)

base\_preds = base\_model.predict(scaled\_X\_test)

#### Evaluate SVR Model

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

mean\_absolute\_error(y\_test, base\_preds)

np.sqrt(mean\_squared\_error(y\_test, base\_preds))

y\_test.mean() # provides context to error measurements

#### Build Grid Search SVR Model

param\_grid = {'C':[0.001, 0.01, 0.1, 0.5, 1, 2],

'kernel':['linear', 'rbf', 'poly'],

'gamma':['scaled', 'auto'],

'degree':[2,3,4],

'epsilon':[0.001, 0.01, 0.1, 0.5, 1, 2]}

from sklearn.model\_selection import GridSearchCV

svr = SVR()

grid = GridSearchCV(svr, param\_grid)

grid.fit(scaled\_X\_train, y\_train)

grid.best\_params\_

grid\_preds = grid.predict(scaled\_X\_test)

mean\_absolute\_error(y\_test, grid\_preds)

np.sqrt(mean\_squared\_error(y\_test, grid\_preds))

### SVC – Support Vector Machine Classification

#!/usr/bin/env python

# coding: utf-8

# SVM Classification - Wine Project

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv("wine\_fraud.csv")

df.head()

df['quality'].unique()

df['quality'].value\_counts()

sns.countplot(data=df, x='quality');

sns.countplot(data=df, x='quality', hue='type');

sns.countplot(data=df, x='type', hue='quality');

reds = df[df['type']=='red']

whites = df[df['type']=='white']

100\*(len(reds[reds['quality'] =='Fraud'])/len(reds))

100\*(len(whites[whites['quality'] =='Fraud'])/len(whites))

df['Fraud'] = df['quality'].map({'Legit':0, 'Fraud':1})

df3 = df.copy()

df3 = df3.drop('type', axis=1)

df3 = df3.drop('quality', axis=1)

df3.head()

df3.corr()['Fraud'][:-1].sort\_values().plot(kind='bar');

# ## Machine Learning Model - SVC

df = df.drop('Fraud', axis=1)

df['type'] = pd.get\_dummies(df['type'], drop\_first=True)

X = df.drop('quality', axis=1)

y = df['quality']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=101)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_X\_train = scaler.fit\_transform(X\_train)

scaled\_X\_test = scaler.transform(X\_test)

from sklearn.svm import SVC

svc = SVC(class\_weight='balanced')

from sklearn.model\_selection import GridSearchCV

param\_grid = {'C':[0.001, 0.01, 0.1, 0.5, 1]}

grid = GridSearchCV(svc, param\_grid)

grid.fit(scaled\_X\_train, y\_train)

grid.best\_params\_

from sklearn.metrics import confusion\_matrix, classification\_report, ConfusionMatrixDisplay

grid\_preds = grid.predict(scaled\_X\_test)

cm = confusion\_matrix(y\_test, grid\_preds)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix")

plt.show()

print(classification\_report(y\_test, grid\_preds))