-UNITE!-
University Network for Innovation, Technology and Engineering

School of Science
Department of Computer Science

unite!

Human-Centered Machine Learning Course

Predictive analysis of area burned by forest fires using Machine Learning techniques

Professor:
Alex Jung

Authors:
Pablo De Ramon
Giang Le
Vasco Pearson
Amir Ingher
Sergi Garcia
Iván Quirante

Academic Course 2022-2023

1 Introduction

While climate change has been of great concern during the last few decades, in the immediate present society is beginning to experience the brutal consequences scientists had anticipated years ago. Rapidly rising temperatures, longer summers, shorter winters, and heat waves are harmful byproducts of a rapidly growing population, over-consumption and lack of environmental regulation among many other interrelated factors. Critically, this radical shift in climate conditions has led to a global increase in the number of wildfires (and the amount of land ravaged by them), especially in many European countries, which is the central focus of this project.

Forest fires are a major environmental issue not only because they create economical and ecological damage, but also because they directly threaten human life. Therefore it is of utmost importance to accurately predict where fires are likely to occur and how much territory will be affected. Properly harnessing Machine Learning techniques will be a powerful tool to achieve this critical task. In particular, we aim to predict the area damaged during a forest fire by utilizing real-time and non-costly meteorological data.

This report is organized as follows: Section 1 is a brief introduction of the problem tackled in the project (forest fires), as well as an outline of the different parts of the document. In Section 2, the database is presented to identify the data points and the features, as well as the label to be predicted. Finally, the loss functions used are also presented along with the values used as benchmarks. In Section 3, we discuss the methodology used (regression methods) to gather all the data and its later pre-processing along with the chosen models, the hyper-parameters of training algorithms and finally the model validation technique. In section 4, a comparison of training and validation errors for all models is made in order to choose the best one, which is finally discussed in section 5, where the test-set error is analysed with full transparency and possible improvements are suggested.

2 Problem Formulation

In this study we consider forest fire data from the Montesinho natural park, which is located in the Trás-os-Montes northeastern region of Portugal. The database consists of a total of 517 rows and 13 columns, each data point being a set of environmental conditions collected from a specific area of the park at a certain month and day of the week. The data was collected between January 2000 and December 2003 every time a forest fire occurred. Columns ranging from 1 to 12 represent a certain feature of the data and are shown in table 1:

Attribute	Description
X	x-axis coordinate (from 1 to 9)
Y	y-axis coordinate (from 1 to 9)
month	Month of the year
day	Day of the week
FFMC	Fine Fuel Moisture Code
DMC	Duff Moisture Code
DC	Drought Code
ISI	Initial Spread Index
temp	Outside temperature (^{o}C)
RH	Outside relative humidity (%)
wind	Outside wind speed (km/h)
rain	Outside rain (mm/mm^2)
area	Total burned area (ha)

Table 1: Data features (adapted from [2])

Also shown in the last row of table 1 is the label or quantity of interest: the total burned area in *ha* of each incident. Hence, Machine Learning methods for Regression have been implemented due to the numerical characteristic of the output. The features in this dataset are all numerical with the exception of the month and day, which are categorical features with 12 and 7 categories, respectively. All the numerical features are continuous except for the coordinates which are discrete (values from 1 to 9). FFMC, DMC, DC and ISI are components of the forest Fire Weather Index (FWI), a Canadian system for rating fire danger. FFMC represents the moisture content surface litter and influences ignition of a fire and its spread, while the DMC and DC represent the moisture content of shallow and deep organic layers, which affect fire intensity. The ISI is a score that correlates with fire velocity spread.

3 Methods

Before finding the best predictive model, the team conducted some previous research on the data. According to [2], the information used in the experiment was collected from January 2000 to December 2003 and it was built using two sources. Every time a forest fired occurred, several features such as time, date, spatial location within a 9×9 grid of the Montesinho natural park, the type of vegetation involved, the six components of the FWI system (FFMC, DMC, DC, ISI, BUI and FWI) and the total burned area were registered in the first one. On the other hand, the second database collected the temperature, relative humidity, wind speed and accumulated precipitation in periods of 30 minutes. However, some changes were made beforehand: the source's owners added personally the **month** and **day** of the records after consulting with Montesinho fire inspector, and both the BUI and FWI attributes were discarded since they can only be obtained from previous variables as stated in Section 1.

Finally, both sources were combined and the resulting database was made up of 517 rows - i.e, data points - and 13 columns.
The dataset contains a total of 247 samples classified as zero area burnt, which actually correspond to fires smaller than 100m^2. For this reason, (and that small fires are more frequent) a positive skew is observed if a histogram of the label is plotted. In order to reduce this effect and improve symmetry, a logarithm transformation has been applied to the target attribute: $\texttt{area'} = \ln(\texttt{area}+1)$, resulting in the distribution of frequencies depicted in Figure 1.

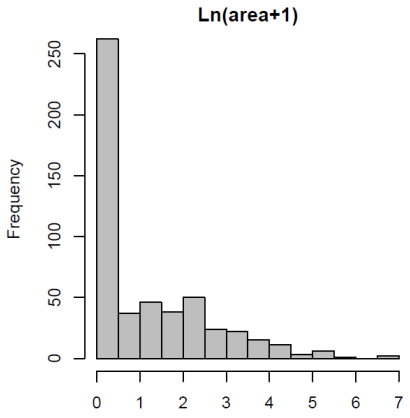

Figure 1: Histogram of ln(area+1) (adapted from [2])

Finally, a last step has been made during the data pre-processing: the month and the day have been transformed to numbers between the

ranges 1-12 and 1-7, respectively.

As for feature selection, the final models use the 12 attributes to predict the amount of area burnt because we observe that lower validation errors are obtained when compared with smaller feature subsets.

Since it is a Regression problem, at the beginning two loss functions were considered by the team:

$$\texttt{Mean Squared Error} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 \quad (1)$$

$$\texttt{Mean Absolute Error} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}| \quad (2)$$

However, MSE was finally chosen because, on the one hand, using a loss function more sensitive to big differences, which usually appear when predicting big fires, would make the model try to fit better these datapoints and focus less on small burned areas, which are over-represented in the dataset. On the other, the MSE is a widely-used loss function because it has got appealing characteristics in terms of computational efficiency: it's a convex and differentiable function.

Finally, [2] has been used as a reference document for benchmarking purposes. Their results show that if an error of $1ha$ is accepted when testing the final models, 46% of the examples are accurately predicted, whereas this value increases up to 61% if the admissible error is $2ha$. These conclusions have been used as a baseline to compare the final model of this project in the test set.

As for the model selection, the group first plotted a correlation matrix to find whether there was correlation between the features and the label. All the correlation coefficients were smaller then 0.1 and that is why it was decided to use all the regression models learnt during the course and see which one achieved the smallest validation error.
The dataset has been split into two groups with `train_test_split()` method from `sklearn.model_selection`: one to train and validate each one of the models, and the other to test the final candidate and compare it with benchmark values, whose size corresponds to the 20% of that of all the dataset. After that, in order to reduce overfitting and improve the generalisation capability of the model, a 10-fold cross validation was implemented to the training and validation set. A function named `kfold()` has been created, which is able to return the mean of the squared error of the folds given an specific estimator and the training-validation set.
The different methods and the main motivation lying behind their selection are explained below to put the reader in the proper context before showing the results in Section 4:
Linear Regression (LR), **Huber Regression (HR)**, **Ridge Regression (RR)**, and **Lasso Regression (RR)**, which use linear models to fit the training data, were tried even though the correlation matrix did not show any clear linear relation between features and labels. The fact that we want to predict a continuous dependent variable from a number of independent variables was the motivation behind trying these regression models. Later, methods with non-linear models were applied to see if they were the most appropriate for the data. The **Polynomial Regression (PR)**, **Decision Trees (DT) (with Naive (Np) and Cost-Complexity (CCp) pruning)**,

Support Vector Regression (SVR) and **Artificial Neural Networks (ANN)** were tried, expecting to get the best results with SVR in the same way as [2].

4 Results

Once all the data has been properly preprocessed and randomly split for calculation, all the models were trained and validated. The `kfold()` function has been applied, and for each model the mean of 10 validation error estimates has been computed. The results obtained are summarized in the following table:

Model	AVE	ATE	Comments
LR	2.27	1.82	
HR	2.18	1.9	
RR	2.27	1.17	Best alpha value: $\alpha = 0.001$
LaR	1.89	–	Best alpha value: $\alpha = 1.43$
PR	2.27	1.82	Best degree: d = 1
DT (Np)	**1.83**	**1.81**	min_samples_leaf = 75
DT (CCp)	1.89	1.89	ccp_alpha = 0.058
SVR	**1.85**	**0.25**	$\gamma = 2000, \mathtt{C} = 1.2, \epsilon = 0.362$
ANN	1.95	1.87	4 hidden layers

Table 2: Average validation (AVE) and Training (ATE) Errors for each model

As it can be observed in table 2, the best mean (in **bold**) for the training and validation errors was obtained with the Decision Tree with Naive pruning, closely followed by Support Vector Regression. Since the initial criteria used to compare between the different models was the smallest validation error, the SVR and the Decision Tree were selected as the candidate model to test the test set.

The DT (Np) results were obtained by fitting the hyperparameter `min_samples_leaf` using cross validation on the training-validation set, looking to minimize the squared loss. This pruning works by restricting the tree from growing fully (and therefore overfitting) by simply setting a minimum required amount of data points per terminal node of the tree, hence its "Naive" designation.

To achieve the SVR results, the features were firstly scaled using the `sklearn.preprocessing.StandardScaler()` because the SVR model considers distances between observations and these may vary between scaled and non-scaled data. Standardizing the features makes the model more flexible to new values that are not yet seen in the dataset, besides it brings a higher accuracy [4]. Next, the parameters of the class `sklearn.svm.SVR()` have been tuned until the optimum value has been reached. Following the results obtained by Pablo Cortez and Aníbal Morais in [2], values of $C = 3$ and $\epsilon = 3 \cdot \hat{\sigma} \cdot \sqrt{\frac{\ln N}{N}}$ have been used, where $\hat{\sigma}$ is the standard deviation of the label vector used for training and validation and N its length. Afterwards, a for-loop has been implemented to find the best $\gamma \in \{2^{-9}, 2^{-7}, 2^{-5}, 2^{-3}, 2^{-1}, 2, 20, 200, 2000\}$.

Finally, the two candidate models have been used to make predictions over the test set. As mentioned in the previous section, the test set was obtained after splitting the data with `train_test_split` and, giving to it a size corresponding to the 20% of that of the whole dataset. This percentage has been selected following the Pareto principle[1], which is a good crytheria according to [5].

With DT (Np) we have obtained an MSE on the test set of 2.22, with a prediction accuracy

[1] About 20% of all the cases are responsible of 80% of the problem

of 39% considering an absolute error under 2 ha. Using SVR, the MSE in the test set has been 2.2, with 69.23% of the samples being accurately predicted if an absolute error of $2ha$ is admitted. It is worth mentioning that this result is relatively higher than that presented in [2].

5 Conclusion

In conclusion, the results obtained with the SVR model have been satisfactory. First of all, the training error (0.25) is fairly small, which means that the inner algorithm has properly fit the data points of the training set. Secondly, a slightly higher validation error (1.85) compared with the training one demonstrates that some overfitting might have occurred and the model could be improved in order to get better generalisation characteristics. Thus, we believe that collecting more training data is a promising task that could solve this overfitting problem. Another interesting approach would be to generate more data, using data augmentation techniques. The error on the test set differs from the validation error in 0.7 units (MSE = 2.2), which reveals that even though the model is slightly overfitted, it still has the ability to generalize outside the training data. When an absolute error of $1ha$ was admitted on the test set, the percentage of accuracy was 12.5%, which is quite smaller than the 46% achieved in [2]. However, this value increases up to 69.23% when the admissible value changes to $2ha$ as mentioned in the previous section, which is a better result than the established benchmark (61%). As can be seen, there is a big difference between the two obtain percentages, so there is room for improvement in the robustness of the model before the maximum absolute error accepted.

References

[1] Sklearn webpage. *Supervised learning.* https://scikit-learn.org/stable/supervised_learning.htmlsupervised-learning.(Visited 8/07/2022)

[2] Paulo Cortez and Aníbal Morais. January 2007. *A Data Mining Approach to Predict Forest Fires using Meteorological Data.* Department of Information Systems/RD Algoritmi Centre, University of Minho, 4800-058 Guimarães, Portugal. (Visited 10/07/2022)

[3] Benjamin Naibei. 04/02/2022. *Getting started with Support Vector Regression in Python.* https://www.section.io/engineering-education/support-vector-regression-in-python/. (Visited 17/08/2022)

[4] A. Aylin Tokuç. 25/08/2021. *Why Feature Scaling in SVM?.* https://www.baeldung.com/cs/svm-feature-scaling.(Visited 24/08/2022)

[5] jmount. 20/01/2021. What is a good test set size?. https://www.r-bloggers.com/2021/01/what-is-a-good-test-set-size. (Visited 27/08/2022)

Appendix

Python code

```
#--Importing libraries--
import time
import numpy as np
import pandas as pd
import keras
from matplotlib import pyplot as plt
import seaborn as sb
import tensorflow as tf
from numpy.lib.function_base import median
from google.colab import drive
from keras import layers
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import HuberRegressor
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.svm import SVR,SVC
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error as mae
from sklearn.preprocessing import StandardScaler, PolynomialFeatures

#--Loading the data--
def mount_folder():
    drive.mount('https://drive.google.com/file/d/1GI4nkTM6L60gukYNVoe6tAjjK_4ZP4UX
    /view?usp=sharing')

def load_data():
    # Define the file path
    path = './shared/forestfires.csv'
    # Load the data as a Pandas DataFrame
    df = pd.read_csv(path)

    return df

def load_data_from_github():
  url = 'https://raw.githubusercontent.com/erasherra/Forest_fire_prediction/main/
    forestfires.csv'
  df = pd.read_csv(url)
  return df
```

```
43
44 df = load_data_from_github()
45
46 #--Data pre-processing--
47 df.head() #View dataset
48 df.isnull().sum() #Check null values
49 #Convert categorical values to numerical
50 df.day.replace(('mon','tue','wed','thu','fri','sat','sun'),(1,2,3,4,5,6,7),
       inplace=True)
51 df.month.replace(('jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov
       ','dec'),(1,2,3,4,5,6,7,8,9,10,11,12), inplace=True)
52 df.head(15) #View new dataset
53
54 #Distribution of area values
55 print('The number of records with a burnt area smaller than 100m^2 is: ',
56       len(df[df['area'] == 0]))
57
58 print('The number of records with burnt area bigger than 100m^2 is: ',
59       len(df[df['area'] != 0]))
60
61 print("The number of records with burnt area between 100m^2 and 1ha is: ",
62       len(df.loc[(df['area'] > 0) & (df['area'] <= 1)]))
63
64 #Distribution of area values
65 hist_0_10 = df['area'].hist(bins=[0,1,2,3,4,5,6,7,8,9,10])
66 hist_0_100 = df['area'].hist(bins=[0,10,20,30,40,50,60,70,80,90,100])
67 hist_0_1000 = df['area'].hist(bins=[0,100,200,300,400,500,600,700,800,900,1000])
68
69 #Corelation between features
70 plt.figure(figsize=(16, 10))
71 corrMatrix = df.corr()
72 sb.heatmap(corrMatrix, annot=True)
73 plt.show()
74
75 #Statistical properties of the data
76 df.corr()['area'].sort_values()
77 df.describe().T
78
79 #Log transformation to the area values
80 df[['area']] = np.log(df[['area']] + 1)
81
82 #Split the data in training, validation and test set
83 #To use all features:
84 X, Y = df[['X', 'Y', 'month', 'day', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', '
       wind', 'rain']], df[['area']]
85 Xtrainval, Xtest, Ytrainval, Ytest = train_test_split(X, Y, test_size=0.20,
       random_state=42)
86 Xtrain, Xval, Ytrain, Yval = train_test_split(Xtrainval, Ytrainval, test_size
       =0.20, random_state=42)
```

```
87
88 #To use temperature, rain, relative humidity and wind speed features:
89 #X, Y = df[['temp', 'RH', 'wind', 'rain']], df[['area']]
90 #Xtrainval, Xtest, Ytrainval, Ytest = train_test_split(X, Y, test_size=0.20,
       random_state=42)
91 #Xtrain, Xval, Ytrain, Yval = train_test_split(Xtrainval, Ytrainval, test_size
       =0.20, random_state=42)
92
93
94 #--Scorer and accuracy functions--
95 #Helper function to compute MSE ('score')
96 def scorermse(model, X, y):
97   y_pred = model.predict(X)
98   score = mean_squared_error(y,y_pred)
99   return score
100
101 #Helper function to compute MAE ('score')
102 def scorermae(model, X, y):
103   y_pred = model.predict(X)
104   score = mae(y,y_pred)
105   return score
106
107 #Accuracy function used in the test set
108 # % of correct predictions with a given absolute/relative error.
109 # err is the absolute error permitted in hectares (1 or 2 ha is what is used in
       the paper)
110 # OR the relative error permitted in %
111 def scoreaccuracy(model, x, y, err_type, err):
112     y_pred = model.predict(x)
113     # Inverse log transform
114     y_pred = np.exp(y_pred) - 1
115     y = np.exp(y) - 1
116     # Replacing negative area elements by 0
117     y_pred[y_pred < 0] = 0
118     y[y < 0] = 0
119     # Make y_pred the same shape as y
120     y_pred = y_pred.reshape(-1, 1)
121     abs_err_vec = np.abs(np.subtract(y, y_pred))
122     if err_type == "abs_err":
123         score = (abs_err_vec < err).sum()/np.size(abs_err_vec)
124     else:
125         # Cannot compute relative error around 0, so we'll take 1/100 hectare abs
       error as good, as they do in the paper
126         # Then we check all other elements relative error
127         score = (abs_err_vec[y == 0] < 1/100).sum() + (np.divide(abs_err_vec[y !=
       0], y[y != 0]) < err/100).sum()
128         score = score / np.size(abs_err_vec)
129         print('Hello: ')
130         print(score)
```

```
131
132     return float(score)
133
134
135 #--Linear Regression--
136 def kfold(reg,X,Y,cv,train_score):
137   cv_resultsmae = cross_validate(reg, X, Y, scoring=scorermae, cv = 10,
      return_train_score = train_score)
138   cv_resultsmse = cross_validate(reg, X, Y, scoring=scorermse, cv = 10,
      return_train_score = train_score)
139
140   # Get train and validation error
141   errors_trainmae = cv_resultsmae['train_score']
142   errors_valmae = cv_resultsmae['test_score']
143   errors_trainmse = cv_resultsmse['train_score']
144   errors_valmse = cv_resultsmse['test_score']
145
146   err_trainmae = np.mean(errors_trainmae)
147   err_valmae = np.mean(errors_valmae)
148   err_trainmse = np.mean(errors_trainmse)
149   err_valmse = np.mean(errors_valmse)
150
151   print('*****************************************************************\nMAE:')
152   print(f"Validation MAE for 10-fold CV folds are: {errors_valmae}")
153   print('-----------------------------------------------------------------')
154   print(f"Average validation MAE for for 10-fold CV folds is: {err_valmae: .2f}")
155   print(f"Average training MAE for for 10-fold CV folds is: {err_trainmae: .2f}")
156   print('*****************************************************************\nMSE:')
157   print(f"Validation MSE for 10-fold CV folds are: {errors_valmse}")
158   print('-----------------------------------------------------------------')
159   print(f"Average validation MSE for for 10-fold CV folds is: {err_valmse: .2f}")
160   print(f"Average training MSE for for 10-fold CV folds is: {err_trainmse: .2f}")
161   print('*****************************************************************')
162
163   return err_valmse, np.std(errors_trainmse)
164
165 # Create the linear regression object
166 start=time.time()
167 lin_reg = LinearRegression(fit_intercept=True)
168 kfold(lin_reg, Xtrainval, Ytrainval, 10, True)
169 end = time.time()
170 print("Execution time: " + str(end-start))
171
172
173 #--Huber Regression--
174 # Create the huber regression object
175 hub_reg = HuberRegressor(fit_intercept=True, max_iter = 10000)
176 kfold(hub_reg, Xtrainval, Ytrainval.values.ravel(), 10, True)
177
```

```
178
179 #--Ridge Regression--
180 alpha_vals = np.linspace(0.001, 10, 50)
181 rid_reg_mse = RidgeCV(alphas = alpha_vals, fit_intercept=True, scoring=scorermse,
        cv=10)
182 rid_reg_mae = RidgeCV(alphas = alpha_vals, fit_intercept=True, scoring=scorermae,
        cv=10)
183 rid_reg_mse.fit(Xtrainval, Ytrainval)
184 rid_reg_mae.fit(Xtrainval, Ytrainval)
185
186 print("Best value of alpha using MSE: ", rid_reg_mse.alpha_)
187 print("Average validation MSE for 10-fold CV is: ", rid_reg_mse.best_score_)
188 print("Best value of alpha using MAE: ", rid_reg_mae.alpha_)
189 print("Average validation MAE for 10-fold CV is: ", rid_reg_mae.best_score_)
190
191
192 #--Lasso Regression--
193 alpha_vals = np.linspace(0.001, 10, 50)
194 lasso_reg_mse = LassoCV(alphas = alpha_vals, fit_intercept=True, cv=10)
195 lasso_reg_mse.fit(Xtrainval, Ytrainval.values.ravel())
196
197 print("Best value of alpha using MSE: ", lasso_reg_mse.alpha_)
198 print("Average validation MSE for 10-fold CV is: ", min(lasso_reg_mse.mse_path_.
        mean(1)))
199
200
201 #--Polinomial Regression--
202 for i in [1,2,3,4,5]:
203   poly = PolynomialFeatures(degree=i)
204
205   Xtrainval_poly = poly.fit_transform(Xtrainval)
206
207   poly_reg=LinearRegression(fit_intercept = True)
208   print("Polinomial regression with degree ", i, ":\n", sep = "")
209   kfold(poly_reg, Xtrainval_poly, Ytrainval,10,True)
210   print("\n\n\n")
211
212 #--Decision Tre --
213 #Naive Pruning (Minimum Data points x leaf)
214 # Loop to check for lowest val error as a function of min_samples
215 min_samp_leaf_vec = np.arange(1, 200)
216 err_val_vec = []
217
218 for i in min_samp_leaf_vec:
219     # Create the DT regression object
220     dt_reg = DecisionTreeRegressor(criterion='absolute_error', min_samples_leaf=i)
221
222     # Use 10-fold cross validation
223     cv_results = cross_validate(dt_reg, Xtrainval, Ytrainval, scoring=scorermae,
```

```
    cv=10, return_train_score=True)

    # Get train and validation error
    errors_train = cv_results['train_score']
    errors_val = cv_results['test_score']

    err_train = np.mean(errors_train)
    err_val = np.mean(errors_val)

    err_val_vec.append(err_val)

# Finding the minimum
print("The min samples x leaf that produces the smallest MAE is:",
    min_samp_leaf_vec[np.argmin(err_val_vec)])
print("The lowest 10-fold cv MAE is:", np.min(err_val_vec))

# Plotting the results
plt.figure(figsize=(5, 3))
plt.plot(min_samp_leaf_vec, err_val_vec, label="DT (Trained with A Loss)")

plt.legend(loc='upper right')
plt.xlabel('Minimum samples x leaf')
plt.ylabel('Validation MAE')
plt.tight_layout()
plt.show()

# The same but for MSE
err_val_vec = []
for i in min_samp_leaf_vec:
    # Create the DT regression object
    dt_reg = DecisionTreeRegressor(criterion='squared_error', min_samples_leaf=i)

    # Use 10-fold cross validation
    cv_results = cross_validate(dt_reg, Xtrainval, Ytrainval, scoring=scorermse,
    cv=10, return_train_score=True)

    # Get train and validation error
    errors_train = cv_results['train_score']
    errors_val = cv_results['test_score']

    err_train = np.mean(errors_train)
    err_val = np.mean(errors_val)

    err_val_vec.append(err_val)

# Finding the minimum
print("The min samples x leaf that produces the smallest MSE is:",
    min_samp_leaf_vec[np.argmin(err_val_vec)])
print("The lowest 10-fold cv MSE is:", np.min(err_val_vec))
```

```
269
270 # Plotting the results
271 plt.figure(figsize=(5, 3))
272 plt.plot(min_samp_leaf_vec, err_val_vec, label="DT (Trained with S Loss)")
273
274 plt.legend(loc='upper right')
275 plt.xlabel('Minimum samples x leaf')
276 plt.ylabel('Validation MSE')
277 plt.tight_layout()
278 plt.show()
279
280 #Cost-complexity pruning
281 # MSE
282 err_val_vec = []
283 a_vec = np.linspace(0.01, 0.1, 1000)
284
285 for a in a_vec:
286     # Create the DT regression object
287     dt_reg = DecisionTreeRegressor(criterion='squared_error', ccp_alpha=a)
288
289     # Use 10-fold cross validation
290     cv_results = cross_validate(dt_reg, Xtrainval, Ytrainval, scoring=scorermse,
        cv=10, return_train_score=True)
291
292     # Get train and validation error
293     errors_train = cv_results['train_score']
294     errors_val = cv_results['test_score']
295
296     err_train = np.mean(errors_train)
297     err_val = np.mean(errors_val)
298
299     err_val_vec.append(err_val)
300
301 # Finding the minimum
302 print("The a param that produces the smallest MSE is:", a_vec[np.argmin(
        err_val_vec)])
303 print("The lowest 10-fold cv MSE is:", np.min(err_val_vec))
304
305 # Plotting the results
306 plt.figure(figsize=(5, 3))
307 plt.plot(a_vec, err_val_vec, label="DT (Trained with S Loss)")
308
309 plt.legend(loc='upper right')
310 plt.xlabel('ccp alpha parameter')
311 plt.ylabel('Validation MSE')
312 plt.tight_layout()
313 plt.show()
314
315 # MAE
```

```
err_val_vec = []

for a in a_vec:
    # Create the DT regression object
    dt_reg = DecisionTreeRegressor(criterion='absolute_error', ccp_alpha=a)

    # Use 10-fold cross validation
    cv_results = cross_validate(dt_reg, Xtrainval, Ytrainval, scoring=scorermae,
    cv=10, return_train_score=True)

    # Get train and validation error
    errors_train = cv_results['train_score']
    errors_val = cv_results['test_score']

    err_train = np.mean(errors_train)
    err_val = np.mean(errors_val)

    err_val_vec.append(err_val)

# Finding the minimum
print("The a param that produces the smallest MAE is:", a_vec[np.argmin(
    err_val_vec)])
print("The lowest 10-fold cv MAE is:", np.min(err_val_vec))

# Plotting the results
plt.figure(figsize=(5, 3))
plt.plot(a_vec, err_val_vec, label="DT (Trained with A Loss)")

plt.legend(loc='upper right')
plt.xlabel('ccp alpha parameter')
plt.ylabel('Validation MAE')
plt.tight_layout()
plt.show()

#--Support Vector Regression--
start2= time.time()
#Standarize data
X_s = StandardScaler()
Y_s=StandardScaler()
Xtrainval_svr = X_s.fit_transform(Xtrainval)  #To make the algorithm easier
Ytrainval_svr = Ytrainval

N=Ytrainval_svr.shape[0]
sigma = np.std(Ytrainval_svr)
e=3*sigma*np.sqrt(np.log(N)/N)

#Fit the model using the Radial Basis Function (RBF) kernel
gammas = [2e-9, 2e-7, 2e-5,2e-3,2e-1,2,20,200,2000]
es = np.zeros(len(gammas))
```

```
363
364 m=0
365 for g in gammas:
366   SVR_reg = SVR(kernel = 'rbf',C=3, epsilon=e, gamma=g)
367   print(m)
368   es[m],s=kfold(SVR_reg, Xtrainval_svr, Ytrainval_svr.values.ravel(), 10, True)
369   m+=1
370
371 end2 = time.time()
372
373 minimum = min(es)
374 idx = np.argmin(es)
375 best_gamma = gammas[idx]
376 print("Best gamma value =",best_gamma)
377 print("Lowest validation error =", minimum)
378 print("Execution time: " + str(end2-start2))
379
380
381 plt.plot(gammas,es)
382 plt.xlabel('Gamma')
383 plt.ylabel('Validation Error')
384 plt.show()
385
386 #Testing the model
387 regressor = SVR(kernel = 'rbf', C=1.2, epsilon=e, gamma=2000)
388 regressor.fit(Xtrainval_svr,Ytrainval)
389 Xtest_svr = X_s.transform(Xtest)
390 y_pred=regressor.predict(Xtest_svr)
391 print("MSE: ", mean_squared_error(y_pred, Ytest))
392
393 abs_error_1=scoreaccuracy(regressor, Xtest_svr, Ytest, 'abs_err', 1)
394 abs_error_2=scoreaccuracy(regressor, Xtest_svr, Ytest, 'abs_err', 2)
395 print(abs_error_1*100)
396 print(abs_error_2*100)
397
398 #--Multi-layer Perceptron--
399 def plot_history(history):
400   pd.DataFrame(history).plot(figsize=(7,4))
401   plt.grid(True)
402   plt.xlabel('epoch', fontsize=14)
403   plt.show()
404
405 k_fold = KFold(n_splits=10, shuffle=True, random_state=42)
406 cvscores = []
407
408 for train, test in k_fold.split(Xtrainval, Ytrainval):
409   # Create model
410   model = keras.Sequential()
411   model.add(layers.Dense(200, "relu", input_shape=(12,)))
```

```
  model.add(layers.Dropout(0.2))
  model.add(layers.Dense(200, "relu"))
  model.add(layers.Dropout(0.2))
  model.add(layers.Dense(100, "relu"))
  model.add(layers.Dropout(0.2))
  model.add(layers.Dense(100, "relu"))
  model.add(layers.Dropout(0.2))
  model.add(layers.Dense(20, "relu"))
  model.add(layers.Dense(1))
  # Compile model
  model.compile(optimizer='adam',
              loss=tf.keras.losses.MeanSquaredError(),
              metrics=['MeanSquaredError'])
  # Fit the model
  history = model.fit(Xtrainval.iloc[train], Ytrainval.iloc[train],
    validation_data = (Xtrainval.iloc[test], Ytrainval.iloc[test]), epochs=150,
    batch_size=10, verbose=0)
  # evaluate the model
  scores = model.evaluate(Xtrainval.iloc[test], Ytrainval.iloc[test], verbose=0)
  print("%s: %.2f" % (model.metrics_names[1], scores[1]))
  cvscores.append(scores[1])

print("%.2f (+/- %.2f)" % (np.mean(cvscores), np.std(cvscores)))
plot_history(history.history)
```