Applications of Machine Learning HW3

-by Srilalith Nampally

# Part 1:

## Code:

import numpy as np

from sklearn import model\_selection, neighbors

import pandas as pd

from imblearn.under\_sampling import RandomUnderSampler

def getDF(path):

"""Returns the csv as a pandas dataframe"""

print("Reading . . .")

df = pd.read\_csv(path, low\_memory=False)

print("Done")

return df

def pp(pdframe):

"""Preprocessing Steps

Input - Dataframe

Output - Feature\_DF, Target\_DF"""

pdframe.replace('?', 0, inplace=True) #Replacing '?' values with 0s, since its numeric column

X = pdframe.drop(pdframe.columns[-1], axis=1) #Dropping the target column

Y = pdframe.iloc[:, -1] #Making the Target into a seperate DF

X = X.to\_numpy(np.float64)

Y.replace(('inactive', 'active'), range(2), inplace=True) #Converting target values to numbers

Y = Y.to\_numpy(np.float64)

return X, Y

def KNN\_unbalanced(feature, target, score\_df):

'''KNN for Unbalanced Dataset type'''

trainX, testX, trainY, testY = model\_selection.train\_test\_split(feature, target, test\_size=0.3, random\_state=92313)

#iterate through neighbors and weighting

for nnbrs in [1, 3, 5, 9, 11, 15, 21]: # [1, 3, 5, 9, 11, 15, 21]

for weighting in ['uniform', 'distance']:

clf = neighbors.KNeighborsClassifier(n\_neighbors=nnbrs, weights=weighting)

clf = clf.fit(trainX, trainY.ravel())

accuracy\_test = clf.score(testX, testY)

accuracy\_train = clf.score(trainX, trainY)

print(

f"For {nnbrs}-NN Classifier on (Unbalanced) data with weighting type ({weighting}); \n accuracy for test\_set = {accuracy\_test:.4f} \n accuracy for train\_set = {accuracy\_train:.4f} \n difference in test\_set & train\_set performace = {abs(accuracy\_train - accuracy\_test):.4f}\n")

row = {'Dataset': 'Unbalanced', 'Neighbors': nnbrs, 'Weighting': weighting, 'Accuracy\_test': accuracy\_test,

'Accuracy\_train': accuracy\_train, 'Diff in Train&Test Perf': abs(accuracy\_train - accuracy\_test)}

score\_df.loc[len(score\_df.index)] = row

def KNN\_blanaced(feature, target, score\_df):

'''KNN for Balanced Dataset'''

rus = RandomUnderSampler(random\_state=0) #Used randomundersampler to create the 50-50

X\_resampled, y\_resampled = rus.fit\_resample(feature, target)

trainX, testX, trainY, testY = model\_selection.train\_test\_split(X\_resampled, y\_resampled, test\_size=0.3,

random\_state=92313)

#iterate through neighbors and weighting

for nnbrs in [1, 3, 5, 9, 11, 15, 21]: # [1, 3, 5, 9, 11, 15, 21]

for weighting in ['uniform', 'distance']:

clf = neighbors.KNeighborsClassifier(n\_neighbors=nnbrs, weights=weighting)

clf = clf.fit(trainX, trainY.ravel())

accuracy\_test = clf.score(testX, testY)

accuracy\_train = clf.score(trainX, trainY)

print(f"For {nnbrs}-NN Classifier on (Balanced) data with weighting type ({weighting}); \n accuracy for test\_set = {accuracy\_test:.4f} \n accuracy for train\_set = {accuracy\_train:.4f} \n difference in test\_set & train\_set performace = {abs(accuracy\_train - accuracy\_test):.4f}\n")

row = {'Dataset': 'Balanced', 'Neighbors': nnbrs, 'Weighting': weighting, 'Accuracy\_test': accuracy\_test,

'Accuracy\_train': accuracy\_train, 'Diff in Train&Test Perf': abs(accuracy\_train - accuracy\_test)}

score\_df.loc[len(score\_df.index)] = row

if \_\_name\_\_ == "\_\_main\_\_":

filename = "K8.csv"

print("Making a Dataframe . . .")

df = getDF(filename)

feature, target = pp(df)

score\_df = pd.DataFrame(

columns=['Dataset', 'Neighbors', 'Weighting', 'Accuracy\_test', 'Accuracy\_train', 'Diff in Train&Test Perf'])

print("Accuracy Results: \n")

KNN\_unbalanced(feature, target, score\_df)

KNN\_blanaced(feature, target, score\_df)

print('Performance Table: \n')

print(score\_df)

score\_df.to\_csv('results\_table.csv', index=False)

## Output:

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment\_3> & C:/Users/srico/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/srico/OneDrive/Desktop/Applications of Machine Learning/Assignment\_3/Q1\_KNN.py"

Making a Dataframe . . .

Reading . . .

Done

Accuracy Results:

For 1-NN Classifier on (Unbalanced) data with weighting type (uniform);

accuracy for test\_set = 0.9895

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0105

For 1-NN Classifier on (Unbalanced) data with weighting type (distance);

accuracy for test\_set = 0.9895

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0105

For 3-NN Classifier on (Unbalanced) data with weighting type (uniform);

accuracy for test\_set = 0.9911

accuracy for train\_set = 0.9936

difference in test\_set & train\_set performace = 0.0026

For 3-NN Classifier on (Unbalanced) data with weighting type (distance);

accuracy for test\_set = 0.9911

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0089

For 5-NN Classifier on (Unbalanced) data with weighting type (uniform);

accuracy for test\_set = 0.9911

accuracy for train\_set = 0.9931

difference in test\_set & train\_set performace = 0.0020

For 5-NN Classifier on (Unbalanced) data with weighting type (distance);

accuracy for test\_set = 0.9911

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0089

For 9-NN Classifier on (Unbalanced) data with weighting type (uniform);

accuracy for test\_set = 0.9907

accuracy for train\_set = 0.9923

difference in test\_set & train\_set performace = 0.0017

For 9-NN Classifier on (Unbalanced) data with weighting type (distance);

accuracy for test\_set = 0.9909

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0091

For 11-NN Classifier on (Unbalanced) data with weighting type (uniform);

accuracy for test\_set = 0.9911

accuracy for train\_set = 0.9920

difference in test\_set & train\_set performace = 0.0009

For 11-NN Classifier on (Unbalanced) data with weighting type (distance);

accuracy for test\_set = 0.9911

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0089

For 15-NN Classifier on (Unbalanced) data with weighting type (uniform);

accuracy for test\_set = 0.9907

accuracy for train\_set = 0.9922

difference in test\_set & train\_set performace = 0.0016

For 15-NN Classifier on (Unbalanced) data with weighting type (distance);

accuracy for test\_set = 0.9907

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0093

For 21-NN Classifier on (Unbalanced) data with weighting type (uniform);

accuracy for test\_set = 0.9903

accuracy for train\_set = 0.9922

difference in test\_set & train\_set performace = 0.0019

For 21-NN Classifier on (Unbalanced) data with weighting type (distance);

accuracy for test\_set = 0.9907

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.0093

For 1-NN Classifier on (Balanced) data with weighting type (uniform);

accuracy for test\_set = 0.8023

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.1977

For 1-NN Classifier on (Balanced) data with weighting type (distance);

accuracy for test\_set = 0.8023

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.1977

For 3-NN Classifier on (Balanced) data with weighting type (uniform);

accuracy for test\_set = 0.8140

accuracy for train\_set = 0.8500

difference in test\_set & train\_set performace = 0.0360

For 3-NN Classifier on (Balanced) data with weighting type (distance);

accuracy for test\_set = 0.8140

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.1860

For 5-NN Classifier on (Balanced) data with weighting type (uniform);

accuracy for test\_set = 0.7907

accuracy for train\_set = 0.8450

difference in test\_set & train\_set performace = 0.0543

For 5-NN Classifier on (Balanced) data with weighting type (distance);

accuracy for test\_set = 0.7907

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.2093

For 9-NN Classifier on (Balanced) data with weighting type (uniform);

accuracy for test\_set = 0.7442

accuracy for train\_set = 0.7900

difference in test\_set & train\_set performace = 0.0458

For 9-NN Classifier on (Balanced) data with weighting type (distance);

accuracy for test\_set = 0.7558

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.2442

For 11-NN Classifier on (Balanced) data with weighting type (uniform);

accuracy for test\_set = 0.7209

accuracy for train\_set = 0.7950

difference in test\_set & train\_set performace = 0.0741

For 11-NN Classifier on (Balanced) data with weighting type (distance);

accuracy for test\_set = 0.7558

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.2442

For 15-NN Classifier on (Balanced) data with weighting type (uniform);

accuracy for test\_set = 0.7442

accuracy for train\_set = 0.7750

difference in test\_set & train\_set performace = 0.0308

For 15-NN Classifier on (Balanced) data with weighting type (distance);

accuracy for test\_set = 0.7558

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.2442

For 21-NN Classifier on (Balanced) data with weighting type (uniform);

accuracy for test\_set = 0.7209

accuracy for train\_set = 0.7650

difference in test\_set & train\_set performace = 0.0441

For 21-NN Classifier on (Balanced) data with weighting type (distance);

accuracy for test\_set = 0.7558

accuracy for train\_set = 1.0000

difference in test\_set & train\_set performace = 0.2442

Performance Table:

Dataset Neighbors Weighting Accuracy\_test Accuracy\_train Diff in Train&Test Perf

0 Unbalanced 1 uniform 0.989467 1.000000 0.010533

1 Unbalanced 1 distance 0.989467 1.000000 0.010533

2 Unbalanced 3 uniform 0.991057 0.993611 0.002554

3 Unbalanced 3 distance 0.991057 1.000000 0.008943

4 Unbalanced 5 uniform 0.991057 0.993100 0.002043

5 Unbalanced 5 distance 0.991057 1.000000 0.008943

6 Unbalanced 9 uniform 0.990660 0.992333 0.001673

7 Unbalanced 9 distance 0.990859 1.000000 0.009141

8 Unbalanced 11 uniform 0.991057 0.991993 0.000935

9 Unbalanced 11 distance 0.991057 1.000000 0.008943

10 Unbalanced 15 uniform 0.990660 0.992248 0.001588

11 Unbalanced 15 distance 0.990660 1.000000 0.009340

12 Unbalanced 21 uniform 0.990262 0.992163 0.001901

13 Unbalanced 21 distance 0.990660 1.000000 0.009340

14 Balanced 1 uniform 0.802326 1.000000 0.197674

15 Balanced 1 distance 0.802326 1.000000 0.197674

16 Balanced 3 uniform 0.813953 0.850000 0.036047

17 Balanced 3 distance 0.813953 1.000000 0.186047

18 Balanced 5 uniform 0.790698 0.845000 0.054302

19 Balanced 5 distance 0.790698 1.000000 0.209302

20 Balanced 9 uniform 0.744186 0.790000 0.045814

21 Balanced 9 distance 0.755814 1.000000 0.244186

22 Balanced 11 uniform 0.720930 0.795000 0.074070

23 Balanced 11 distance 0.755814 1.000000 0.244186

24 Balanced 15 uniform 0.744186 0.775000 0.030814

25 Balanced 15 distance 0.755814 1.000000 0.244186

26 Balanced 21 uniform 0.720930 0.765000 0.044070

27 Balanced 21 distance 0.755814 1.000000 0.244186

## Results Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | Neighbors | Weighting | Accuracy\_test | Accuracy\_train | Diff in Train&Test Perf |
| Unbalanced | 1 | uniform | 0.9894674085850557 | 1.0 | 0.010532591414944337 |
| Unbalanced | 1 | distance | 0.9894674085850557 | 1.0 | 0.010532591414944337 |
| Unbalanced | 3 | uniform | 0.9910572337042926 | 0.993611040122668 | 0.002553806418375437 |
| Unbalanced | 3 | distance | 0.9910572337042926 | 1.0 | 0.00894276629570745 |
| Unbalanced | 5 | uniform | 0.9910572337042926 | 0.9930999233324814 | 0.0020426896281888762 |
| Unbalanced | 5 | distance | 0.9910572337042926 | 1.0 | 0.00894276629570745 |
| Unbalanced | 9 | uniform | 0.9906597774244833 | 0.9923332481472016 | 0.0016734707227182843 |
| Unbalanced | 9 | distance | 0.9908585055643879 | 1.0 | 0.009141494435612074 |
| Unbalanced | 11 | uniform | 0.9910572337042926 | 0.9919925036204106 | 0.0009352699161180311 |
| Unbalanced | 11 | distance | 0.9910572337042926 | 1.0 | 0.00894276629570745 |
| Unbalanced | 15 | uniform | 0.9906597774244833 | 0.9922480620155039 | 0.0015882845910205612 |
| Unbalanced | 15 | distance | 0.9906597774244833 | 1.0 | 0.009340222575516699 |
| Unbalanced | 21 | uniform | 0.990262321144674 | 0.9921628758838061 | 0.0019005547391320876 |
| Unbalanced | 21 | distance | 0.9906597774244833 | 1.0 | 0.009340222575516699 |
| Balanced | 1 | uniform | 0.8023255813953488 | 1.0 | 0.19767441860465118 |
| Balanced | 1 | distance | 0.8023255813953488 | 1.0 | 0.19767441860465118 |
| Balanced | 3 | uniform | 0.813953488372093 | 0.85 | 0.03604651162790695 |
| Balanced | 3 | distance | 0.813953488372093 | 1.0 | 0.18604651162790697 |
| Balanced | 5 | uniform | 0.7906976744186046 | 0.845 | 0.05430232558139536 |
| Balanced | 5 | distance | 0.7906976744186046 | 1.0 | 0.2093023255813954 |
| Balanced | 9 | uniform | 0.7441860465116279 | 0.79 | 0.04581395348837214 |
| Balanced | 9 | distance | 0.7558139534883721 | 1.0 | 0.2441860465116279 |
| Balanced | 11 | uniform | 0.7209302325581395 | 0.795 | 0.07406976744186056 |
| Balanced | 11 | distance | 0.7558139534883721 | 1.0 | 0.2441860465116279 |
| Balanced | 15 | uniform | 0.7441860465116279 | 0.775 | 0.030813953488372126 |
| Balanced | 15 | distance | 0.7558139534883721 | 1.0 | 0.2441860465116279 |
| Balanced | 21 | uniform | 0.7209302325581395 | 0.765 | 0.04406976744186053 |
| Balanced | 21 | distance | 0.7558139534883721 | 1.0 | 0.2441860465116279 |

## Discussion:

From the above performance table, the best performance is achieved by The following combinations:

1. Unbalanced, 3NN, Uniform, Accuracy on Test = 0.9910572337042926
2. Unbalanced, 3NN, Distance, Accuracy on Test = 0.9910572337042926
3. Unbalanced, 5NN, Uniform, Accuracy on Test = 0.9910572337042926
4. Unbalanced, 5NN, Distance, Accuracy on Test = 0.9910572337042926
5. Unbalanced, 11NN, Uniform, Accuracy on Test = 0.9910572337042926
6. Unbalanced, 11NN, Distance, Accuracy on Test = 0.9910572337042926

The Accuracy on test is same for these 6 combinations.

From the above table, the one with highest Train&Test set performance difference is:

1. Balanced, 9NN, Distance, Accuracy Difference = 0.2441860465116279
2. Balanced, 11NN, Distance, Accuracy Difference = 0.2441860465116279
3. Balanced, 15NN, Distance, Accuracy Difference = 0.2441860465116279
4. Balanced, 21NN, Distance, Accuracy Difference = 0.2441860465116279

Similarly, the least difference is from:

1. Unbalanced, 11NN, Uniform, Accuracy on Difference = 0.0009352699161180311

# Question 2:

# Code:

from sklearn import linear\_model as linmod

from sklearn import metrics, model\_selection, feature\_selection

from sklearn import preprocessing as preproc

from sklearn import impute

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

def showStats(W, X, Yact, Ypred, mlr):

"""Shows stats about the linear regression(Used from lecture notes)"""

print("R2 = %f, MSE = %f" % (mlr.score(X, Yact), metrics.mean\_squared\_error(Yact, Ypred)))

print("W: ", W)

def GetDF(path):

"""Just reads the excel file and returns it as a Dataframe"""

DF = pd.read\_excel(path)

return DF

def prerocessDF(df):

""""PREPROCESS STEPS"""

"""Dropping ID Columns"""

IDName = ["playerID", "yearPlayer"]

df.drop(columns=IDName, inplace=True)

"""Splitting Target and Features right here to make imputation easier"""

#X = df.drop(columns=targetName, axis=1)

#Y = df[targetName]

""""Handling Categorical Columns"""

"""I decided to use OneHotEncoding, since the cardinality of categorical columns is not too high (35, 2)"""

object\_cols = df[['lgID', 'teamID']] # Selecting the columns to be encoded

encoder = preproc.OneHotEncoder()

encoded\_objects = encoder.fit\_transform(object\_cols)

encoded\_df = pd.DataFrame(encoded\_objects.toarray(), columns=encoder.get\_feature\_names\_out(['lgID', 'teamID']))

df.reset\_index(drop=True, inplace=True)

df = pd.concat([df.drop(['lgID', 'teamID'], axis=1), encoded\_df], axis=1)

"""Handle Missing Values and Outliers"""

# I ran the Data\_statistics.py report generator from HW1 on BattingSalariesData.xlsx, so I know in which column

# the oultiers are, and that they are 99999 and 9999,

# I also know that the target Salary has about a 3rd of its values missing(from the report)

"""So I decided to use the KNN imputator for outliers and missing values"""

df.replace([99999, 9999], np.nan, inplace=True)

# Initialize the KNNImputer

KNNimputer = impute.KNNImputer(missing\_values=np.nan, n\_neighbors=5)

transform\_df = KNNimputer.fit\_transform(df)

df\_imputed = pd.DataFrame(transform\_df, columns=df.columns, index=df.index) #converting the np array back into a dataframe

# Y\_imputed\_series = pd.Series(Y\_imputed.flatten())

return df\_imputed

def tryVariableSelection(xtrain, xtest, ytrain, ytest, sel, dir, labels, model):

if sel == 'sequential':

selector = feature\_selection.SequentialFeatureSelector(model, direction=dir, n\_features\_to\_select='auto', scoring='r2')

elif sel == 'RFE':

selector = feature\_selection.RFE(model, step=1)

elif sel == 'RFECV': # This works Best

selector = feature\_selection.RFECV(model, step=1, cv=5)

selector.fit(xtrain, ytrain)

newxtrain = selector.transform(xtrain)

newxtest = selector.transform(xtest)

model.fit(newxtrain, ytrain)

print("\nUsing: {0}".format(labels[selector.get\_support() == True]))

print("Method {0}: Training set R-sq={1:8.5f}, test set MSE={2:e}".format(dir, model.score(newxtrain,

ytrain), metrics.mean\_squared\_error(ytest, model.predict(newxtest))))

def linregTotal(X, Y, labels):

doScale = True

trainX, testX, trainY, testY = model\_selection.train\_test\_split(X, Y, test\_size=0.2, random\_state=22222)

if doScale:

scalerX = preproc.MinMaxScaler(feature\_range=(-1, 1))

scalerX.fit(trainX)

trainX = scalerX.transform(trainX)

scalerX.fit(testX)

testX = scalerX.transform(testX)

mlr = linmod.LinearRegression() # creates the regressor object

#Uncomment for Vanilla Linear Regression

'''mlr.fit(trainX, trainY)

print("On Train Set\n")

print("R2 is {}; RMSE is {}".format(mlr.score(trainX, trainY), metrics.mean\_squared\_error(trainY, mlr.predict(trainX))))

print("W = ", mlr.intercept\_, mlr.coef\_)

print("\nOn Test Set\n")

print("R2 is {}; RMSE is {}".format(mlr.score(testX, testY), metrics.mean\_squared\_error(testY, mlr.predict(testX))))

print("W = ", mlr.intercept\_, mlr.coef\_)'''

#tryVariableSelection(trainX, testX, trainY, testY, 'sequential', 'forward', labels, model=mlr)

#tryVariableSelection(trainX, testX, trainY, testY, 'sequential', 'backward', labels, model=mlr)

#tryVariableSelection(trainX, testX, trainY, testY, 'RFE', 'RFE', labels, model=mlr)

tryVariableSelection(trainX, testX, trainY, testY, 'RFECV', 'RFECV', labels, model=mlr) #This was giving me the best result so I Ended up using only RFECV

#Uncomment if you want to try ridge regression

'''print("\n\nUsing Ridge Regression")

ridge\_reg = linmod.Ridge(alpha=10, solver='sag', random\_state=22222)

ridge\_reg.fit(trainX, trainY)

print("Training complete; {} epochs".format(ridge\_reg.n\_iter\_))

print("On Train Set\n")

print("R2 is {}; RMSE is {}".format(ridge\_reg.score(trainX, trainY), metrics.mean\_squared\_error(trainY, ridge\_reg.predict(trainX))))

print("W = ", ridge\_reg.intercept\_, ridge\_reg.coef\_)

print("\nOn Test Set\n")

print("R2 is {}; RMSE is {}".format(ridge\_reg.score(testX, testY), metrics.mean\_squared\_error(testY, ridge\_reg.predict(testX))))

print("W = ", ridge\_reg.intercept\_, ridge\_reg.coef\_)'''

#Uncomment If you want to try polynomial feauture enhanced regression

'''poly = preproc.PolynomialFeatures(2) # object to generate polynomial basis functions

bigTrainX = poly.fit\_transform(trainX)

mlr = linmod.LinearRegression() # creates the regressor object

mlr.fit(bigTrainX, trainY)

showStats(np.append(np.array(mlr.intercept\_), mlr.coef\_), bigTrainX, trainY, mlr.predict(bigTrainX), mlr)'''

def LinRegByYear(df):

# Initialize lists to store results

years = []

dataset\_sizes = []

mse\_values = []

r2\_values = []

# Print header for the outputs

print(f"{'Year':<6} {'Size':<6} {'MSE':<20} {'R2':<6}")

# Loop through each year

for year in df['yearID'].unique():

# Filter the data for the given year

yearly\_data = df[df['yearID'] == year]

# Define the predictors and target variable

X = yearly\_data.drop(['Salary', 'yearID'], axis=1)

y = yearly\_data['Salary']

# Split the data

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.2, random\_state=22222)

#Scaler Normalization USed from Class notes

scalerX = preproc.MinMaxScaler(feature\_range=(-1, 1))

scalerX.fit(X\_train)

X\_train = scalerX.transform(X\_train)

scalerX.fit(X\_test)

X\_test = scalerX.transform(X\_test)

# Initialize Linear Regression

model = linmod.LinearRegression()

#Using RFECV variable selection (gave me best result in previous reg)

selector = feature\_selection.RFECV(model, step=1, cv=5)

selector.fit(X\_train, y\_train)

newxtrain = selector.transform(X\_train)

newxtest = selector.transform(X\_test)

#Fitting Model

model.fit(newxtrain, y\_train)

predictions = model.predict(newxtest)

# Evaluation

mse = metrics.mean\_squared\_error(y\_test, predictions)

r2 = metrics.r2\_score(y\_test, predictions)

# Store the results

years.append(year)

dataset\_sizes.append(len(yearly\_data))

mse\_values.append(mse)

r2\_values.append(r2)

# Print results for each year

print(f"{year:<6} {len(yearly\_data):<6} {mse:<20} {r2:<6}")

# Plotting

plt.figure(figsize=(10, 6))

plt.plot(years, mse\_values, marker='x')

plt.title('MSE vs Year for Salary Prediction')

plt.xlabel('Year')

plt.ylabel('Mean Squared Error')

plt.grid(True)

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

path = "BattingSalariesData.xlsx"

print("Creating DF . . .")

df = GetDF(path)

print("Got Dataframe ", type(df), "Size:", df.shape)

print("\nPerforming pre-processing . . .")

ndf = prerocessDF(df)

print("Pre-Processed Dataframe ", type(df), "Size:", df.shape)

X = ndf.drop(columns='Salary', axis=1)

Y = ndf['Salary']

labels = X.columns

print("\nLinear Regression On Total Dataset. . .\n")

linregTotal(X, Y, labels)

print("\nLinear Regression By Year . . .\n")

LinRegByYear(ndf)

## Output:

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment\_3> & C:/Users/srico/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/srico/OneDrive/Desktop/Applications of Machine Learning/Assignment\_3/Q2\_LinearRegression.py"

Creating DF . . .

Got Dataframe <class 'pandas.core.frame.DataFrame'> Size: (45174, 24)

Performing pre-processing . . .

Pre-Processed Dataframe <class 'pandas.core.frame.DataFrame'> Size: (45174, 22)

Linear Regression On Total Dataset. . .

Using: Index(['yearID', 'stint', 'G', 'R', 'H', '2B', '3B', 'HR', 'RBI', 'SB', 'CS',

'BB', 'SO', 'IBB', 'HBP', 'SH', 'SF', 'GIDP', 'lgID\_AL', 'lgID\_NL',

'teamID\_ANA', 'teamID\_ARI', 'teamID\_ATL', 'teamID\_BAL', 'teamID\_BOS',

'teamID\_CAL', 'teamID\_CHA', 'teamID\_CHN', 'teamID\_CIN', 'teamID\_CLE',

'teamID\_COL', 'teamID\_DET', 'teamID\_FLO', 'teamID\_HOU', 'teamID\_KCA',

'teamID\_LAA', 'teamID\_LAN', 'teamID\_MIA', 'teamID\_MIL', 'teamID\_MIN',

'teamID\_ML4', 'teamID\_MON', 'teamID\_NYA', 'teamID\_NYN', 'teamID\_OAK',

'teamID\_PHI', 'teamID\_PIT', 'teamID\_SDN', 'teamID\_SEA', 'teamID\_SFN',

'teamID\_SLN', 'teamID\_TBA', 'teamID\_TEX', 'teamID\_TOR', 'teamID\_WAS'],

dtype='object')

Method RFECV: Training set R-sq= 0.24694, test set MSE=7.082991e+12

Linear Regression By Year . . .

Year Size MSE R2

1985.0 998 95554212936.4347 0.20356744163828122

1986.0 1017 117726956958.33241 -0.11864055210783686

1987.0 1048 240364793306.30298 0.06183342624244759

1988.0 1035 171604962495.80096 -0.06936785166234882

1989.0 1073 207637086676.4922 0.13509571013210286

1990.0 1115 200440195056.8861 0.2884513135773672

1991.0 1086 656908364835.9406 0.10945729326087217

1992.0 1066 1158875477579.5522 0.019319352902644082

1993.0 1180 1388464985643.5925 0.05603231041060919

1994.0 1030 1398900496693.449 0.10568384420386401

1995.0 1253 2087713779706.3792 0.015221165245653978

1996.0 1253 1354447495203.85 0.10757701434840594

1997.0 1236 2450926690186.016 0.2410836922640982

1998.0 1322 1992210470430.5984 0.3828424151814731

1999.0 1299 2611883356308.55 0.2390100985092728

2000.0 1384 3956634027481.694 0.1442994542490782

2001.0 1339 5135962148060.1455 0.16730669055693026

2002.0 1319 6062179817040.308 0.03997667899976143

2003.0 1347 6918021046498.614 0.0090632909460453

2004.0 1346 8895908457850.07 0.04032094760693061

2005.0 1330 7005528894685.886 0.24038930545103965

2006.0 1377 12992096654852.115 0.06933948442372506

2007.0 1385 8173088359140.217 0.15064822270628275

2008.0 1385 9630877215834.05 0.0982917800288926

2009.0 1388 11173649476593.96 -0.027719559899203716

2010.0 1356 13905660106013.656 0.1157922491271991

2011.0 1389 14058728079894.92 0.12036354427537532

2012.0 1408 16377945610846.875 0.12652934563714158

2013.0 1409 15749545215538.67 0.023663941674605393

2014.0 1435 19120297990707.527 -0.014772232955768239

2015.0 1486 14360470654582.578 0.1357999445285908

2016.0 1483 19721179284314.25 0.10620739404266433

2017.0 1494 6592582964525.642 -0.12997339871401614

2018.0 1535 4762516336116.612 -0.06970556355032254

2019.0 1568 4980623795840.871 -0.04153483624012955

## Plot:

A graph showing the growth of salary

Description automatically generated

The Year for which salary was most predictable is 2016 which is followed by 2014.