Why did I choose SVC?

Our dataset has multiple features and SVC can effectively find decision boundaries in high dimensional spaces. Also, in our dataset, we do not necessarily have a linear relationship between our features and SVC is a good model for complex nonlinear relationships by using different kernels such as RBF.

Important: in our data set we have outliers and it is better to set the regularization parameter (C) in our SVC model to small values which allows larger margin by tolerating some misclassifications: making the model more robust to outliers.

What is Overfitting? And how we defined it in my implementation?
Overfitting happens when the model is very well trained on the training data but poorly on unseen test data. For this we defined the following condition and check the performance of our models with:

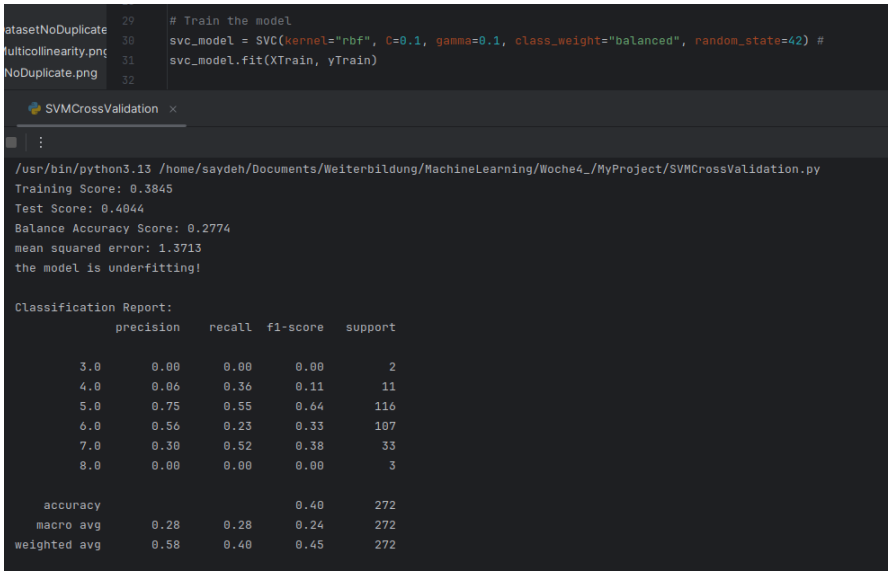
```
if tr_score - test_score > 0.1: # if the difference is more than 10%, the model is overfitting
    print(f"the model is overfitting!")

elif test_score < 0.75 and tr_score < 0.75:
    print(f"the model is underfitting!")

elif abs(tr_score-test_score) < 0.05 and test_score > 0.75 and tr_score > 0.75:
    print(f"the model is well-fitted!")

else:
    print(f"the model's performance is inconsistent!")
```

Why did we evaluate our model with balanced_accuracy_score?
Our dataset is very unbalanced, and if we only want to judge the performance of our model based on normal accuracy, this could lead to misunderstandings. Therefore, we used the balanced_accuracy_score from sklearn.metric, which calculates the average recall across all classes. This results in all classes being given the same importance. In this way, we ensure that our model is not only focused on the majority class.

class_weight parameter:
I learned about a very important argument in the SVC model called class_weight. This argument is very useful for an unbalanced dataset, which is the case in our case, as it assigns a higher importance to the classes with low numbers. In the table below, you can see the difference between using and not using this argument in terms of the performance of our model, which is reflected in the overall accuracy and the balanced accuracy.

Scenario	Without `class_weight="balanced"`	With `class_weight="balanced"`
Majority Class (Frequent Class)	Model focuses on it	Penalized for dominating predictions
Minority Class (Rare Class)	Often misclassified	More weight given to correct classification
Overall Accuracy	Might be high due to bias	Can decrease, but more fair
Balanced Accuracy Score	Likely low due to poor recall on rare classes	Likely higher since recall improves for all classes

Below I give two examples generated by my code and show how the overall accuracy, balanced accuracy and classification report are affected by this parameter:



```
# Train the model
svc_model = SVC(kernel="rbf", C=0.1, gamma=0.1, class_weight="balanced", random_state=42) #
svc_model.fit(XTrain, yTrain)
```

SVMCrossValidation

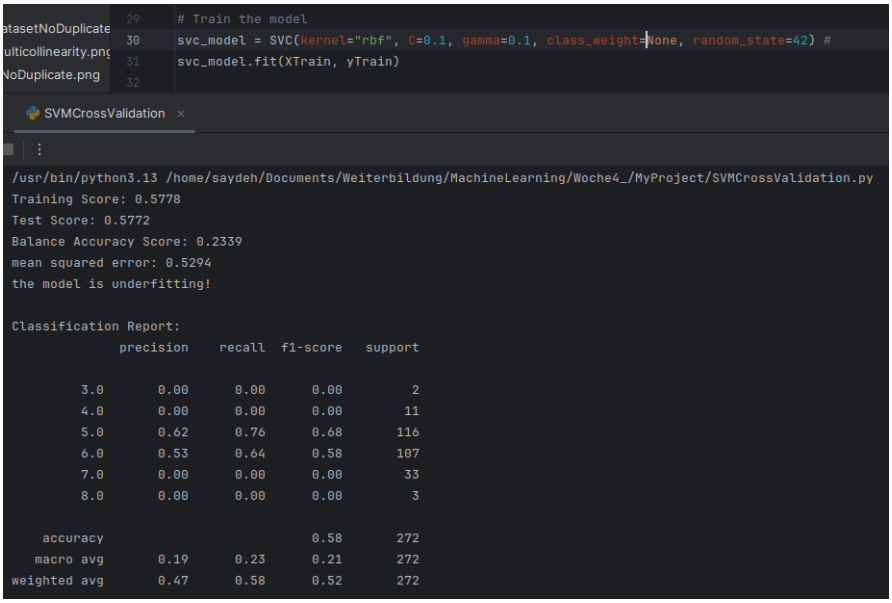
```
/usr/bin/python3.13 /home/saydeh/Documents/Weiterbildung/MachineLearning/Woche4_/MyProject/SVMCrossValidation.py
Training Score: 0.3845
Test Score: 0.4044
Balance Accuracy Score: 0.2774
mean squared error: 1.3713
the model is underfitting!

Classification Report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.06      0.36      0.11        11
         5.0       0.75      0.55      0.64       116
         6.0       0.56      0.23      0.33       107
         7.0       0.30      0.52      0.38        33
         8.0       0.00      0.00      0.00         3

    accuracy                           0.40       272
   macro avg       0.28      0.28      0.24       272
weighted avg       0.58      0.40      0.45       272
```

Based on the classification report:

I got 0% accuracy and recall for classes 3 and 8. This means that the amount of data for these two classes in this dataset is too small.

The model is strongly biased towards the wine with class 5. you see on the left that the weighted average 45% and the maximum support number belongs to class 5 with a value of 116.

If we look at the macro-aggregate F1 score (average across all classes), it shows a very low performance of around 23%.



```
# Train the model
svc_model = SVC(kernel="rbf", C=0.1, gamma=0.1, class_weight=None, random_state=42) #
svc_model.fit(XTrain, yTrain)
```

SVMCrossValidation

```
/usr/bin/python3.13 /home/saydeh/Documents/Weiterbildung/MachineLearning/Woche4_/MyProject/SVMCrossValidation.py
Training Score: 0.5778
Test Score: 0.5772
Balance Accuracy Score: 0.2339
mean squared error: 0.5294
the model is underfitting!

Classification Report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        11
         5.0       0.62      0.76      0.68       116
         6.0       0.53      0.64      0.58       107
         7.0       0.00      0.00      0.00        33
         8.0       0.00      0.00      0.00         3

    accuracy                           0.58       272
   macro avg       0.19      0.23      0.21       272
weighted avg       0.47      0.58      0.52       272
```

Based on the classification report:

I got 0% accuracy and recall not only for the classes 3 and 8 but also for 4 and 7. This is due to not using calss_weight argument.

The model is strongly biased towards the wine with class 5. you see on the left that the weighted average 45% and the maximum support number belongs to class 5 with a value of 116.

If we look at the macro-aggregate F1 score (average across all classes), it shows a very low performance of around 21%.

a very interesting Observation!

If we look at the mean squared error (MSE) output in the two figures above, we see that the MSE is 0.52 if we set class_weight=None, and MSE=1.37 if we set class_weight="balanced". As will be explained later, an MSE < 0.5 means that the model is strong and the predictions differ by at most one step. I expected class_weight="balanced" to be good for the model because it increases the importance of classes with few data points. Therefore, we can get a more robust model.

But we discussed with Elizabeth yesterday and the conclusion was that the classifiers are not the best for this dataset due to ordinal data of our target column.

How to Interpret a Confusion Matrix?
The diagonal elements (TP & TN) are the correct predictions (high values = good model performance)
The off-diagonal Elements (FP & FN)

- False Positives (FP) → When the model wrongly predicts the positive class
- False Negatives (FN) → When the model misses a positive case
- If FP is high → The model makes too many false alarms
- If FN is high → The model misses real cases, which is dangerous in medical diagnosis

When we make mistakes, we learn more!
When we started preparing our dataset we only considered the nominal and continuous data! Following table shows the definition of these to in addition to ORDINAL data:

Type	Definition	Example	Can Be Averaged?	Can Be Ranked?
Nominal	Categories **without order**	Colors (Red, Blue, Green)	❌ No	❌ No
Ordinal	Categories **with order**, but no fixed distance	Wine Quality (1-10), Satisfaction (Low, Medium, High)	✅ Sometimes	✅ Yes
Continuous	Numeric values **with fixed intervals**	Height, Temperature, Age	✅ Yes	✅ Yes

Ordinal data is a type of categorical data **with a meaningful order or ranking** but **without a fixed interval between values**. In other words, ordinal data represents categories that **can be ranked**, but the difference between them is **not necessarily uniform**.

Having only nominal and continuous data type in mind we decided to work with the classifiers. So no Ordinal Encoder or Label Encoding. And the result of all four trained classifiers was just under-fitting models where the normal accuracy and balanced accuracy scores were at the best parameter set between 40% and 60%.

Confusion matrix does not show the severity of the errors! How can we solve this?
When we use the **confusion matrix**, it only shows where the model makes errors and treats all errors equally. However, the **mean square error** weights larger errors more heavily and helps to capture how far off the mark the model is. That's why it would be very useful to look at the MSE for our classifiers. In the following table you can see this comparison:

Actual Quality	Predicted Quality	Squared Error
5	5	$(5-5)^2 = 0$
5	6	$(5-6)^2 = 1$
5	8	$(5-8)^2 = 9$

MSE gives more insight than accuracy or confusion matrix alone!
For example for the case of SVC, MSE was at the beginning 1.37 and then it reduces to 0.75 when we found a better set of the best parameters. This means that with the initial set of parameters our model predicts frequently 1-2 points away from actual values. Then for MSE = 0.75 the model is getting closer to the correct actual values. I got a lower MSE=0.73, when we applied some feature engineering to our data set. **Our aim is to reach MSE<0.5!**

important: a model with MSE<0.5 would be very strong: it means most predictions are at most one step off!

How can I improve the model?
1) by tuning further the hyperparameters C (regularization) and Gamma (kernel coefficient)
2) by more systematic feature engineering, for example with PCA and FeatureSelection
3) by trying an ordinal regression model (for example SVR)! The SVC used in this project is designed for classification tasks, not for ordinal tasks

FE: Feature Engineering

Name	CV Score	Std CV	Training Score	Test Score	Fitting	Scaler	Balanced Accuracy	F1-Score ave	Weighted avg	MSE
Dataset No FE*	0.5796	0.0	0.5915	0.5772	Under-fitting	Standard	0.2339	0.21	0.52	0.5294
Best Estimator No FE	0.5998	0.043	0.6632	0.5919	Under-fitting	Standard	0.2683	0.27	0.56	0.5257
Dataset With FE	0.5685	0.0	0.5888	0.5551	Under-fitting	Standard	0.2244	0.20	0.50	0.5625
Best Estimator With FE	0.5832	0.0448	0.6504	0.6066	Under-fitting	Standard	0.2846	0.29	0.58	0.5110

```
Dataset is: DatasetNoDuplicate
Cross Validation Score: 0.5796
Standard Deviation of the CV Scores: 0.0
Training Score: 0.5915
Test Score: 0.5772
Balance Accuracy Score: 0.2339
mean squared error: 0.5294
the model is underfitting!

Classification Report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        11
         5.0       0.62      0.76      0.68       116
         6.0       0.53      0.64      0.58       107
         7.0       0.00      0.00      0.00        33
         8.0       0.00      0.00      0.00         3

    accuracy                           0.58       272
   macro avg       0.19      0.23      0.21       272
weighted avg       0.47      0.58      0.52       272
```

```
Dataset is: Multicollinearity
Cross Validation Score: 0.5685
Standard Deviation of the CV Scores: 0.0
Training Score: 0.5888
Test Score: 0.5551
Balance Accuracy Score: 0.2244
mean squared error: 0.5625
the model is underfitting!

Classification Report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        11
         5.0       0.60      0.77      0.67       116
         6.0       0.50      0.58      0.54       107
         7.0       0.00      0.00      0.00        33
         8.0       0.00      0.00      0.00         3

    accuracy                           0.56       272
   macro avg       0.18      0.22      0.20       272
weighted avg       0.45      0.56      0.50       272
```

```
/usr/bin/python3.13 /home/saydeh/Documents/Weiterbildung/MachineLearni
the Dataset is: DatasetNoDuplicate
Fitting 5 folds for each of 432 candidates, totalling 2160 fits
the best mean score (None): 0.5998393438464465
the best parameters: {'classifier__C': 1, 'classifier__class_weight': 
the best estimator: Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier', SVC(C=1, degree=2, random_state=42))])
the best score has index: 366
the list of std scores: 0.043551444483577
/home/saydeh/.local/lib/python3.13/site-packages/sklearn/metrics/_class
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
/home/saydeh/.local/lib/python3.13/site-packages/sklearn/metrics/_class
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
/home/saydeh/.local/lib/python3.13/site-packages/sklearn/metrics/_class
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)
training accuracy: 0.6632934682612696
test accuracy: 0.5919117647058824
balanced accuracy: 0.2683286295113559
mean squared error: 0.5257

Classification Report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        11
         5.0       0.66      0.74      0.70       116
         6.0       0.54      0.63      0.58       107
         7.0       0.44      0.24      0.31        33
         8.0       0.00      0.00      0.00         3

    accuracy                           0.59       272
   macro avg       0.27      0.27      0.27       272
weighted avg       0.55      0.59      0.56       272

the model is underfitting!
```

```
/usr/bin/python3.13 /home/saydeh/Documents/Weiterbildung/MachineLearni
the Dataset is: Multicollinearity
Fitting 5 folds for each of 432 candidates, totalling 2160 fits
the best mean score (None): 0.5832917600304401
the best parameters: {'classifier__C': 1, 'classifier__class_weight': 
the best estimator: Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier', SVC(C=1, degree=2, random_state=42))])
the best score has index: 366
the list of std scores: 0.04488402779869463
training accuracy: 0.6504139834406624
test accuracy: 0.6066176470588235
balanced accuracy: 0.2846752246929495
mean squared error: 0.5110

Classification Report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        11
         5.0       0.64      0.77      0.70       116
         6.0       0.56      0.61      0.58       107
         7.0       0.61      0.33      0.43        33
         8.0       0.00      0.00      0.00         3

    accuracy                           0.61       272
   macro avg       0.30      0.28      0.29       272
weighted avg       0.57      0.61      0.58       272

the model is underfitting!
```

GridSearchCV:

Original data with name of "DatasetNoDuplicate.csv":
the best parameters: {'classifier__C': 1, 'classifier__class_weight': None, 'classifier__degree': 2, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf', 'scaler': StandardScaler()}
the best estimator: Pipeline(steps=[('scaler', StandardScaler()),
('classifier', SVC(C=1, degree=2, random_state=42))])

FeaturedEngineered data with name of "multicollinearity.csv":
the best parameters of SVC: {'classifier__C': 1, 'classifier__class_weight': None, 'classifier__degree': 2, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf', 'scaler': StandardScaler()}
the best estimator: Pipeline(steps=[('scaler', StandardScaler()),
('classifier', SVC(C=1, degree=2, random_state=42))])

I got the best parameters and the best estimator for both data set the same.

Votingclassifier:
Since I did not got the best parameters from some models I used them with default values and the scalers with which have better performances. Following shows a screenshot of part of my code:

```
# Define Pipelines for models with appropriate scalers
dt_model = DecisionTreeClassifier(random_state=42)  # No scaling needed

svc_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # this scaler I got from the best parameter set
    ('svc', SVC(kernel="rbf", C=1, gamma="scale", degree=2, random_state=42, probability=True))
])

nb_pipeline = Pipeline([
    ('scaler', MinMaxScaler()),  # GaussianNB often benefits from MinMaxScaler
    ('nb', GaussianNB())
])

logreg_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Logistic Regression benefits from StandardScaler
    ('logreg', LogisticRegression(C=1, max_iter=10000, penalty="l2", solver="lbfgs", random_state=42))
])

# Create a Voting Classifier with Hard Voting
voting_clf = VotingClassifier(
    estimators=[
        ('dt', dt_model),
        ('svc', svc_pipeline),
        ('nb', nb_pipeline),
        ('logreg', logreg_pipeline)
    ],
    voting='soft'  # 'soft' for probability-based voting and "hard" os for the majority-based voting
)
```

	Training Score	Test Score	Balance Accuracy Score	MSE	Fitting
Voting(soft)-Dataset-NoDuplicate	0.8473	0.6103	0.3008	0.5515	Over-Fitting
Voting(hard)-Dataset-NoDuplicate	0.6789	0.5882	0.2835	0.5404	Under-Fitting
Voting(soft)-Dataset-FE	0.8270	0.5809	0.2846	0.5699	Over-Fitting
Voting(hard)-Dataset-FE	0.6679	0.6176	0.3026	0.4890	Under-Fitting

VotingClassifier with voting "soft" has the best performance since the MSE is less than 0.5.

I should say that there is an error in my voting classifier code! When I look at the last 10 fields of the predicted data. I see that all four models predict also 1 and 2. however, there are no classes and we only have between 3 and 8. the problem is because in votingclassifier we have the LabelEncoder adjustment which gives the result of zero!

In SVC model I got the MSE between 0.5 and 0.58. This means that my model is doing okay but still making small mistakes.

What I would like to emphasize again:

We got the first step wrong in selecting the models. The type of our target columns is ordinal and we should have aimed for regressors and not classifiers. Then we should have changed the ordinal data in the target column using OrdinalEncoder or LabelEncoding.

Lessons Learned:

- It has been proven that the data preparation phase requires a lot of time and energy.
- I have learned that investing more time in data visualization and understanding the data set is of great benefit!
- I learned a lot about ordinal data (although we had talked about it during the course, but I think I hadn't learned it properly due to language limitations)
- With a simple Google search, you can find out that the criteria for a good or bad wine are, for example, the type of grape or the place of production. This information is not included in our data set. That alone tells you that our dataset is not good. It contains a lot of chemistry-related characteristics.
- It was very cool to learn how to stratify each folds (StratifiedKFold) during cross validation (for example in GridSearchCV)! This increases the robustness of our model, I believe.
- I would use again Pipeline and GridSearchCV. They are very cool and make the life easy!

- Using the pipeline is very compelling. One of the main advantages is the avoidance of information loss, which can easily happen if we don't use the pipeline and accidentally apply transform_fit to our test data.

- I learned about a very cool argument in SVM model named, “class_weight” which is very important for imbalanced dataset. It intensify the importance of the classes with little data. I just tuned my model between “None” and “balanced” values. Based on what I discussed with Elizabeth you can also tune more specifically a particular class. For this you need to define the input as a dictionary format and so on. If I had more time I would have played more with this parameter.

- Next time I would also add the AttributAdder to my pipeline in order to engineer the features of the data set (here we did it “manually” in a separate Python file and saved it to use later)

- Next time I would try to use sklearn.FeatureSelection to engineer the features more systematically

- This time I learned about the learning curve, which shows how the training and validation results evolve as the training set gets bigger! That was a new analysis for me, which, if I'm not mistaken, we didn't discuss in our course. I have implemented the code for this analysis and tried to learn how to interpret it, but I still have a lot of questions that need to be answered. So next time I will try to dive more into the subject matter

- the tasks such as programming and troubleshooting were easy, but I found the interpretation of the results very challenging

- I really enjoyed helping others with programming and troubleshooting!

- What I noticed more vividly that I generally like to solve my problems very quickly. I may have found the solution for each of these problems, but before I internalize all these solution and newly learned concepts, I quickly move on to the next problem. I should give myself more time!

- Next time I would give more thoughts to the dataset, and spend more time on its analysis, visualization and allow myself to raise more questions during this phase!