Machine Learning: Assignment 3

Name: Marcel Aguilar Garcia Student Id: 20235620 Class: 2021-CT5143

December 17, 2020

1 Machine Learning package

I have chosen Scikit-learn as my Machine Learning package. Scikit-learn is a free software machine learning library for the Python programming language. The main reason of my choice is the fact that Scikit-learn includes a wide range of regression models, metrics functions and tools to handle data [6].

2 Data Preparation

The dataset has been loaded into a DataFrame using read_csv from Pandas library. An additional feature 'bin' has been added. It assigns an interval to the target value. This column will be used to evenly split the examples between the train and test sets. The column is deleted before training the model to avoid data leakage.

```
columns = ["alcohol_by_weight", "rating", "bitterness", "nitrogen", "turbidity", "sugars",
                "degree_of_fermentation", "calorific_value", "density", "pH", "colour", "sulphites"]

df = pd.read_csv('beers_rating.txt',sep = '\t',names = columns)
df['bin'] = pd.qcut(df['rating'], q = 4)
df.head()
```

	alcohol_by_weight	rating	bitterness	nitrogen	turbidity	sugars	degree_of_fermentation	calorific_value	density	pH	colour	sulphites	bin
0	4.476190	60	10.4	0.500	1.6268	15.75	66.470	41	0.9678	3.72	11.825	44.3750	(37.999, 60.0]
1	4.523810	61	10.3	0.530	1.6268	15.12	67.252	26	0.9676	3.73	13.760	36.2500	(60.0, 64.0]
2	4.714286	61	11.5	0.740	0.7968	17.64	82.110	32	0.9698	3.74	14.405	16.5625	(60.0, 64.0]
3	5.238095	60	8.4	0.465	1.6268	15.75	68.816	42	0.9666	3.86	16.770	33.1250	(37.999, 60.0]
4	4.476190	60	11.0	0.730	0.7968	15.12	60.214	9	0.9674	3.59	12.255	10.3125	(37.999, 60.0]

3 Regression Models

3.1 Train, validation and test set

The dataset has been split in three sets [8]:

- The **Train** set has been used to train the models. It has 72% of the dataset examples (673 examples).
- The **Validation** set has been used to find optimal parameters for the models making sure that there is no underfitting or overfitting. It has 18% of the dataset examples (169 examples).
- The **Test** set has been used to provide an unbiased evaluation of the final model. It is the smallest one and has 10% of the dataset examples (94 examples).

```
columns_X = [col for col in columns if col != 'rating']+['bin']

X = df[columns_X]
y = df['rating']

X, X_test, y, y_test = train_test_split(X, y, test_size=0.10,random_state=42,stratify=X['bin'])
X_train, X_val, y_train, y_val = train_test_split(X, y,test_size=0.20,random_state=20,stratify=X['bin'])

X_train.drop('bin', axis=1, inplace=True)
X_test.drop('bin', axis=1, inplace=True)
X_val.drop('bin', axis=1, inplace=True)
```

3.2 Metrics Selection

The following two metrics have been used to compare the performance of the models:

- The **coefficient of determination**, denoted R^2, which is a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model. [2]
- The **Root-mean-square error**, denoted RMSE, which is a measure of the differences between values predicted by the model and the values observed. [5]

Note that these two metrics have been chosen as they show different results. R^2 measures the performance of the model while $RMSE$ measures the error between the target and prediction.

3.3 Models Selection

The two regression models I have selected are:

- **Support Vector Machine Regressor**: SVR is formulated as an optimization problem by first defining a convex ϵ-insensitive loss function to be minimized and finding the flattest tube that contains most of the training instances. [7] [4]
- **AdaBoost Regressor**: Adaboost combines many weak hypotheses that perform slightly better than random guess into a strong hypothesis that has very low error. [1]

I have implemented the following function to calculate R^2 and RMSE in the Train and Validation sets for a given list of models:

```
def df_results(models,X_train,y_train,X_val,y_val):
    results = pd.DataFrame(columns = ['model','Score_Train','MSQR_Train','Score_Validation','MSQR_Validation']
    for model in models:
        model.fit(X_train,y_train)
        row =  {'model':str(type(model).__name__),
                'Score_Train':model.score(X_train,y_train),
                'MSQR_Train':mean_squared_error(y_train, model.predict(X_train)),
                'Score_Validation':model.score(X_val,y_val),
                'MSQR_Validation':mean_squared_error(y_val, model.predict(X_val))}
        results = results.append(row,ignore_index = True)
    return results
```

Before training the models, let's take a look to the correlations between the features and the target: **alcohol_by_weight**: 0.46, **nitrogen**: 0.46, **colour**: 0.34, **turbidity**: 0.32, **bitterness**: 0.19, **sulphites**: 0.17, density: 0.13, degree_of_fetrmentation: 0.13, pH: 0.13, calorific_value: 0.1, sugars: 0.01.

As some of the features have very low correlation, I will be using just the six most correlated features to train the models. This will lower the execution time of training and helps me to test more parameters. Additionally, it may remove noise from the dataset.

As SVR performs better if the data has been previously scaled, I use a pipeline to scale the data using

a StandardScaler before training the model. The results of SVR and AdaBoostRegressor with default paramaters are:

```
X_corr = X_train.copy()
X_corr['rating'] = y_train
best_columns = list(np.abs(X_corr.corr()['rating']).sort_values(ascending = False)[1:7].index)
SVR_pipe = Pipeline([('scaler', StandardScaler()), ('SVR', SVR())])
type(SVR_pipe).__name__ = 'SVR_Pipeline'
AdaB = AdaBoostRegressor(random_state = 0)
df_results([SVR_pipe,AdaB],X_train[best_columns],y_train,X_val[best_columns],y_val)
```

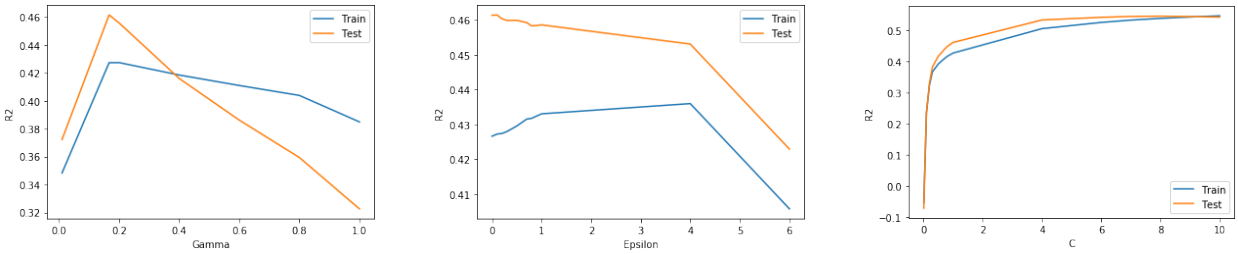
	model	Score_Train	MSQR_Train	Score_Validation	MSQR_Validation
0	SVR_Pipeline	0.427183	57.607175	0.461387	54.556851
1	AdaBoostRegressor	0.531069	47.159540	0.464624	54.228976

In the case of AdaBoost, R^2 and RMSE are considerably better for the train set which probably means that is overfitting. SVR has slightly better results for the validation set, this may be a sign of underfitting but it seems to generalize better. Let's try to optimize the parameters of the models.

3.4 Hyperparameter optimization

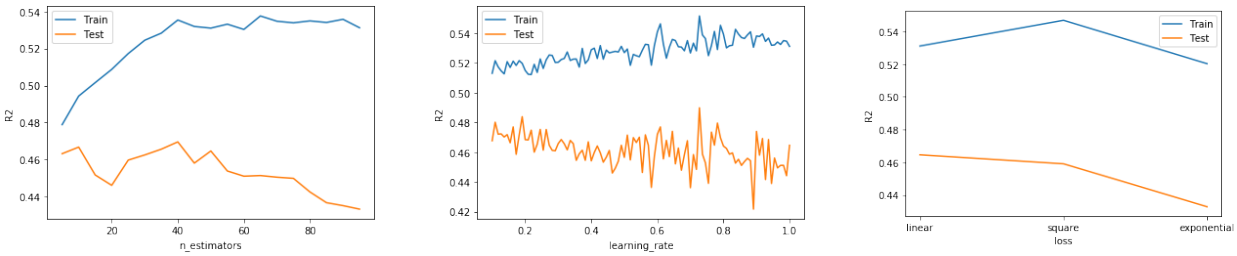
For SVR, I will be optimizing **C**, **Gamma** and **Epsilon**. Gamma decides how far the influence of a single training example reaches. C trades off correct classification of training examples against maximization of the decision function's margin. Epsilon specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. [7]

The following plots show the results of R^2 for different values of each parameter:



In a visual way, it seems that the optimal values for these parameters for the train and test sets are: Gamma between 0.1 and 0.3, Epsilon between 0 and 4 and C between 8 and 10.

For AdaBoostRegressor I will be optimizing the **number of estimators**, the **learning rate** and the **loss function**. The number of estimators is the maximum number of estimators at which boosting is terminated. The learning rate shrinks the contribution of each regressor by learning rate. The loss is the loss function to use when updating the weights after each boosting iteration. [1]



In a visual way, it seems that the optimal values for these parameters for the train and test sets are: n_estimators - between 20 and 60, learning_rate between 0.6 and 0.8 and the loss function seems to work

better as linear or square.

However, the combination of best parameters could be different than the individual optimals. Therefore, I decide to use GridSearchCV to check all possible combinations from them [3]:

```
parameters_SVR = {'SVR__gamma':[0.1,0.15,0.2,0.25,0.3,'auto','scale'],
                  'SVR__epsilon':[0,0.2,0.4,0.6,0.8,1,2,3,4],
                  'SVR__C':[8,9,10]}

parameters_AdaB = {
                   'n_estimators':[10,15,25,40,50],
                   'learning_rate':[0.001,0.01,0.1,0.15,0.25,0.5],
                   'loss':['linear','square','exponential']}


SVR_optimized = GridSearchCV(SVR_pipe,parameters_SVR,cv=5,scoring='r2',n_jobs = -1)
SVR_optimized.fit(X_train[best_columns],y_train)

AdaB_optimized = GridSearchCV(AdaB,parameters_AdaB,scoring='r2',cv=5,n_jobs=-1)
AdaB_optimized.fit(X_train[best_columns],y_train)


df_results([SVR_optimized,AdaB_optimized],X_train[best_columns],y_train,X_val[best_columns],y_val)
```

	model	Score_Train	MSQR_Train	Score_Validation	MSQR_Validation
0	SVR_optimized	0.545192	45.739246	0.548650	45.717866
1	AdaBoost_optimized	0.515796	48.695486	0.470642	53.619450

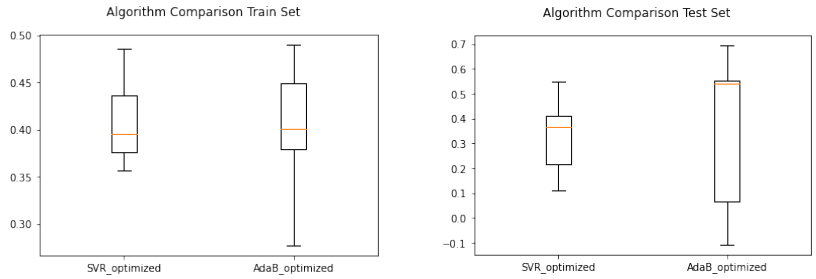
Now, SVR does not have signs of underfitting as the scores in the train and test sets are similar. AdaBoost train results are better but the difference has reduced after optimizing the paramaters. Note that GridSearchCV optimises the results of the train set, therefore this may cause overfitting. However, as I am happy with me results, I decide not to manually change them.

4 Models Evaluation and Comparison

Finally, the test set can be used to have an unbiased evaluation of the two models:

	model	Score_Train	MSQR_Train	Score_Test	MSQR_Test
0	SVR_optimized	0.545192	45.739246	0.515888	49.324875
1	AdaB_optimized	0.515796	48.695486	0.480613	52.919028

While both models have better results on the train set, the difference is very small and they seem to generalize good. SVR performs better in this dataset as R^2 is higher and RMSE lower than AdaBoostRegressor. The following boxplots show the results of each fold for the train and test set using a 5-cross validation.



AdaBoost results show a high variance and while it does better in some folds, other folds have very low R^2. SVR seems consistent across both sets. Therefore, I would suggest to use Support Vector Regressor for this dataset.

References

[1] *AdaBoostRegressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html.

[2] *Coefficient of determination*. URL: https://en.wikipedia.org/wiki/Coefficient_of_determination.

[3] *GridSearchCV*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[4] Rahul Khanna Mariette Awad. *Efficient Learning Machines*. Apress, 2013.

[5] *Mean absolute error*. URL: https://en.wikipedia.org/wiki/Mean_absolute_error.

[6] *Scikit-Learn Machine Learning in Python*. URL: https://scikit-learn.org/stable/.

[7] *SVR*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.

[8] *Wikipedia: Training, validation, and test sets*. URL: https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets.