Prédire la qualité d'un vin

Adrien CAUBEL

1er novembre 2022

Table des matières

1 Description des données

Pour ce TP nous travaillerons sur la prédiction de la qualité des vins rouges. Pour ce faire, nous utiliserons un dataset composé 11 caractéristiques numériques et la valeur cible qu'est la qualité notée de 0 à 10.

Liste des caractéristiques :
- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol

2 Prétraitement des données

Etant donné que toutes les valeurs sont numériques et qu'il n'y a ni valeur manquante ni valeur aberrante, nous n'avons pas réalisé de prétraitement sur les données.

3 Régression linéaire multiple

3.1 Algorithme pour la régression

Nous étudierons en premier la régression linéaire multiple. Le code suivant permet de découper notre dataset en training et testset tout en séparant la valeur cible. De plus, la taille de notre testset sera de 20%. Cette valeur a été déterminée empiriquement.

```
# Faire un data train et data test
from sklearn.model_selection import train_test_split

X = df.iloc[:,:-1]
y = df.iloc[:,-1] # colonne quality isolée

# On sépare notre dataframe en training et test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

Il nous reste plus qu'à entrainer notre modèle et vérifier si les prédictions sur le testset sont bonnes

```
from sklearn import linear_model

# Entrainement du modèle
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# Je passe le dataframe de test au model
predictedQuality = regr.predict(X_test)
```

3.2 Évaluation du modèle avec RMSE

Une fois l'entrainement réalisé nous devons nous assurer que notre modèle est correct. Pour ce faire nous utiliserons la métrique RMSE en comparant les valeurs prédites `predictedQuality` avec les valeurs attendues `y_test`.

```
from sklearn.metrics import mean_squared_error
import math
MSE = mean_squared_error(y_test, predictedQuality)

RMSE = math.sqrt(MSE)
```

Le résultat de ce calcul avoisine les 0.63. Par conséquent, un vin noté 5 sera évalué en moyenne dans l'intervalle [4.4, 5.6]. Cette valeur est assez importante pour un intervalle allant de [0, 10].

3.3 Comprendre ce RMSE élevé

Il fallait donc comprendre pourquoi nous avions un RMSE élevé. Ainsi, nous avons regardé la répartition de la qualité de vin. On remarque que la majorité des vins ont une note de 5 ou 6. Ainsi, notre RMSE est fortement impacté par les vins notés 5, 6 ou 7.

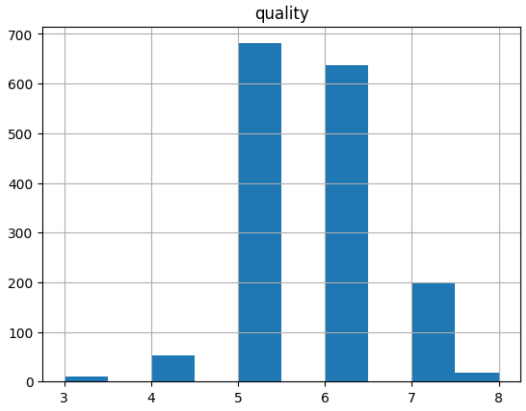


FIGURE 1 – Nombre de vins en fonction de la qualité

3.4 Évaluation de trois sous-ensembles avec RMSE

Au vu de l'histogramme, nous avons donc décidé de découper notre dataset en 3 sous-ensembles en fonction des notes

- Dataset avec une basse qualité : notes de [0, 4]
- Dataset avec une qualité moyenne : notes de [5, 6]
- Dataset avec une haute qualité : notes de [7, 10]

Nous avons réalisé une régression linéaire multiple sur ces trois dataset puis calculé leur RSME respectif :

- Qualité basse RMSE = 0.372
- Qualité moyenne RMSE = 0.446
- Qualité haute RMSE = 0.250

3.5 Conclusion

Les RMSE sur les trois sous-modèles sont plus petits que le RMSE sur le modèle global. Par conséquent il serait intéressant de noter un vin en deux étapes :

1. Classifier le vin : le vin dont on souhaite prédire la qualité va être classé dans une des catégories suivantes, *mauvais*, *bon*, *excellent*. En utilisant un algorithme de classification.
2. Noter le vin : une fois la classe déterminée, nous passons le vin au sous-modèle correspondant (*mauvais*, *bon*, *excellent*) qui étant plus précis nous permettra d'obtenir une note plus précise.

4 Amélioration de nos modèles de régression linéaire multiple

Malgré un RMSE correct pour les vins de haute qualité (0.250) nous nous sommes rendu compte que les valeurs prédites par le modèle étaient toutes proche de 7 même pour les vins ayant une qualité de 8.

Ceci a pu être expliqué par le fait que notre modèle est biaisé. En effet, le nombre d'instances ayant une qualité de 7 est 20 fois supérieure aux instances de qualité 8. Afin d'avoir un meilleur balancement des données, nous créons des instances synthétiques de qualité 8 via l'algorithme *Synthetic Minority Oversampling Technique*.

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_smote_haute, y_smote_haute = sm.fit_resample(X_haute_train, y_haute_train)
```

L'utilisation de cette méthode a eu l'effet escompté, car maintenant les vins ayant une qualité de 8 ont une prédiction avoisinant 8. En contrepartie, nous avons augmenté notre RMSE de 0.04 mais ceci est négligeable.

5 Overfitting, underfitting avec régression linéaire multiple ?

Cette section a été écrite, car nous avions une RMSE de 4 pour les vins de haute qualité et nous souhaitons savoir si cela n'était pas dû à un overfitting ou underfitting. Cependant, l'erreur etait dû à une étourderie dans la programmation. Mais nous trouvons intéressant de discuter de ce travail, car il permet de compléter la validation de modèles.

5.1 Comment déterminer s'il y a overfitting ou underfitting

Pour déterminer s'il y a un problème lors de la phase d'entrainement, nous allons procéder comme suit :

1. Utiliser la *cross-validation* pour vérifier les performances de notre modèle à chaque étape de la validation croisée k-fold.
2. Pour vérifier les performances, nous allons utiliser la métrique *Mean Absolute Error*.
3. Et comparer la MAE d'entrainement à la MAE de test pour chaque *fold*.

5.1.1 Algorithme

```
df_quality_basse = df[df['quality'] < 5]
df_quality_moyenne = df[ (df['quality'] >= 5) & (df['quality'] <= 6) ]
df_quality_haute = df[df['quality'] >= 7]
```

```
# df_selected = df_quality_basse OU df_quality_moyenne OU df_quality_haute
X = df_selected.iloc[:,:-1]
y = df_selected.iloc[:,-1]

kf = KFold(n_splits=5)
mae_train = []
mae_test = []
for train_index, test_index in kf.split(X):
X_train, X_test = X.iloc[train_index], X.iloc[test_index]
y_train, y_test = y[train_index], y[test_index]

model = linear_model.LinearRegression()
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

mae_train.append(mean_absolute_error(y_train, y_train_pred))
mae_test.append(mean_absolute_error(y_test, y_test_pred))
```

5.2 Résultats

5.2.1 Vins de basse qualité

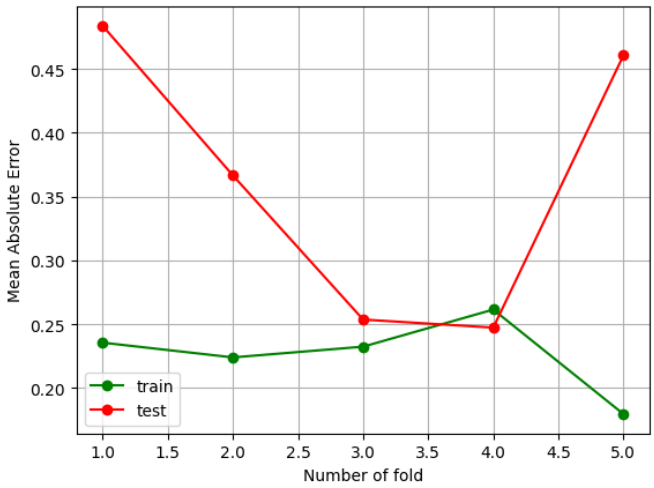


FIGURE 2 – MAE du training set et test set en fonction du fold pour les vins de basse qualité

Pour les vins de qualité basse on remarque qu'en moyenne la MAE trainingset est basse (environ 0.23) tandis que la MAE du testset est haute (de 0.25 à 0.47). Puisque la MAE d'apprentissage est petite et que la MAE de test est grande, on peut conclure que le modèle est en overfitting. Il faudrait donc reprendre le modèle gérant la qualité basse pour l'adapter ou changer d'algorithme.

5.2.2 Vins de moyenne et haute qualité

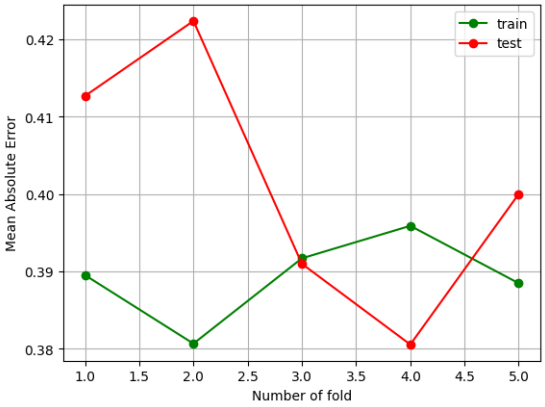


FIGURE 3 – MAE du training set et test set en fonction du fold pour les vins de qualité moyenne

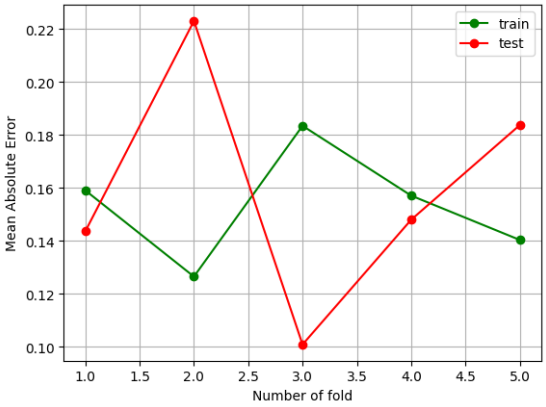


FIGURE 4 – MAE du trainingset et testset en fonction du fold pour les vins de haute qualité

Ici, pour ces deux modèles, les résultats sont meilleurs. Si on prend une moyenne sur les 5 folds on remarque que les courbes seront presque superposées. De plus, les écarts observés sont faibles, maximum 0.04 pour la qualité moyenne sur le pli 2 et 0.08 sur le pli 3 de la qualité haute. Par conséquent, ces deux modèles n'ont pas subi d'overfitting ni d'underfitting.

6 Comparaison Regression linéaire, Ridge, Lasso et ElacticNet

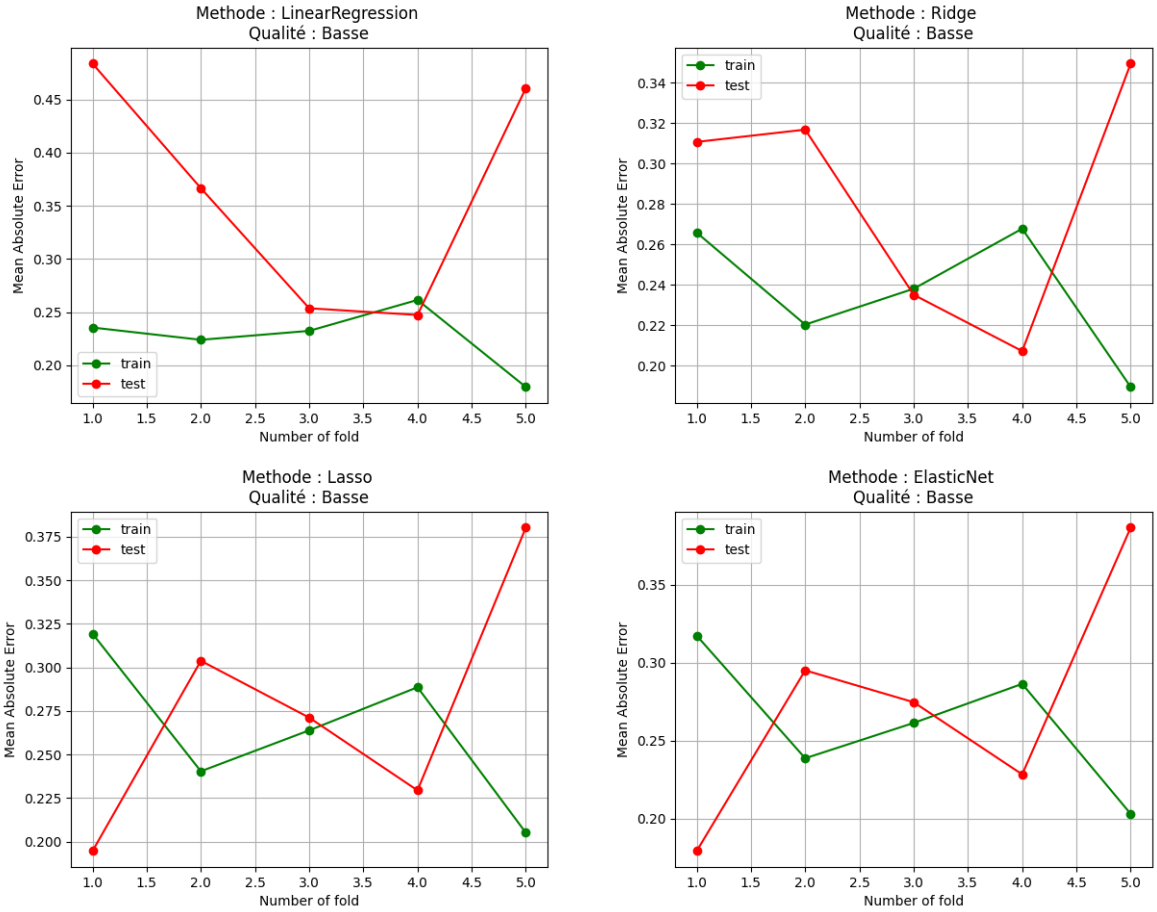
Nous reprenons ici le même dataset mais avec différents algorithmes de régression. Nous nous attarderons plus particulièrement sur le résultat RMSE et l'entrainement des données.

6.1 Pour les vins de basse qualité

6.1.1 RMSE

— Reégression linéaire = 0.454
— Ridge = 0.371
— Lasso = 0.360
— ElasticNet = 0.495

6.1.2 MAE du trainingset et testset en fonction du fold



6.1.3 Conclusion

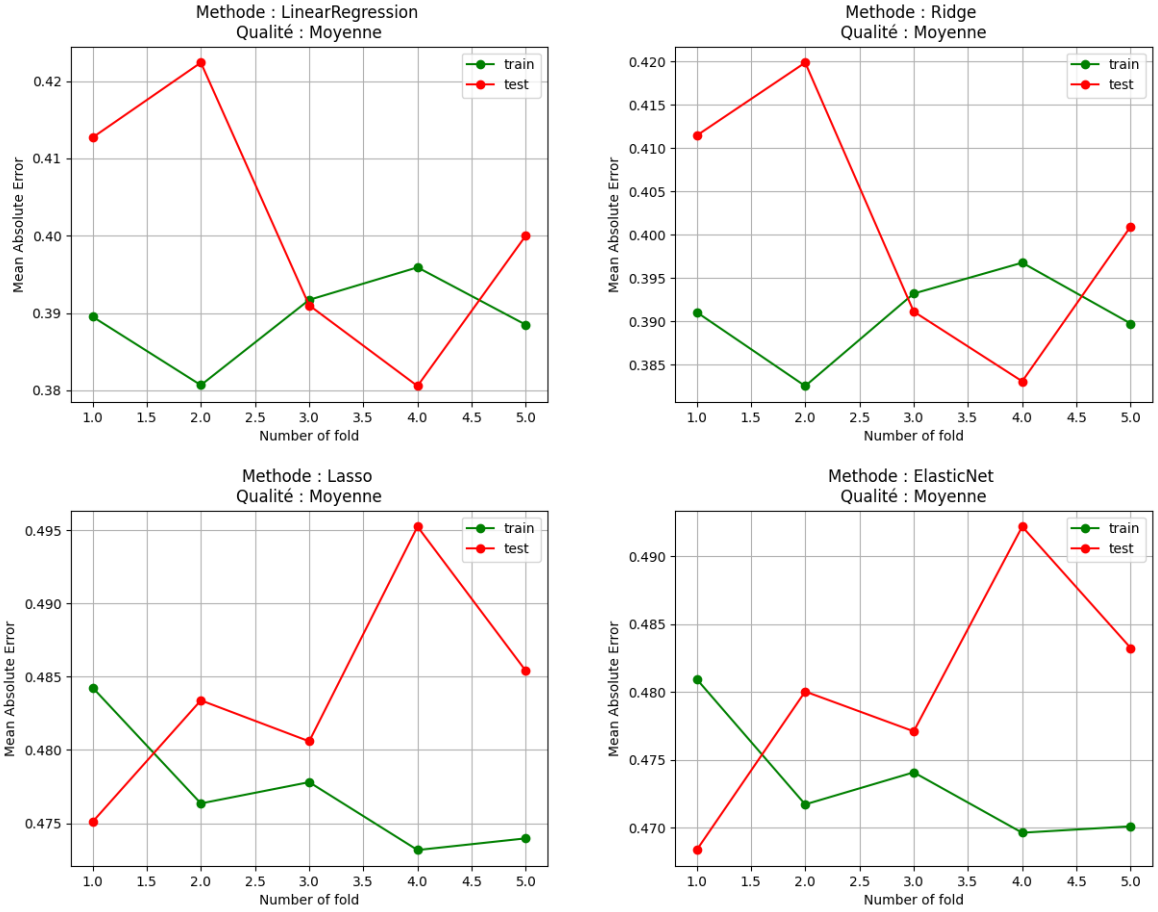
Pour les vins de basse qualité les modèles les plus satisfaisants sont Ridge ou Lasso offrant les meilleurs RMSE sans avoir d'overfitting ni d'underfitting.

6.2 Pour les vins de qualité moyenne

6.2.1 RMSE

— Régression linéaire = 0.422
— Ridge = 0.210
— Lasso = 0.484
— ElasticNet = 0.479

6.2.2 MAE du trainingset et testset en fonction du fold



6.2.3 Conclusion

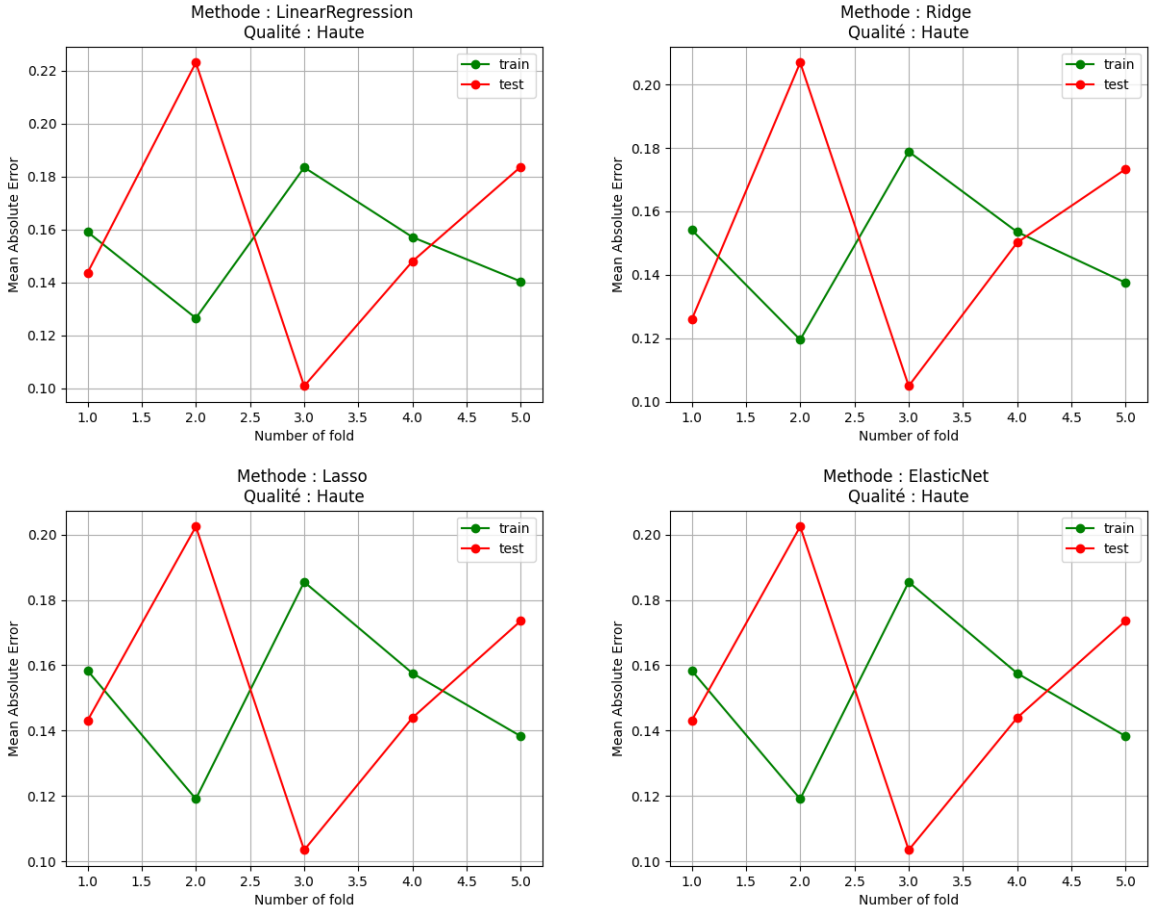
Les modèles de régression linéaire et Ridge ainsi que Lasso et ElasticNet ont des erreurs lors de l'apprentissage très semblable. Cependant, le modèle qui performe le mieux est Ridge avec un RMSE de 0.210 tandis que les autres modèles sont proches des 0.45.

6.3 Pour les vins de haute qualité

6.3.1 RMSE

— Régression linéaire = 0.213
— Ridge = 0.449
— Lasso = 0.319
— ElasticNet = 0.227

6.3.2 MAE du trainingset et testset en fonction du fold



6.3.3 Conclusion

Les erreurs lors de l'apprentissage sont presque identique à chaque fold. Les RMSE de la régression linéaire et d'ElasticNet sont les plus bas. On opterait donc pour un de ces deux algorithmes.

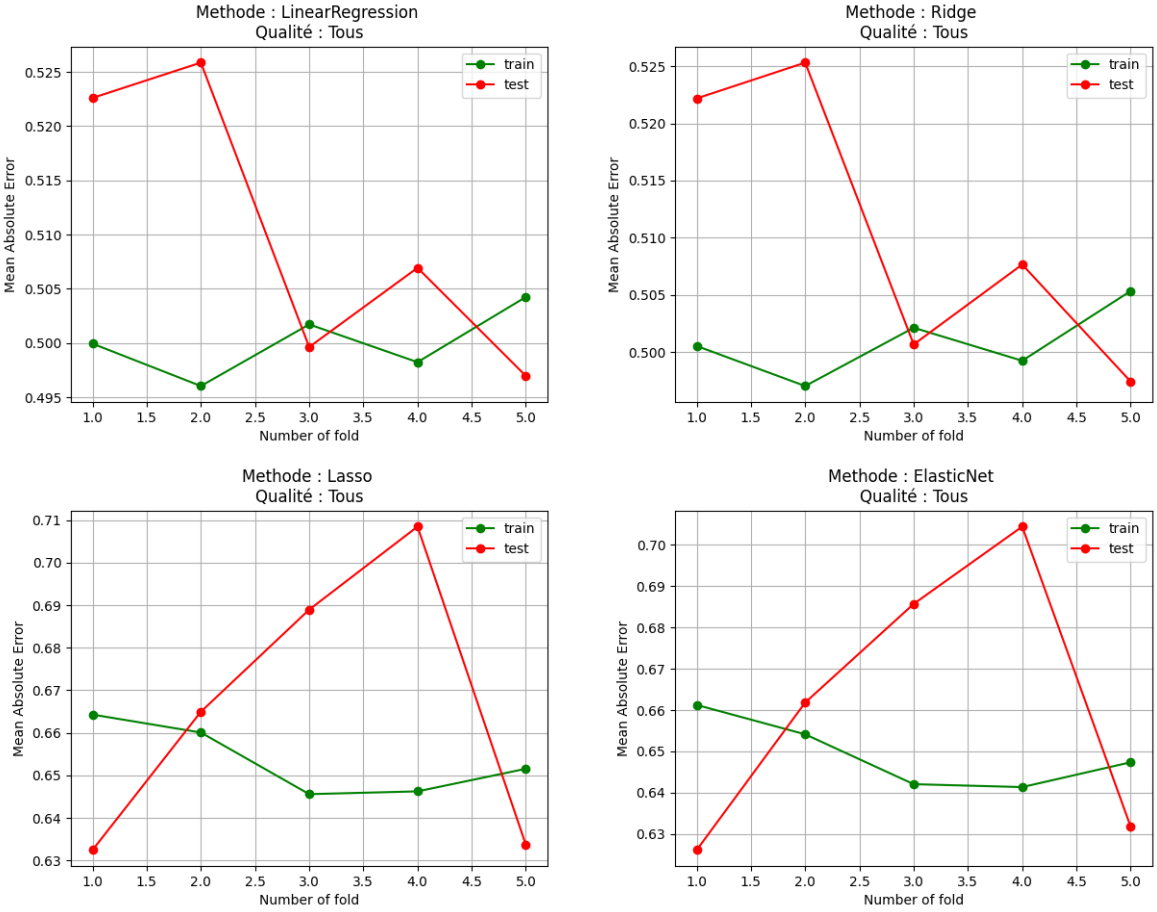
6.4 Pour tous les vins sans préclassification

Comme dit précédemment, le mieux est de faire une classification des vins puis ensuite de noter les vins via un des trois modèles de régression linéaire. Mais il est quand même intéressant de comparer l'ensemble des algorithmes sans préclassification des vins.

6.4.1 RMSE

— Régression linéaire = 0.656
— Ridge = 0.631
— Lasso = 0.808
— ElasticNet = 0.703

6.4.2 MAE du trainingset et testset en fonction du fold



6.4.3 Conclusion

Les écarts sur les MAE sont très petit, de l'ordre de 0.03 pour chaque algorithme. Cependant, les RMSEs sont assez élevés.

7 Conclusion

Pour conclure ce TP, si nous souhaitons prédire la qualité des vins il faudrait d'abord les classer suivant les états *mauvais*, *bon* ou *excellent* puis en fonction de la classe appliquer l'algorithme de régression donnant le meilleur résultat.

Comme le TP précédent, il a été très intéressant, travailler sur ce dataset était aussi agréable, car il s'adapte parfaitement à la régression et aux enjeux financiers que pourraient fournir par la suite les modèles.