Supervised Machine Learning: Classification

Instructions:

One of the main objectives of this course is to help you gain hands-on experience in communicating insightful and impactful findings to stakeholders. In this project you will use the tools and techniques you learned throughout this course to train a few classification models on a data set that you feel passionate about, select the regression that best suits your needs, and communicate insights you found from your modeling exercise.

After going through some guided steps, you will have insights that either explain or predict your outcome variable. As a main deliverable, you will submit a report that helps you focus on highlighting your analytical skills and thought process.

Setup instructions:

Before you begin, you will need to choose a data set that you feel passionate about. You can brainstorm with your peers about great public data sets using the discussion board in this module.

Please also make sure that you can print your report into a pdf file.

How to submit:

The format of your work must adhere to the following guidelines. The report should be submitted as a pdf. Optionally, you can include a python notebook with code.

Make sure to include mainly insights and findings on your report. There is no need to include code, unless you want to.

Project

Optional: Find your own data set

As a suggested first step, spend some time finding a data set that you are really passionate about. This can be a data set similar to the data you have available at work or data you have always wanted to analyze. For some people this will be sports data sets, while some other folks prefer to focus on data from a datathon or data for good.

Optional: Participate in a discussion board

As an optional step, go into a discussion board and brainstorm with your peers great data sets to analyze. If you prefer to skip this step, feel free to use the Ames housing data set or the Churn phone data set that we used throughout the course.

Required

Once you have selected a data set, you will produce the deliverables listed below and submit them to one of your peers for review. Treat this exercise as an opportunity to produce analysis that are ready to highlight your analytical skills for a senior audience, for example, the Chief Data Officer, or the Head of Analytics at your company.

Sections required in your report:

1. Main objective of the analysis that specifies whether your model will be focused on prediction or interpretation and the benefits that your analysis provides to the business or stakeholders of this data.
2. Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.
3. Brief summary of data exploration and actions taken for data cleaning and feature engineering.
4. Summary of training at least three different classifier models, preferably of different nature in explainability and predictability. For example, you can start with a simple logistic regression as a baseline, adding other models or ensemble models. Preferably, all your models use the same training and test splits, or the same cross-validation method.
5. A paragraph explaining which of your classifier models you recommend as a final model that best fits your needs in terms of accuracy and explainability.
6. Summary Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your classifier model.
7. Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model after adding specific data features that may help you achieve a better explanation or a better prediction.

Objective of the analysis

Describe the main objective of the analysis that specifies whether your model will be focused on prediction or interpretation and the benefits that your analysis provides to the business or stakeholders of this data.

Main Objective

The primary objectives of this machine learning classification project centres around building a model that accurately predicts categorical outcomes. This involves meticulous data preprocessing, including cleaning, transformation, and feature engineering, to ensure the data is suitable for model training. Subsequently, the project focuses on selecting and training appropriate classification algorithms, such as logistic regression, decision trees, support vector machines and others optimising their performance through hyper-parameter tuning and cross-validation. Finally, the project aims to rigorously evaluate each model's predictive capabilities using metrics like accuracy, precision, recall, and F1-score, ensuring it meets the desired performance criteria and can be deployed for practical applications.

In the following section we will introduce the data set and the classification subject.

Description of the data

Provide a brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.

Introduction

In this project we will be using a wine quality data set. This data set contains various chemical properties of wine, such as acidity, sugar, pH, and alcohol. It also contains a quality metric (3-9, with highest being better) and a colour (red or white). We will be attempting to predict the colour of the wine by using various classification models.

This Vinho Verde dataset describes a unique wine from the Minho (north-west) region of Portugal. Medium in alcohol, is it particularly appreciated due to its freshness (specially in the summer). This wine accounts for 15% of the total Portuguese production, and around 10% is exported, mostly white wine.

Once viewed as a luxury good, nowadays wine is increasingly enjoyed by a wider range of consumers. Portugal is a top ten wine exporting country with 3.17% of the market share in 2005. Exports of its vinho verde wine (from the northwest region) have increased by 36% from 1997 to 2007. To support its growth, the wine industry is investing in new technologies for both wine making and selling processes. Wine certification and quality assessment are key elements within this context. Certification prevents the illegal adulteration of wines (to safeguard human health) and assures quality for the wine market. Quality evaluation is often part of the certification process and can be used to improve wine making (by identifying the most influential factors) and to stratify wines such as premium brands (useful for setting prices).

Wine certification is generally assessed by physicochemical and sensory tests. Physicochemical laboratory tests routinely used to characterize wine include determination of density, alcohol or pH values, while sensory tests rely mainly on human experts. It should be stressed that taste is the least understood of the human senses, thus wine classification is a difficult task. Moreover, the relationships between the physicochemical and sensory analysis are complex and still not fully understood.

Advances in information technologies have made it possible to collect, store and process massive, often highly complex datasets. All this data hold valuable information such as trends and patterns, which can be used to improve decision making and optimize chances of success.

Source

The Data Source

The dataset is available here or here.

Description of the Data Attributes

Variable			

Name	Role	Type	Description / Comment
fixed acidity	Feature	Continuous	Fixed acidity in wine is primarily due to the presence of stable acids such as tartaric, malic, citric, and succinic. These acids contribute to the overall taste and balance of the wine.
volatile acidity	Feature	Continuous	Often referred to as VA, volatile acidity is a measure of a wine's gaseous acids. The amount of VA in wine is often considered an indicator of spoilage.
citric acid	Feature	Continuous	Citric acid is often added to wines to increase acidity, complement a specific flavour or prevent ferric hazes.
residual sugar	Feature	Continuous	Residual Sugar (or RS) is from natural grape sugars leftover in a wine after the alcoholic fermentation finishes.
chlorides	Feature	Continuous	Chlorides in wine are salts of mineral acids that can affect the taste and quality of the wine.
free sulfur dioxide	Feature	Continuous	Free sulfur dioxide (SO2) is a preservative for wine. It has both antioxidant and antimicrobial properties, making it an effective preservative
total sulfur dioxide	Feature	Continuous	Total Sulfur Dioxide (TSO2) in wine is the portion of SO2 that is free in the wine plus the portion that is bound to other chemicals in the wine such as aldehydes, pigments, or sugars.
density	Feature	Continuous	Wine density is determined by the amount of sugar, alcohol, and other solutes present in the wine. Generally, the higher the sugar and alcohol content, the higher the density of the wine.
pH	Feature	Continuous	The acidity of the wine. The pH of wine typically ranges from about 2.9 to 4.0. White wine usually has a pH level of 3.0 to 3.4, while red wine is between 3.3 to 3.6
sulphates	Feature	Continuous	Sulfates, or sulfur dioxide (SO2), are naturally occurring compounds that have been used in winemaking for centuries. They are a type of preservative that can help prevent oxidation and microbial spoilage in wine.
alcohol	Feature	Continuous	Wine alcohol content varies depending on the type of wine and the amount poured. A standard serving of wine is 5 ounces and generally contains between 11-13% alcohol by volume
quality	Feature	Integer	score between 0 and 10
colour	Target	Categorical	red or white

Summary of data exploration

Brief summary of data exploration and actions taken for data cleaning and feature engineering.

Data Exploration

The following analysis was completed to verify the data and to prepare the data for modelling:

1. Label Encode the Target
2. Examine the data types
3. Check for missing data
4. Pair Plot and Pearson's Correlation
5. Data Ranges
6. Class Balance
7. Stratified Shuffle Split

Label Encode the Target

The target variable was encode as `1` for red wine and `0` as white wine.

Data Types

By examining the data types we can see that all types are correct

Attribute	Data Type
fixed acidity	float64
volatile acidity	float64
citric acid	float64
residual sugar	float64
chlorides	float64
free sulfur dioxide	float64
total sulfur dioxide	float64
density	float64
pH	float64
sulphates	float64
alcohol	float64
quality	int64
color	object

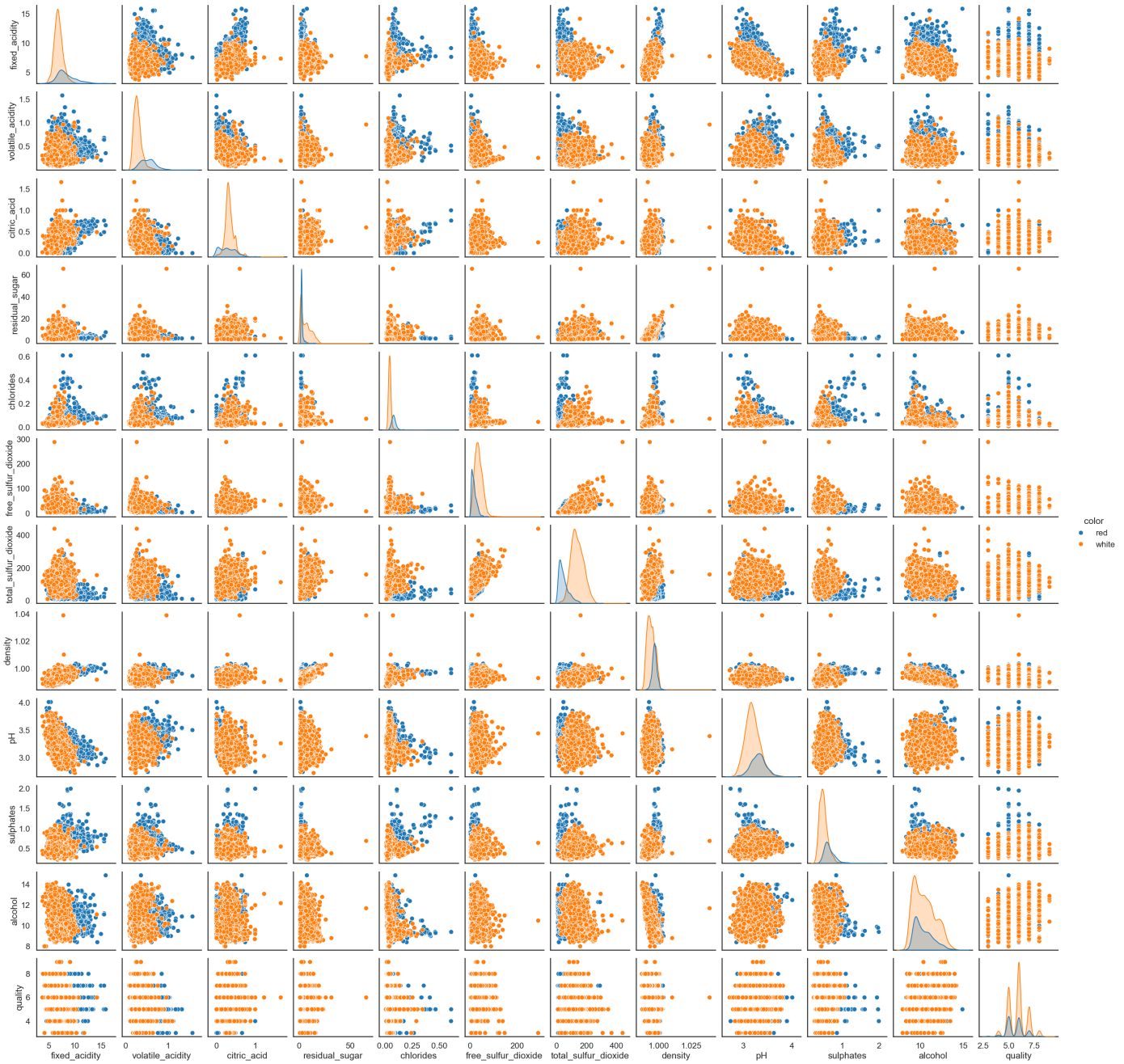
Missing Data

There are no null values so missing data is not a consideration.

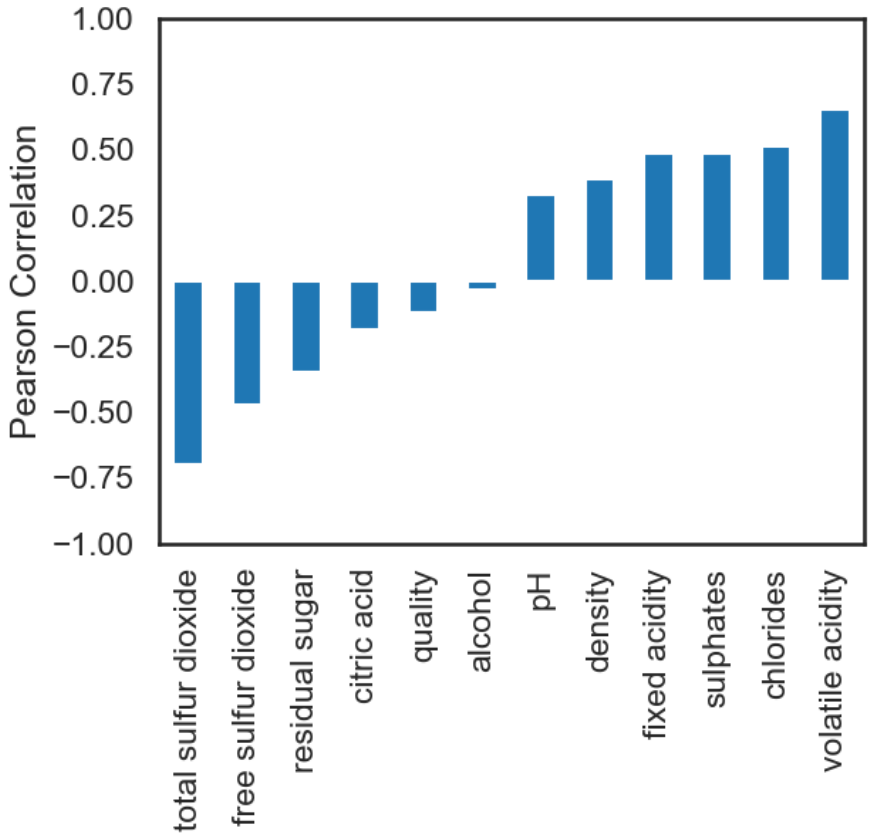
Attribute	Null Values
fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0
free sulfur dioxide	0
total sulfur dioxide	0
density	0
pH	0
sulphates	0
alcohol	0
quality	0
color	0

Pair Plot and Pearson's Correlation

The pair plot shows that there are a lot of promising attributes that are predictive.



The Pearson Correlation values show that `total sulphur dioxide` is the most negatively correlated attribute while `volatile acidity` is the most positively correlated.



Data Ranges

There is a lot of variance in the range of the data so the data will be standardised using a Min Max Scaler.

Attribute	count	mean	std	min	25%	50%	75%	max
fixed acidity	6497	7.22	1.3	3.8	6.4	7	7.7	15.9
volatile acidity	6497	0.34	0.16	0.08	0.23	0.29	0.4	1.58
citric acid	6497	0.32	0.15	0	0.25	0.31	0.39	1.66
residual sugar	6497	5.44	4.76	0.6	1.8	3	8.1	65.8
chlorides	6497	0.06	0.04	0.01	0.04	0.05	0.06	0.61
free sulfur dioxide	6497	30.53	17.75	1	17	29	41	289
total sulfur dioxide	6497	115.74	56.52	6	77	118	156	440
density	6497	0.99	0	0.99	0.99	0.99	1	1.04
pH	6497	3.22	0.16	2.72	3.11	3.21	3.32	4.01
sulphates	6497	0.53	0.15	0.22	0.43	0.51	0.6	2
alcohol	6497	10.49	1.19	8	9.5	10.3	11.3	14.9
quality	6497	5.82	0.87	3	5	6	6	9

After scaling.

```
def minmax_scale_columns(df: pd.DataFrame, columns: list = None) -> pd.DataFrame:
    df_copy = df.copy()  # Avoid modifying the original DataFrame.

    if columns is None:
```

```
        numeric_cols = df_copy.select_dtypes(include=["number"]).columns.tolist()
        columns = numeric_cols

    scaler = MinMaxScaler()
    try:
        df_copy[columns] = scaler.fit_transform(df_copy[columns])
    except KeyError as e:
        print(f"Error: One or more specified columns not found. {e}")
    except ValueError as e:
        print(f"Error during scaling. Check if columns contain numeric data: {e}")
    return df_copy


df_norm = minmax_scale_columns(df, df.columns[:-1]).reset_index(drop=True)
```

	count	mean	std	min	25%	50%	75%	max
fixed acidity	6497	0.28	0.11	0	0.21	0.26	0.32	1
volatile acidity	6497	0.17	0.11	0	0.1	0.14	0.21	1
citric acid	6497	0.19	0.09	0	0.15	0.19	0.23	1
residual sugar	6497	0.07	0.07	0	0.02	0.04	0.12	1
chlorides	6497	0.08	0.06	0	0.05	0.06	0.09	1
free sulfur dioxide	6497	0.1	0.06	0	0.06	0.1	0.14	1
total sulfur dioxide	6497	0.25	0.13	0	0.16	0.26	0.35	1
density	6497	0.15	0.06	0	0.1	0.15	0.19	1
pH	6497	0.39	0.12	0	0.3	0.38	0.47	1
sulphates	6497	0.17	0.08	0	0.12	0.16	0.21	1
alcohol	6497	0.36	0.17	0	0.22	0.33	0.48	1
quality	6497	0.47	0.15	0	0.33	0.5	0.5	1

Class Balance

If we look at the `color` target we can see that it has a class unbalance.

color	proportion
0	0.753886
1	0.246114

Later we will address this class imbalance using different sampling techniques.

Stratified Shuffle Split

Finally the `X` and `y` data sets were split into training and test data using stratified shuffle split.

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_state=42)
for train_index, test_index in split.split(df_norm, df_norm["color"]):
    df_train = df_norm.loc[train_index]
    df_test = df_norm.loc[test_index]

# Create the Train and the Test data
X_train = df_train[df_train.columns[:-1]]
X_test = df_test[df_test.columns[:-1]]

y_train = df_train["color"]
y_test = df_test["color"]
```

Dataset	color	proportion
Train	0	0.753904
Train	1	0.246096
Test	0	0.753846
Test	1	0.246154

We can see that the wine colour sample distribution has been maintained.

Model Training

Summary of training at least three different classifier models, preferably of different nature in explainability and predictability. For example, you can start with a simple logistic regression as a baseline, adding other models or ensemble models. Preferably, all your models use the same training and test splits, or the same cross-validation method.

The following model types and modelling approaches are evaluated:

1. Logistic Regression
2. Support Vector Machines
3. K Nearest Neighbours
4. Decision Trees

Each model type or approach following the same simple approach.

1. Use the training and test data generated during data exploration
    1. No additional processing of the data is performed unless required e.g. balancing
2. Fit the model to the data
3. Use the generated model to get the test data predictions
4. Calculate and store the performance of the model

Logistic Regression

We will first train three Logistic Regression models:

1. Standard logistic regression
2. L1 regularized logistic regression
3. L2 regularized logistic regression

Logistic regression is a statistical model typically used for binary classification. Unlike linear regression, which predicts continuous values, logistic regression predicts the probability of a binary outcome (e.g., yes/no, true/false, 0/1).

Core Concepts:

- Probability Prediction:
    - Logistic regression uses the sigmoid function (or logistic function) to transform a linear combination of input features into a probability value between 0 and 1.
    - This probability represents the likelihood that an instance belongs to a particular class.
- Classification:
    - A threshold is typically set (e.g., 0.5) to convert the probability into a discrete class label. If the probability is above the threshold, the instance is classified into one class; otherwise, it's classified into the other.

- Log-Odds:
    - Internally, logistic regression models the relationship between the input features and the log-odds of the outcome. This allows for a linear relationship between the predictors and the log-odds of the event.

Advantages:

- Interpretability:
    - The coefficients of the logistic regression model can be interpreted to understand the impact of each feature on the probability of the outcome.
- Efficiency:
    - Logistic regression is computationally efficient, making it suitable for large datasets.
- Probability Output:
    - It provides probability estimates, which can be useful for decision-making.
- Widely Applicable:
    - It's used in various fields, including healthcare, finance, marketing, and more.

Disadvantages:

- Linearity Assumption:
    - Logistic regression assumes a linear relationship between the input features and the log-odds of the outcome. This assumption may not hold true for complex datasets.
- Sensitivity to Multicollinearity:
    - High correlation between input features (multicollinearity) can affect the stability and reliability of the model.
- Limited Complexity:
    - It may not perform well on datasets with complex, non-linear relationships.
- Outliers:
    - Logistic regression can be sensitive to outliers.
- Requires relatively large datasets:
    - For reliable results, Logistic regression models often require larger datasets.

In essence, logistic regression is a valuable tool for binary classification tasks, particularly when interpretability and efficiency are important. However, it's essential to be aware of its limitations and to choose the appropriate model based on the characteristics of the data.

Regularisation

When applied to logistic regression, L1 and L2 regularization are techniques used to prevent overfitting, which is when a model performs very well on training data but poorly on unseen data.1 Here's a breakdown:

The Core Idea: Regularization

- Regularization works by adding a penalty term to the cost function that the logistic regression model tries to minimize.
- This penalty discourages the model from assigning excessively large coefficients to the input features.2
- By constraining the size of the coefficients, regularization helps to simplify the model and improve its generalization ability.3

L1 Regularization (Lasso Regression)

- L1 regularization adds the absolute value of the coefficients to the penalty term.4
- A key characteristic of L1 regularization is that it can drive some coefficients to exactly zero.5
-
- This effectively performs feature selection, as features with zero coefficients are excluded from the model.67
- Therefore, L1 regularization is useful when you suspect that many of your features are irrelevant.8

L2 Regularization (Ridge Regression)

- L2 regularization adds the squared value of the coefficients to the penalty term.9
- L2 regularization shrinks the coefficients towards zero, but it rarely sets them exactly to zero.10
- Instead, it distributes the impact of the features more evenly.
- L2 regularization is effective at reducing the impact of multicollinearity (high correlation between features).

Key Differences and Considerations:

- Feature Selection:
  - L1 regularization performs implicit feature selection by driving some coefficients to zero.11
  - L2 regularization does not perform feature selection; it shrinks coefficients but rarely eliminates them.12
- Sparsity:
  - L1 regularization produces sparse models, meaning that they have few non-zero coefficients.13
  - L2 regularization produces non-sparse models.14
- Multicollinearity:
  - L2 regularization is generally better at handling multicollinearity.15
- When to Use Which:
  - Use L1 regularization when you want to perform feature selection or when you suspect that many features are irrelevant.
  - Use L2 regularization when you want to reduce the impact of multicollinearity or when you want to prevent overfitting without performing feature selection.
  - Often times a combination of both is used, in a method called Elastic Net.16

In summary, L1 and L2 regularization are valuable tools for improving the performance of logistic regression models by preventing overfitting and enhancing their generalization capabilities.17

Model Performance

When evaluating the performance of a logistic regression model, the "best" metrics depend on the specific goals of your project and the characteristics of your data (e.g., class imbalance). Here's a breakdown of commonly used metrics and when they're most appropriate:

Fundamental Metrics:

- Accuracy:
  - Definition: The proportion of correctly classified instances.
  - Formula: (True Positives + True Negatives) / Total Instances
  - Use Case: Suitable when classes are balanced and you care about overall correctness.
  - Caveat: Can be misleading in imbalanced datasets.
- Precision:
  - Definition: The proportion of correctly predicted positive instances out of all instances predicted as positive.
  - Formula: True Positives / (True Positives + False Positives)
  - Use Case: Important when minimizing false positives is crucial (e.g., spam detection, medical diagnosis).
- Recall (Sensitivity):
  - Definition: The proportion of correctly predicted positive instances out of all actual positive instances.
  - Formula: True Positives / (True Positives + False Negatives)
  - Use Case: Important when minimizing false negatives is crucial (e.g., detecting diseases).
- F1-Score:
  - Definition: The harmonic mean of precision and recall.
  - Formula: 2 * (Precision * Recall) / (Precision + Recall)
  - Use Case: Provides a balanced measure when both precision and recall are important, especially in imbalanced datasets.6

Metrics for Imbalanced Datasets:

- Area Under the ROC Curve (AUC-ROC):
  - Definition: Measures the model's ability to distinguish between positive and negative classes across various thresholds.
  - Use Case: Excellent for imbalanced datasets and when you want a threshold-independent measure of performance.
  - Interpretation: Higher AUC-ROC values indicate better performance.
- Area Under the Precision-Recall Curve (AUC-PR):
  - Definition: Measures the trade-off between precision and recall across various thresholds.
  - Use Case: Particularly useful for highly imbalanced datasets where the positive class is rare.

    - Interpretation: Higher AUC-PR values indicate better performance.
- Balanced Accuracy:
    - Definition: The average of recall obtained on each class.
    - Use Case: When dealing with imbalanced datasets.
- Confusion Matrix:
    - Definition: A table that shows the number of true positives, true negatives, false positives, and false negatives.
    - Use Case: Provides a detailed view of the model's performance and can help identify specific types of errors.

Other Important Considerations:

- Log Loss (Cross-Entropy **Loss):**
    - Definition: Measures the performance of a classification model whose output is a probability value between 0 and 1.
    - Use Case:11 When you want to evaluate the models probability predictions.
- Threshold Selection:
    - The choice of threshold (usually 0.5) can significantly impact precision and recall.12 Consider using metrics that are less sensitive to threshold selection (e.g., AUC-ROC, AUC-PR).

In summary:

- For balanced datasets and general correctness: Accuracy, F1-score.
- For imbalanced datasets and threshold-independent performance: AUC-ROC, AUC-PR, Balanced Accuracy.
- For minimizing false positives: Precision.
- For minimizing false negatives: Recall.
- For a detailed view of errors: Confusion Matrix.
- For evaluating probability predictions: Log Loss.

It is important to always choose metrics that align with your project's objectives and the characteristics of your data.

Model Creation and Performance

First the three models were fit on the training data:

```
# Standard logistic regression
lr = LogisticRegression(solver="liblinear").fit(X_train, y_train)

# L1 regularized logistic regression
lr_l1 = LogisticRegressionCV(Cs=10, cv=4, penalty="l1", solver="liblinear").fit(
    X_train, y_train
)

# L2 regularized logistic regression
lr_l2 = LogisticRegressionCV(Cs=10, cv=4, penalty="l2", solver="liblinear").fit(
    X_train, y_train
)
```

Next each models was use to predict the target field, in this case wine colour:

```
# Predict the class for each model
y_pred = list()

coeff_labels = ["lr", "l1", "l2"]
coeff_models = [lr, lr_l1, lr_l2]

for lab, mod in zip(coeff_labels, coeff_models):
    y_pred.append(pd.Series(mod.predict(X_test), name=lab))

y_pred = pd.concat(y_pred, axis=1)
```

Finally the performance of each of the models was calculated:

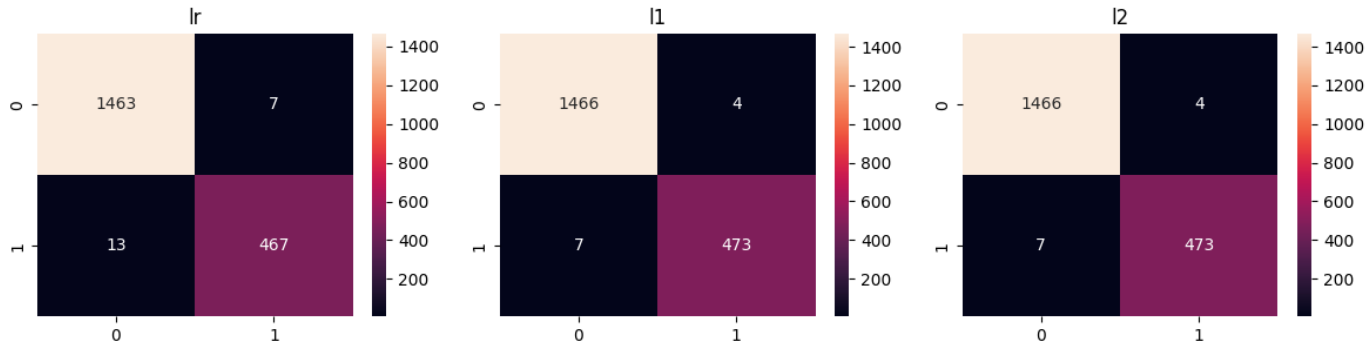
```
metrics = list()
cm = dict()

for lab in coeff_labels:

    # Preciision, recall, f-score from the multi-class support function
    precision, recall, fscore, _ = score(y_test, y_pred[lab], average="weighted")

    # The usual way to calculate accuracy
    accuracy = accuracy_score(y_test, y_pred[lab])

    # ROC-AUC scores can be calculated by binarizing the data
    auc = roc_auc_score(
        label_binarize(y_test, classes=[0, 1]),
        label_binarize(y_pred[lab], classes=[0, 1]),
        average="weighted",
    )

    # Last, the confusion matrix
    cm[lab] = confusion_matrix(y_test, y_pred[lab])

    metrics.append(
```

```
        pd.Series(
            {
                "precision": precision,
                "recall": recall,
                "accuracy": accuracy,
                "fscore": fscore,
                "auc": auc,
            },
            name=lab,
        )
    )

metrics = pd.concat(metrics, axis=1)
```

If we look at the performance of each of the models we can see that logistic regression has performed excellently:

model	precision	recall	accuracy	fscore	auc
lr	0.9897	0.9897	0.9897	0.9897	0.9841
l1	0.9944	0.9944	0.9944	0.9944	0.9913
l2	0.9944	0.9944	0.9944	0.9944	0.9913

Viewing the confusion Matrix leads to the question of whether more complicated models would be warranted to catch to few wine colours that were incorrectly classified:



Conclusion

The logistic regression models were very fast to fit, less than 2 seconds in total for all three models, and provided very high performance scores.

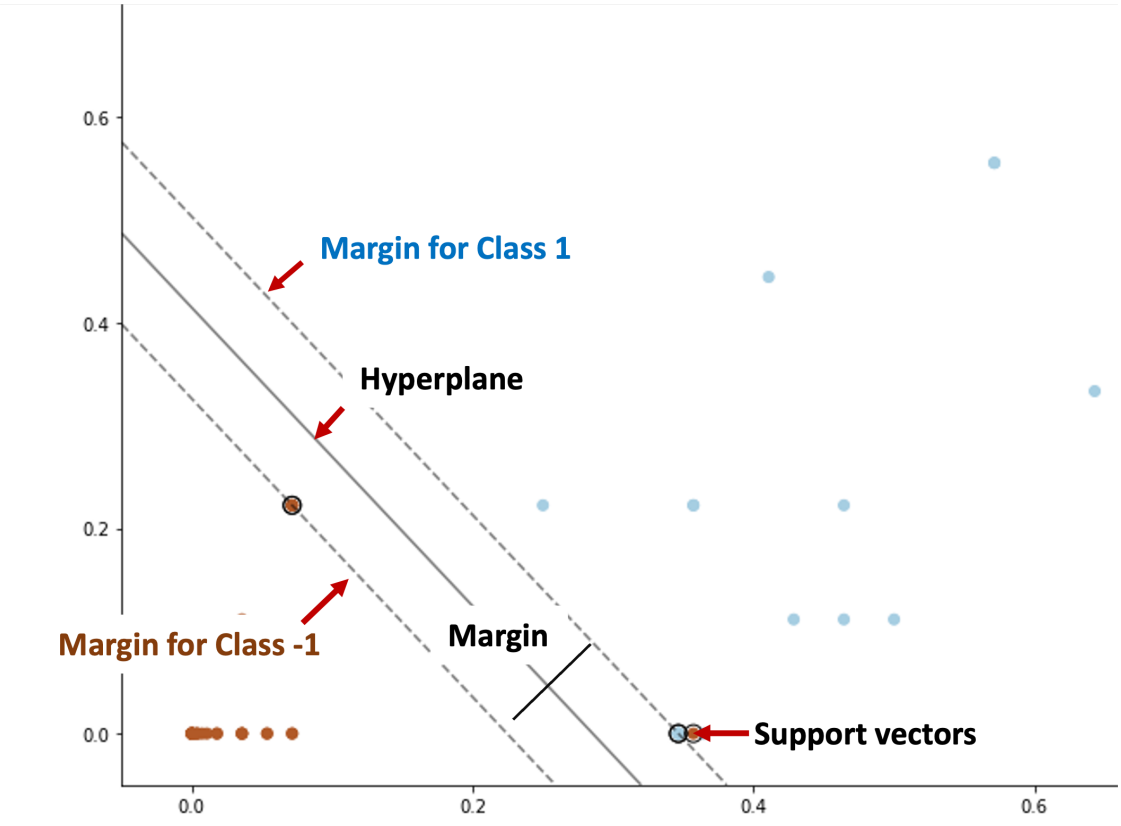
Support Vector Machines

The next model we'll consider are Support Vector Machines (SVMs). In this section we'll create two SVM. First we'll create a simple SVM with the default parameters. After this we'll use grid search to train a SVM using grid search to find the optimised hyper-parameters including identifying the best kernel to use.

Support Vector Machines are a powerful and versatile supervised machine learning algorithm used for both classification and regression tasks. At their core, SVMs aim to find the optimal hyperplane that best separates different classes in a dataset.

Key Concepts

- Hyperplane:
  - In a two-dimensional space, this is a line. In higher dimensions, it's a hyperplane. The goal is to find the hyperplane that maximises the margin between different classes.
- Margin:
  - The margin is the distance between the hyperplane and the closest data points from each class. Maximising this margin helps to improve the generalisation ability of the model.
- Support Vectors:
  - These are the data points that lie closest to the hyperplane and directly influence its position.
- Kernel Trick:
  - This is a key feature of SVMs that allows them to handle non-linear data. Kernel functions transform the input data into a higher-dimensional space, where it becomes linearly separable.



Advantages of SVMs

- Effective in High-Dimensional Spaces:
  - SVMs perform well even when the number of features is much larger than the number of samples. This makes them suitable for applications like text classification and bioinformatics.
- Memory Efficiency:

    - Because the decision function relies only on a subset of the training data (the support vectors), SVMs are relatively memory efficient.
- Versatility with Kernel Functions:
    - The kernel trick allows SVMs to handle complex, non-linear relationships between data points. Different kernel functions can be used to tailor the model to specific datasets.
- Robustness Against Overfitting:
    - By maximising the margin, SVMs tend to have good generalisation performance and are less prone to overfitting, especially in high-dimensional spaces.

Advantages over other modelling techniques

- Compared to Logistic Regression:
    - SVMs often perform better in high-dimensional spaces and with non-linear data.
    - SVMs are generally more robust to outliers.
- Compared to Decision Trees:
    - SVMs can handle high-dimensional data more effectively and are less prone to overfitting.
    - SVMs are better at finding complex decision boundaries.
- Handling Non-Linearity:
    - The kernel trick gives SVMs a distinct advantage in datasets where the classes are not linearly separable, a situation where simpler models like standard logistic regression struggle.

In essence, SVMs are a powerful tool for complex classification and regression problems, particularly when dealing with high-dimensional data or non-linear relationships.

Model Performance

The metrics used to measure the performance of Support Vector Machine (SVM) models depend on whether the SVM is used for classification or regression. Is our scenario we are only considering Classification.

- Accuracy:
    - The proportion of correctly classified instances.
    - Useful when classes are balanced.
    - Can be misleading in imbalanced datasets.
- Precision:
    - The proportion of correctly predicted positive instances out of all instances predicted as positive.
    - Important when minimizing false positives is crucial.
- Recall (Sensitivity):
    - The proportion of correctly predicted positive instances out of all actual positive instances.
    - Important when minimizing false negatives is crucial.
- F1-Score:

    - The harmonic mean of precision and recall.
    - Provides a balanced measure when both precision and recall are important, especially in imbalanced datasets.
- Area Under the ROC Curve (AUC-ROC):
    - Measures the model's ability to distinguish between positive and negative classes across various thresholds.
    - Excellent for imbalanced datasets and when you want a threshold-independent measure of performance.
- Confusion Matrix:
    - A table that shows the number of true positives, true negatives, false positives, and false negatives.
    - Provides a detailed view of the model's performance and can help identify specific types of errors.5
- Balanced Accuracy:
    - The average of recall obtained on each class.
    - Useful when dealing with imbalanced datasets.

Model Creation and Performance

Basic SVM

First the basic SVM. We continue to use the training and test data created earlier.

```
# Create the model
model = SVC(random_state=42)

# Train the model with training dataset:
model.fit(X_train, y_train.values.ravel())

# Make the predictions
svm_y_pred = model.predict(X_test)
```

To calculate the ROC AUC we need to create the model passing `probability=True` so as to predict probabilities.

```
# With Probability for AUC
model_prob = SVC(probability=True, random_state=42)
model_prob.fit(X_train, y_train.values.ravel())
svm_prob_y_pred = model_prob.predict(X_test)
```

Performance

As we can see below the basic SVM performance is very similar to the the logistic regression models with slightly better precision, accuracy and auc values. However, a poorer recall value results is a lower f1 score.

model	precision	recall	accuracy	fscore	auc
svm	0.9958	0.9896	0.9964	0.9927	0.9941

Tuned Parameters and Kernel

Next we will use a grid search to determine the best value for the `C` parameter and which kernel gives the best results:

```
params_grid = {
    "C": [0.1, 1, 10, 100, 500],
    "kernel": ["poly", "rbf", "sigmoid"],
}
opto_model = SVC(random_state=42)

# Define a GridSearchCV to search the best parameters
grid_search = GridSearchCV(
    estimator=opto_model,
    param_grid=params_grid,
    scoring="f1",
    cv=5,
    verbose=1,
)
# Search the best parameters with training data
grid_search.fit(X_train, y_train.values.ravel())
best_params = grid_search.best_params_
```

The best parameters are:

- `C`: 1.0
- Kernel: `poly`

Next we retrain a model using these parameters the same way that we generated the basic SVM model. We also repeat the process to get the probabilities so we can calculate the AUC.

Reviewing a simple print of the confusion matrix for each SVM model we can see that they both out performed the `l1` and `l2` logistic regression models.

```
Basic SVM
[[1468    2]
 [   5  475]]

Best SVM (tuned parameters)
[[1468    2]
 [   4  476]]
```

The results shown in the confusion matrix are confirmed when we look at the performance of the SVM models and compare them to the logistic regression models. Whilst the `l1` and `l2` logistic regression models do have a slightly better f1 score, the SVM models both have a higher AUC score.

Model	precision	recall	accuracy	fscore	auc
lr	0.9897	0.9897	0.9897	0.9897	0.9841
l1	0.9944	0.9944	0.9944	0.9944	0.9913
l2	0.9944	0.9944	0.9944	0.9944	0.9913
svm	0.9958	0.9896	0.9964	0.9927	0.9941
svm best	0.9958	0.9917	0.9969	0.9937	0.9952

K Nearest Neighbours

The k-nearest neighbors (KNN) algorithm is a simple, yet powerful, supervised machine learning algorithm. In our scenario we will train multiple models with `k` varying from 1 to 50 neighbours to see which model gives us the best results.

What is KNN?

- Non-parametric:
  - KNN doesn't make assumptions about the underlying data distribution. This makes it versatile for various types of data.
- Supervised learning:
  - It learns from labeled data, meaning the training data has known outcomes.
- Lazy learning:
  - KNN doesn't explicitly build a model during the training phase. Instead, it stores the training data and performs calculations when a prediction is needed.
- How it works:
  - When you want to predict the class or value of a new data point, KNN finds the "k" closest data points (neighbors) in the training set.
  - For classification: It assigns the new data point to the class that is most common among its k nearest neighbors.
  - For regression: It predicts the value of the new data point by averaging the values of its k nearest neighbors.
  - The "k" is a user-defined constant (e.g., k=3, k=5).
  - The "closeness" is typically determined by a distance metric, such as Euclidean distance.

What is KNN best used for?

KNN can be used for both classification and regression tasks, but it's particularly well-suited for:

- Simple recommendation systems:
  - "People who liked this also liked..." types of recommendations.

- Pattern recognition:
  - Image recognition, handwriting recognition.
- Classification problems with non-linear decision boundaries:
  - Because it's non-parametric, KNN can handle complex data distributions.
- Situations where interpretability is important:
  - KNN's predictions can be relatively easy to understand.
- Use in medical fields:
  - KNN can be used in genetics to calculate the probability of certain gene expressions.

Key considerations:

- Choice of "k":
  - The value of k has a significant impact on the model's performance. A small k can lead to overfitting, while a large k can lead to under-fitting.
- Distance metric:
  - The choice of distance metric (e.g., Euclidean, Manhattan) can also affect the results.
- Computational cost:
  - KNN can be computationally expensive, especially with large datasets, as it needs to calculate distances between the new data point and all training data points.
- Sensitivity to feature scaling:
  - It is very important to scale your features before using knn, as the distance calculations are heavily influenced by the scale of the different features.
- Imbalanced Datasets:
  - KNN can be sensitive to imbalanced datasets.22 You might need to use techniques like oversampling or undersampling to address this.
- Curse of Dimensionality:
  - In high-dimensional spaces, distances between data points become less meaningful, which can degrade KNN's performance.23

In general, you use many of the same evaluation metrics for KNN models as you do for logistic regression and SVMs, especially for classification tasks i.e. accuracy, precision, recall f1 score and AUC.

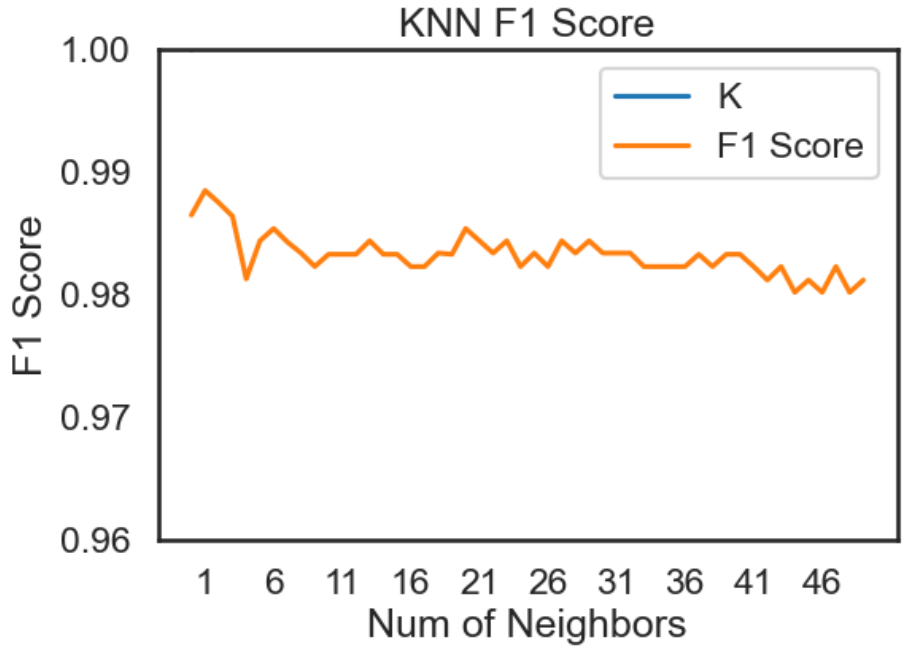
Model Creation and Performance

As previously mentioned we will train multiple models with `k` varying from 1 to 50 to determine which value gives the best model performance. For each model we will calculate the f1 score and plot the scores so that we can identify the best `k` value.

```
# Try K from 1 to 50
max_k = 50

# Create an empty list to store f1score for each k
```

```
f1_scores = []

# Then we will train 50 KNN classifiers with K ranged from 1 to 50.
for k in range(1, max_k + 1):
    # Create a KNN classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    # Train the classifier
    knn = knn.fit(X_train, y_train.values.ravel())
    preds = knn.predict(X_test)
    # Evaluate the classifier with f1score
    f1 = f1_score(preds, y_test)
    f1_scores.append((k, round(f1_score(y_test, preds), 4)))
# Convert the f1score list to a dataframe
f1_results = pd.DataFrame(f1_scores, columns=["K", "F1 Score"])
f1_results.set_index("K")

# This is a long list and different to analysis, so let's visualize the list using a
linechart.
# Plot F1 results
ax = f1_results.plot(figsize=(6, 4))
ax.set(xlabel="Num of Neighbors", ylabel="F1 Score")
ax.set_xticks(range(1, max_k, 5))
plt.ylim((0.96, 1))
plt.title("KNN F1 Score")
```

When we plot the f1 scores we can clearly see the the best `k` value is 2. We will now train a KNN model using a `k` value of two and compare the performance of the model to the other models tested to date.



AUC is not as readily available so was not calculated but by looking at the other metrics and the confusion matrix we can see that the KNN model is the poorest performing so far:

Model	precision	recall	accuracy	fscore	auc
lr	0.9897	0.9897	0.9897	0.9897	0.9841
l1	0.9944	0.9944	0.9944	0.9944	0.9913
l2	0.9944	0.9944	0.9944	0.9944	0.9913
svm	0.9958	0.9896	0.9964	0.9927	0.9941
svm best	0.9958	0.9917	0.9969	0.9937	0.9952
knn	0.9812	0.9812	0.9908	0.9812	nan

```
KNN
[[1461    9]
 [   9  471]]
```

Decision Trees

Decision trees stand out for their interpretability and ability to handle both numerical and categorical data. Decision tree models differ significantly from logistic regression, SVMs, and KNN models in several key aspects, primarily in their structure, learning process, interpretability, and handling of data.

Similar to our approach with SVM we will first train a decision tree using the default parameters. This means that there is no restriction on the depth of the tree which may lead to overfitting. We will then use a grid search to determine the best hyper parameters to tune the optimal decision tree.

Structure and Learning Process

- Decision Trees:
  - Hierarchical, tree-like structure.
  - Learn by recursively partitioning the data based on feature values.
  - Use greedy algorithms (e.g., ID3, C4.5, CART) to find the best split at each node.
  - Can handle both classification and regression tasks.
  - Non-parametric: they make no assumptions about the distribution of the data.

Interpretability

- Decision Trees:
  - Highly interpretable, especially for small trees.
  - The decision rules can be easily visualised and understood.
  - Feature importance can be readily derived.
- Logistic Regression:
  - Relatively interpretable.

    - The coefficients of the features indicate their impact on the log-odds of the outcome.15
- SVM:
    - Less interpretable, especially with non-linear kernels.
    - Difficult to understand the relationships between features and predictions.
- KNN:
    - Relatively interpretable for small k.
    - You can see the nearest neighbours that influenced the prediction.

Handling of Data

- Decision Trees:
    - Can handle both numerical and categorical data.
    - Relatively robust to outliers.
    - Can handle missing values (depending on the implementation).
- Logistic Regression:
    - Requires numerical features (categorical features need to be encoded).
    - Sensitive to outliers.
    - Assumes linearity.
- SVM:
    - Requires numerical features.
    - Can be sensitive to outliers.
    - Feature scaling is crucial.
- KNN:
    - Requires numerical features.
    - Sensitive to outliers.
    - Feature scaling is crucial.

Model Complexity and Overfitting

- Decision Trees:
    - Prone to overfitting, especially with deep trees.
    - Can be controlled by pruning or limiting tree depth.
- Logistic Regression:
    - Less prone to overfitting than decision trees and regularisation techniques can be used to prevent overfitting.
- SVM:
    - Can overfit, especially with complex kernels or small margins but regularisation (`C` parameter) helps to control overfitting.

- KNN:
    - Can overfit with small `k`. Can under fit with large `k`.

5. Decision Boundaries:

- Decision Trees:
    - Create axis-parallel decision boundaries and can model complex, non-linear relationships.
- Logistic Regression:
    - Creates linear decision boundaries so limited to modelling linear relationships.
- SVM:
    - Can create linear or non-linear decision boundaries (depending on the kernel).
- KNN:
    - Creates complex, non-linear decision boundaries.

Model Creation and Performance

Following the same steps we performed with the SVM models we first created a basic decision tree model with the defaults, no parameter tuning, and calculated the performance and confusion matrix. The node count and max depth of the tree were also calculated.

```
Basic Decision Tree Confusion Matrix
[[1459   11]
 [   7  473]]

 Node Count: 143
 Max Depth: 18
```

Next, using a grid search approach the "optimised" parameters for a decision tree model was determined. Looking at the confusion matrix we can quickly see that the "best" model was simpler with only 99 nodes and with a max depth of 9 but it did misclassify one additional red and one additional white wine incorrectly.

```
Best Decision Tree Confusion Matrix
[[1458   12]
 [   8  472]]

 Node Count: 99
 Max Depth: 9
```

The performance metrics tell the same story. The basic decision tree model was on a par with the KNN model from an f1 score perspective but the "best" decision tree model has given the worst model performance thus far.

Model	precision	recall	accuracy	fscore	auc
lr	0.9897	0.9897	0.9897	0.9897	0.9841
l1	0.9944	0.9944	0.9944	0.9944	0.9913
l2	0.9944	0.9944	0.9944	0.9944	0.9913
svm	0.9958	0.9896	0.9964	0.9927	0.9941
svm best	0.9958	0.9917	0.9969	0.9937	0.9952
knn	0.9812	0.9812	0.9908	0.9812	nan
dt	0.9773	0.9854	0.9908	0.9813	nan
dt best	0.9752	0.9833	0.9897	0.9793	nan

Recommended Final Model

A paragraph explaining which of your classifier models you recommend as a final model that best fits your needs in terms of accuracy and explainability.

Based on the performance metrics and the characteristics of the different models evaluated, I would recommend the Support Vector Machine (SVM) with tuned parameters as the final model that best fits the needs in terms of accuracy and explainability. The SVM model with optimized parameters achieved the highest accuracy (0.9969), precision (0.9958), and AUC (0.9952), indicating its superior ability to correctly classify wine colors and distinguish between classes.

While logistic regression models, particularly the L1 and L2 regularized versions, also performed exceptionally well with high accuracy and precision, the SVM's robustness against overfitting and its capability to handle high-dimensional data make it a more reliable choice for this dataset. Additionally, the SVM's use of the kernel trick allows it to manage non-linear relationships within the data effectively. Although decision trees and KNN models offer interpretability advantages, their performance metrics were slightly lower, and they are more prone to overfitting and sensitivity to feature scaling, respectively. Therefore, the SVM with tuned parameters strikes the best balance between high accuracy and the ability to generalize well to new data, making it the most suitable model for this classification task.

Key Findings and Insights

Summary Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your classifier model.

The Support Vector Machine (SVM) models demonstrated exceptional performance in predicting the color of wine, outperforming other models such as logistic regression and K-Nearest Neighbors (KNN). The basic SVM model, trained with default parameters, achieved a precision of 0.9958, recall of 0.9896, accuracy of 0.9964, F1-score of 0.9927, and an AUC of 0.9941. These metrics indicate that the SVM model is highly effective in distinguishing between red and white wines, with a particularly high precision and accuracy, suggesting that it makes very few false positive predictions.

Further optimization of the SVM model through grid search, which identified the best parameters as C=1.0 and kernel=poly, resulted in even better performance. The tuned SVM model achieved a precision of 0.9958, recall of 0.9917, accuracy of 0.9969, F1-score of 0.9937, and an AUC of 0.9952. The confusion matrix for the tuned SVM model showed minimal misclassifications, with only a few instances of red and white wines being incorrectly classified. This indicates that the SVM model, especially when tuned, is highly reliable and robust in predicting wine color, making it an excellent choice for this classification task.

The insights derived from the SVM classifier model highlight its ability to handle complex, non-linear relationships in the data, thanks to the kernel trick. The high AUC values indicate that the model is effective across various thresholds, making it suitable for imbalanced datasets. The SVM's robustness against overfitting and its efficiency in high-dimensional spaces further underscore its suitability for this classification problem. Overall, the SVM models, particularly the tuned version, provide a powerful and accurate tool for predicting wine color based on its chemical properties.

Next Steps

Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model after adding specific data features that may help you achieve a better explanation or a better prediction.

To further enhance the analysis and improve the model's performance, the following steps can be considered:

1. **Feature Engineering**: Investigate and create new features that might capture more complex relationships within the data. For example, interaction terms between existing features or polynomial features could be explored.
2. **Addressing Class Imbalance**: Since there is a class imbalance in the target variable (color), consider using techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or ADASYN (Adaptive Synthetic Sampling) to generate synthetic samples for the minority class. Alternatively, you could try undersampling the majority class.
3. **Hyperparameter Tuning**: Perform a more extensive hyperparameter tuning for all models using techniques like Random Search or Bayesian Optimization, which might yield better results than Grid Search.
4. **Ensemble Methods**: Combine multiple models using ensemble techniques such as bagging, boosting (e.g., Gradient Boosting, AdaBoost), or stacking to potentially improve predictive performance.
5. **Cross-Validation**: Implement k-fold cross-validation to ensure that the model's performance is consistent across different subsets of the data and to reduce the risk of overfitting.
6. **Feature Importance Analysis**: Conduct a thorough analysis of feature importance to understand which features contribute most to the model's predictions. This can help in refining the feature set and potentially removing less important features.
7. **Model Interpretability**: Use model interpretability techniques such as SHAP (SHapley Additive exPlanations) values or LIME (Local Interpretable Model-agnostic Explanations) to gain insights into how the model makes predictions and to ensure that the model's decisions are understandable and justifiable.
8. **Data Augmentation**: If possible, augment the dataset with additional data points. This could involve collecting more data or using data augmentation techniques to artificially increase the size of the dataset.
9. **Handling Outliers**: Identify and handle outliers in the data, as they can significantly impact model performance. Techniques such as robust scaling or transforming the data can be considered.
10. **Exploring Different Algorithms**: Experiment with other machine learning algorithms that were not initially considered, such as Random Forests, Gradient Boosting Machines, or Neural Networks, to see if they offer better performance.
11. **Regular Updates and Monitoring**: Once the model is deployed, continuously monitor its performance and update it with new data to ensure it remains accurate and relevant over time.

By taking these steps, we can refine our model, improve its predictive power, and ensure that it provides valuable insights and accurate predictions for stakeholders.