Recap

Module 3 | Chapter 2 | Notebook 1

In this notebook we'll look at the most important things from the previous chapter. This is all repetition. If you discover a lot of new things here, we recommend taking a look back at Chapter 1. The relevant lessons for each section are clearly marked.

Model agnostic methods for model interpretation

The predictions of some models can be interpreted very well. For example, linear regression gives each feature a weight, which allows us to understand the impact it has on the prediction. However, there are numerous models that are not as easy to interpret. These are called black-box models. These include artificial neural networks and the random forests. In the last chapter, we looked at some methods to understand how much which features influence the predictions. These are model agnostic and can therefore be used for all models. Now let's recap them briefly.

Let's import the wine data from *Polynomial Regression, Module 1, Chapter 4* to do this:

In [1]:
```
import pandas as pd

features_train = pd.read_pickle('features_train.p')
features_test = pd.read_pickle('features_test.p')
target_train = pd.read_pickle('target_train.p')
target_test = pd.read_pickle('target_test.p')

features_train.head()
```

Out[1]:

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dio
0	11.9	0.38	0.51	2.0	0.121	7.0	
1	9.0	0.46	0.31	2.8	0.093	19.0	
2	7.5	0.20	0.41	1.2	0.050	26.0	1
3	6.5	0.44	0.49	7.7	0.045	16.0	1
4	6.6	0.32	0.33	2.5	0.052	40.0	2

You can see that the data contains the different chemical properties of all the wines. The following data dictionary explains what the data means.

Column number	Column name	Type	Description
0	`'fixed.acidity'`	continuous (`float`)	liquid acid content.
1	`'volatile.acidity'`	continuous (`float`)	gaseous acid content, particularly relevant for the odor.
2	`'citiric.acid'`	continuous (`float`)	citric acid content. Part of the liquid acid content. Affects freshness in taste.
3	`'residual.sugar'`	continuous (`float')	Natural sugar of the grapes, which is still present at the end of fermentation stage.
4	`'chlorides'`	continuous (`float`)	chloride content. Minerals, which can be dependent on the wine growing region.
5	`'free.sulfur.dioxide'`	continuous (`float`)	free sulfur dioxide Perceptible by smell.
6	`'total.sulfur.dioxide'`	continuous (`float`)	total sulfur dioxide content
7	`'density'`	continuous (`float`)	density of the wine
8	`'PH'`	continuous (`float`)	pH value. Determined by the acid content.
9	`'sulphates'`	continuous (`float`)	sulphate content.
10	`'alcohol'`	continuous (`float`)	alcohol content
11	`'color'`	categorical	color of the wine. Only contains `0` (white wine) and `1` (red wine)

This time we want to investigate what makes a good wine according to our model. If a wine has been evaluated as good (ranking of 7 or more), the target vector contains a `1` in the corresponding position.

Run the following cell to match a *random forest* of 200 decision trees to the data:

In [2]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=200, random_state=1)
print('accuracy:', cross_val_score(estimator=model, X=features_train, y=target_train,
print('recall:', cross_val_score(estimator=model, X=features_train, y=target_train, cv
print('precision:', cross_val_score(estimator=model, X=features_train, y=target_train,
```

```
accuracy: 0.8739986696538243
recall: 0.5210843373493976
precision: 0.7727047837215902
```

The simple model already predicts 87% of the wines correctly. However, only 52% of the good wines are identified. If the model identifies a wine as good, this is correct in 77% of the cases.

We want to find out how much a feature contributes to our predictions. One method for this is the *permutation feature importance*. The idea is to determine how much worse the predictions become when the information in a column is lost.

The following steps are performed for the permutation feature importance:

1. The model is fitted to the data with all the hyperparameters as usual
2. The prediction quality $V_{orig}$ is calculated with a metric
3. The values in a single column, are shuffled randomly, so the link between this feature and the classes is lost. The distribution of the column values remains the same.
4. The prediction quality $V_{perm}$ is calculated based on the permuted (shuffled) column.
5. The feature importance is calculated either as the ratio $\frac{V_{orig}}{V_{perm}}$, or as the difference $V_{orig}-V_{perm}$.

In the following cell we'll iterate through the individual steps. It is important to "unshuffle" the values in one column before calculating the permutation feature importance* for the next column.

In [3]:
```
# step 1: fit model
model.fit(features_train, target_train)

# step 2: calculate recall on test data to get features, which are important for good
from sklearn.metrics import recall_score
recall_orig = recall_score(target_test, model.predict(features_test))

# steps 3 to 5 for every column
perm_importances = []
features_test_perm = features_test.copy()

for col in features_test_perm.columns:
    # step 3: shuffle values of the feature
    col_perm = features_test_perm.loc[:, col].sample(frac=1, replace=False, random_sta
    features_test_perm.loc[:, col] = col_perm  # replace column with shuffled column

    # step 4: calculate recall with shuffled values
    target_test_pred_perm = model.predict(features_test_perm)
    recall_perm = recall_score(target_test, target_test_pred_perm)

    # step 5: calculate features importance as difference
    perm_importances.append(recall_orig-recall_perm)

    # reset permutation
    features_test_perm.loc[:, col] = features_test.loc[:, col]

# print the feature importance and the column names
list(zip(features_test.columns, perm_importances))
```

```
Out[3]: [('fixed.acidity', 0.03202846975088969),
         ('volatile.acidity', 0.14234875444839856),
         ('citric.acid', 0.07473309608540923),
         ('residual.sugar', 0.042704626334519546),
         ('chlorides', 0.19217081850533801),
         ('free.sulfur.dioxide', 0.07473309608540923),
         ('total.sulfur.dioxide', 0.092526690391459),
         ('density', 0.2170818505338078),
         ('pH', 0.060498220640569 31),
         ('sulphates', 0.08540925266903909),
         ('alcohol', 0.3558718861209964),
         ('color', 0.0)]
```

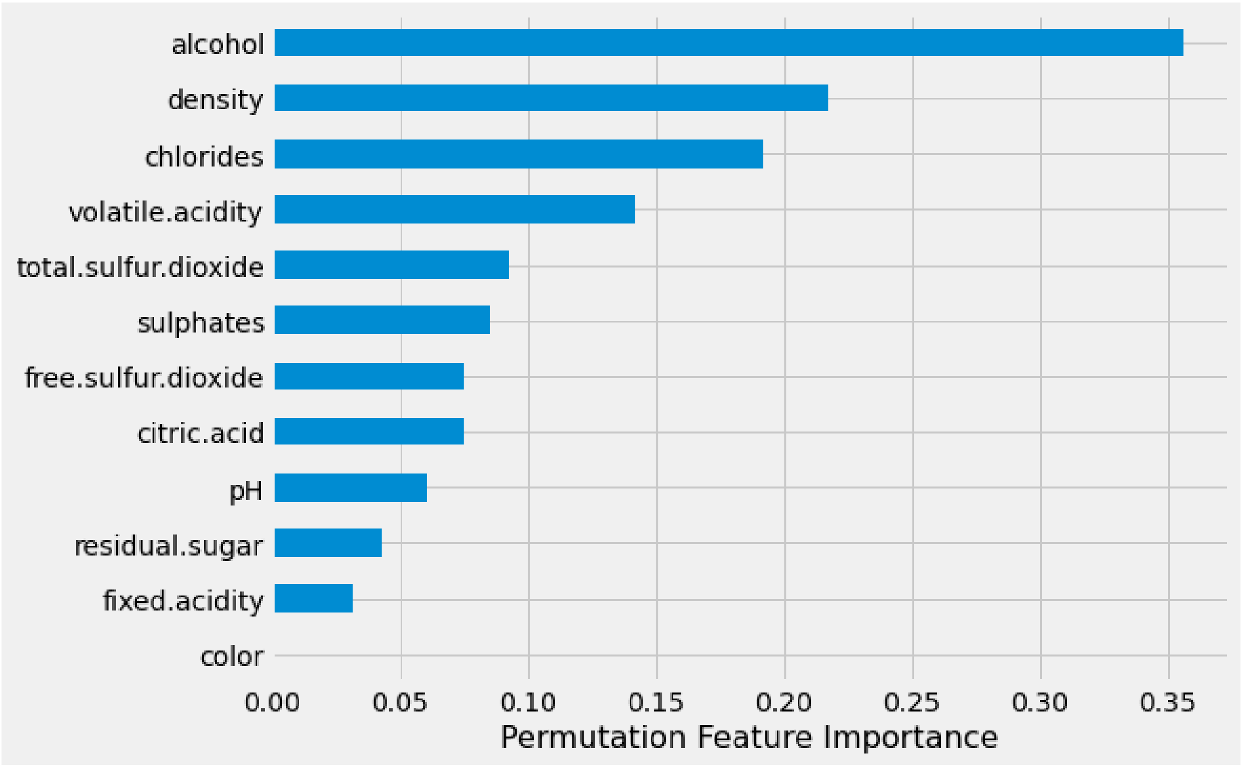
Some features stand out as more important than others. We'll sort the values and display them as a bar chart.

```
In [4]: import matplotlib.pyplot as plt

        plt.style.use('fivethirtyeight')

        fig, ax = plt.subplots(figsize=(8, 6))
        ax.set_xlabel('Permutation Feature Importance', size=16)

        # Convert feature importance array into a series and visualize
        perm_importance = pd.Series(perm_importances, index=features_test.columns).sort_values
        perm_importance.plot(kind='barh', ax=ax);
```



We used the difference in recall values between normal and permuted `DataFrame` as the permutation feature importance. The alcohol content seems to be particularly important for our model. If we shuffle these values, the recall is reduced by about 0.35. The permutation feature importance only gives us the importance of a feature and does not describe how the values the feature takes on affect our predictions.

Partial dependence plots (PDPs) can help us here. PDPs show the marginal influence a feature has on an average prediction. They can visually represent both linear and non-linear relationships between features and predictions. To calculate the partial dependency of a selected feature, the value of this feature is replaced by a given value in all data points. Then the average of all predictions is calculated before the next value is tried out. This way the average prediction is calculated as a function of the selected feature.

You can create PDPs with `pdp` from the `pdpbox` module. You can find the module documentation here.

In [5]:
```
from pdpbox import pdp
```

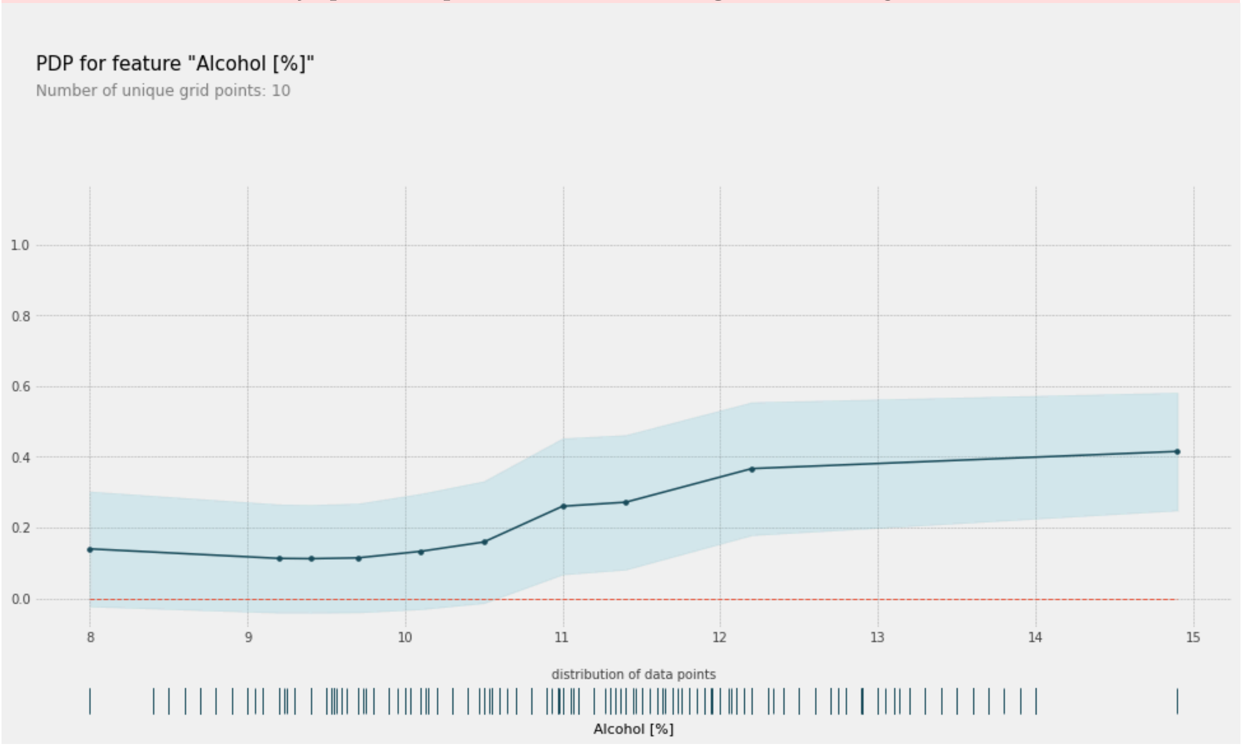
With the `pdp.pdp_isolate()` function, we can calculate the partial dependency:

In [6]:
```
pdp_alcohol = pdp.pdp_isolate(model=model,  # a fitted sklearn model
                              dataset=features_train,  # the dataset on which the mode
                              model_features=features_train.columns,  # names of all t
                              num_grid_points=10,  # number of values to explore
                              feature="alcohol")  # feature's column name in `dataset`
```

The output is an object which we can visualize with `pdp.pdp_plot()`.

In [7]:
```
pdp.pdp_plot(pdp_isolate_out=pdp_alcohol,  # output of pdp.isolate()
             feature_name="Alcohol [%]",  # name of feature (for title)
             plot_pts_dist=True,  # display real feature values
             center=False);  # do not center the plot
```

```
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
```



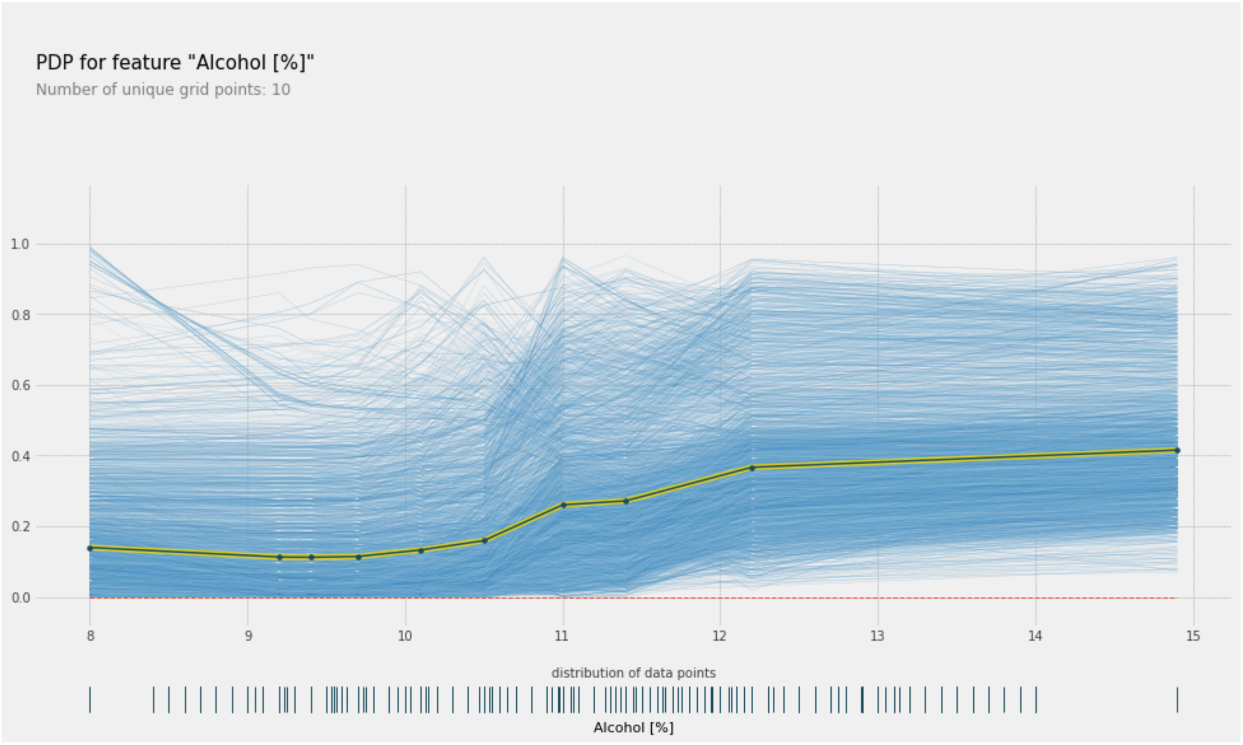
On the y-axis, the PDP shows us the average prediction, i.e. the probability of a wine being classified as good wine. The alcohol content of the wines is displayed on the x axis. The blue points represent the values which the partial dependency was actually calculated for. The light blue sleeve shows the standard deviation of the predictions. The actual values of the alcohol content in the training data are displayed as as blue lines under the x-axis.

The PDP shows us that, according to the model, an increasing alcohol content leads to a better evaluation of the wines. The average prediction improves from about 15% for low alcohol content (8% alcohol) to about 40% for high alcohol content (15% alcohol).

With the PDP we can see the average effect of a feature value on the prediction. It is therefore a global method that tells us something about the overall model, but not about individual predictions. The local version of a PDP is the *individual conditional expectation* (ICE) *plot*. An ICE plot visualizes the dependence of the predictions on a feature separately for each data point. This way we get one line per data point instead of one line in total like in the PDP. The PDP corresponds to the average of the ICEs.

To see the individual data points, we'll use the `pdp.pdp_plot()` function again. But this time we have to set the parameter `plot_lines=True`.

In [8]:
```
pdp.pdp_plot(pdp_isolate_out=pdp_alcohol,
             feature_name="Alcohol [%]",
             plot_lines=True,  # show a single line for every datapoint
             plot_pts_dist=True,
             center=False);
```



In addition to the average dependency, you will now see numerous thin blue lines. Each one represents a wine in the data set, whose alcohol content we change. We can see from their course that there isn't an average correlation for all the wines to the same extent. Some wines are classified as good wines if they have a low alcohol content. This probability initially decreases as the alcohol content increases.

Congratulations: You have refreshed your knowledge on permutation feature importance, partial dependence plots and individual conditional expectation plots. These methods can help you to evaluate your model and understand how the model generated its predictions.

Remember:

- You can use permutation feature importance to estimate the importance of the features
- *Partial dependence plots* (PDPs) show the average influence of feature values on the predictions
- *Independent conditional expectation plots* (ICE plots) visualize the influence of feature values for each data point separately
- You can create PDPs and ICE plots with the `pdpbox` module.

The data was published here first: P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.
