ML visualizations : : **CHEAT SHEET**

Usage

- **Inter cluster distance** map shows how distant are the cluster centres and the relative sizes of them.
- From **Clustering heatmap** one may tell how are the examples clustered and what features differentiate clusters.
- Locking at the **Elbow plot** it's easy to find optimal parameters.
- **Parallel coordinates** visualize the characteristic features for the chosen clusters.

- **Beeswarm** is to get an overview of which features are most important for a model.
- On the **Waterfall** plot we see features each contributing to push the model output from the base value to the model output.

Libraries

For Inter cluster distance and Elbow plot:

```
import yellowbrick.cluster
```

For Parallel coordinates:

```
import yellowbrick.features
```

For Clustering heatmap:

```
import seaborn as sns
```

For Beeswarm and Waterfall plots:

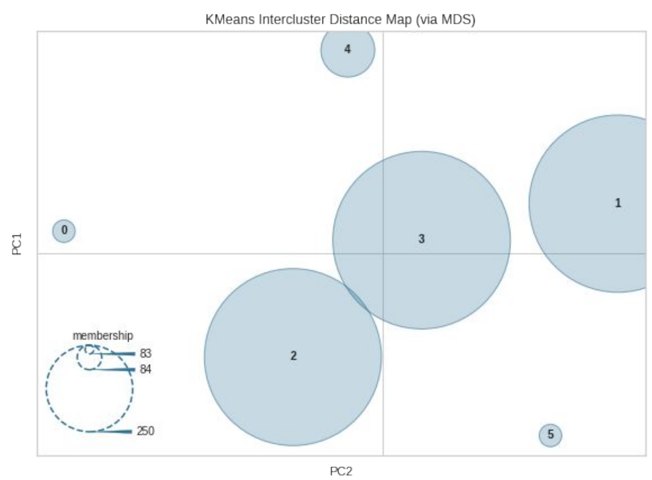
```
import shap
```

Links

- [yellowbrick]
- [seaborn]
- [SHAP]

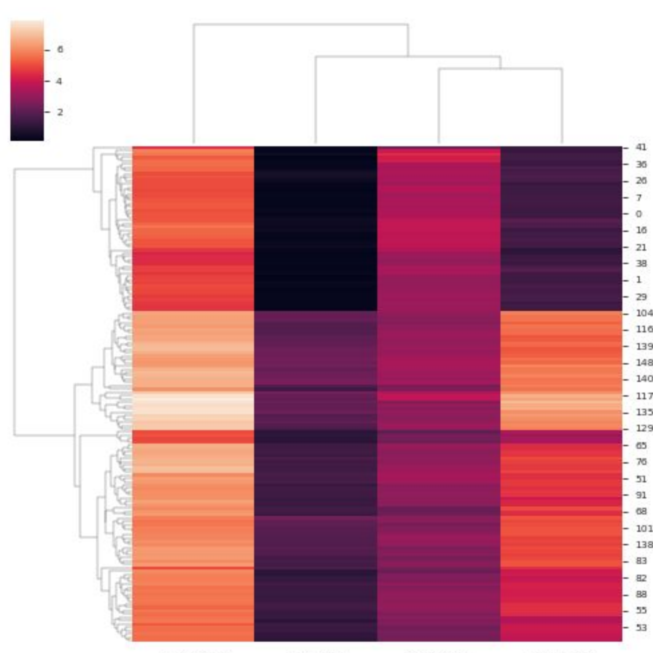
Clustering

Inter cluster distance map:



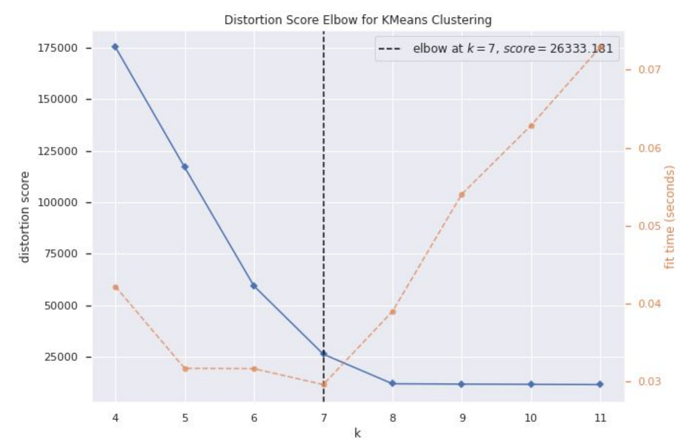

```
visualizer =
yellowbrick.cluster.InterclusterDistance(mode
l, size = (700, 500))
visualizer.fit(X)
visualizer.show()
```

Clustering heatmap:



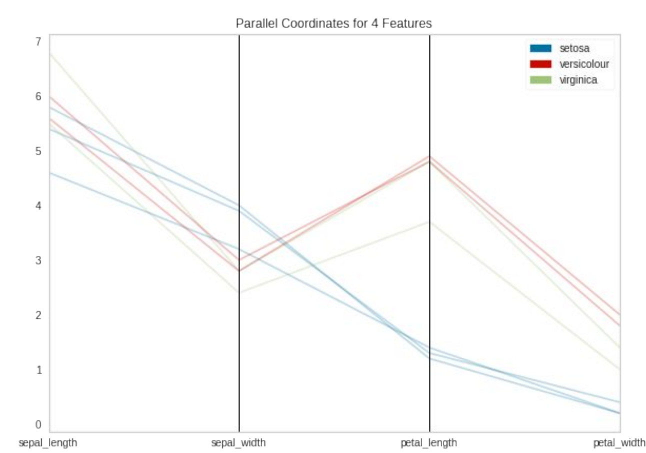

```
heatmap = sns.clustermap(data)
```

Elbow plot for KMeans:




```
visualizer =
yellowbrick.cluster.KElbowVisualizer(sklear
n.cluster.KMeans(), k = (4, 12), metric =
"distortion", size = (700, 500))
visualizer.fit(X)
visualizer.show()
```

Parallel coordinates plot:

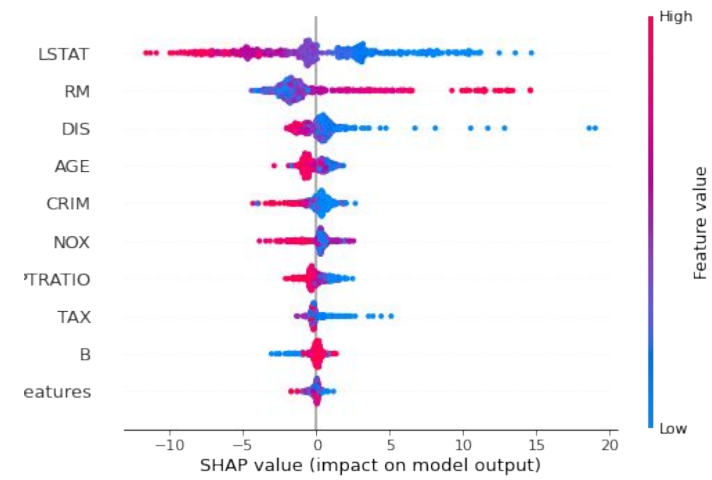



```
features = ["sepal_length", "sepal_width",
"petal_length", "petal_width"]
classes = ["setosa", "virginica",
"versicolour"]

visualizer =
yellowbrick.features.ParallelCoordinates(cl
asses = classes, features = features,
sample = 0.05, shuffle = True, size = (700,
500))
visualizer.fit_transform(X, y)
visualizer.show()
```

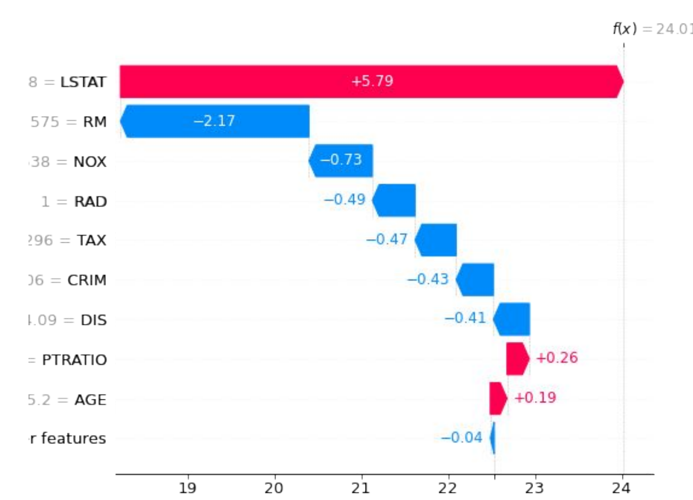
Feature analysis

Beeswarm plot:




```
explainer = shap.Explainer(model)
shap_values = explainer(X)
shap.plots.beeswarm(shap_values)
```

Waterfall plot:




```
explainer = shap.Explainer(model)
shap_values = explainer(X)
shap.plots.waterfall(shap_values[0])
```

POZNAN UNIVERSITY OF TECHNOLOGY

Usage

- **Correlation matrix** displays how correlated are attributes.
- **Confusion matrix** shows the performance of the classification algorithm.
- **Validation curve** plots the model's scores over a varying hyper parameter.
- **Learning curve** plots the model's scores over varying numbers of training samples.
- **ROC curves** tell how much the model is capable of distinguishing between classes.
- **Calibration plots** helps to figure out if the probabilities of classifier trustworthy.

Libraries

For Correlation matrix:

```
import seaborn as sns
import numpy as np
```

For Confusion matrix:

```
import sklearn.metrics
import seaborn as sns
import numpy as np
```

For Validation curve:

```
import yellowbrick.model_selection
import numpy as np
```

For Learning curve:

```
import yellowbrick.model_selection
```

For ROC curves:

```
import yellowbrick.classifier
```

For Calibration plots:

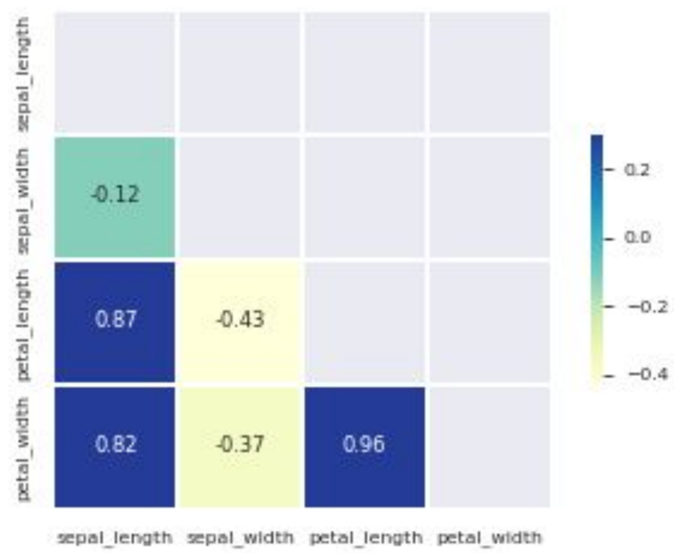
```
import scikitplot as skplt
```

Links

- [sklearn]
- [skplot]
- [numpy]

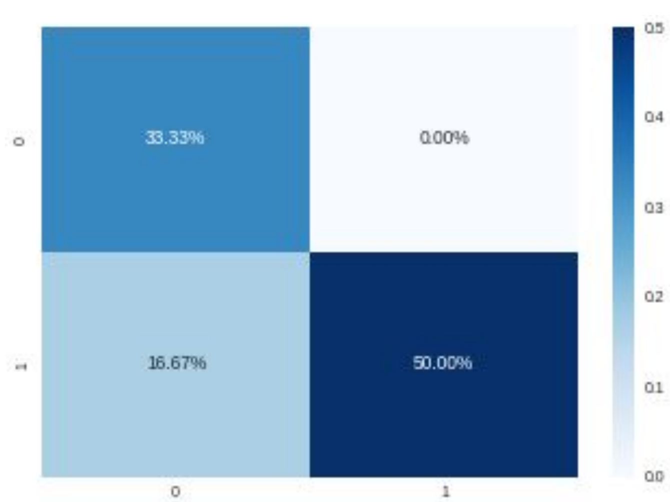
Classification

Correlation matrix:



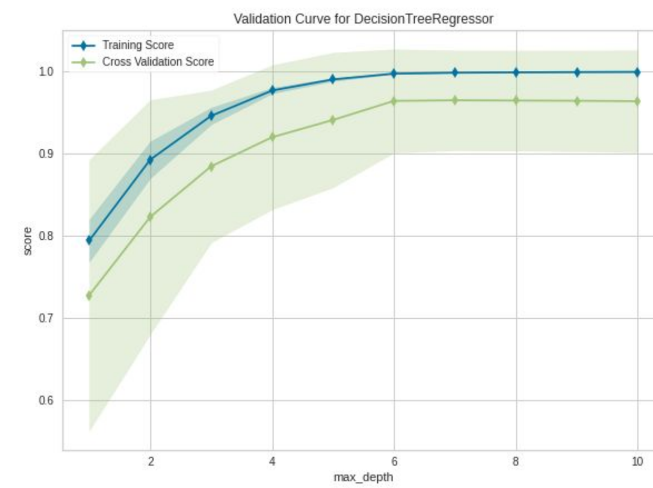

```
mask = np.triu(np.ones_like(data.corr(),
dtype = bool))
fig = sns.heatmap(data.corr(), mask = mask,
cmap = "YlGnBu", annot = True, vmax = 0.3,
center = 0, square = True, linewidths = 0.5,
cbar_kws = {'shrink':0.5})
```

Confusion matrix:



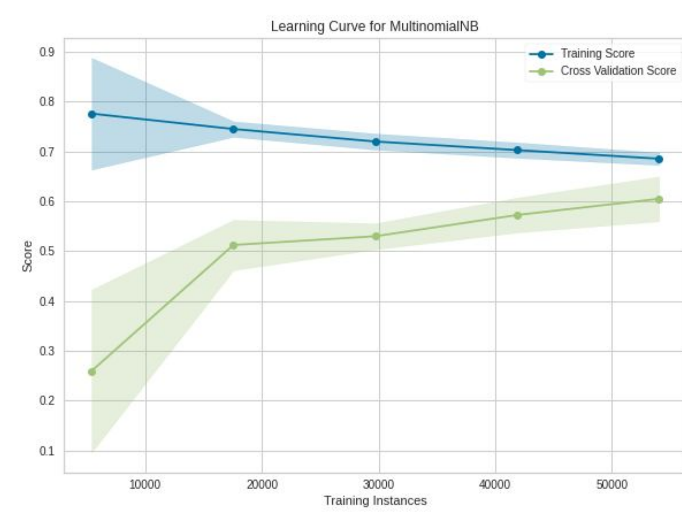

```
cm = sklearn.metrics.confusion_matrix(y,
y_pred)
fig = sns.heatmap(cm/np.sum(cm), fmt = '.2%',
annot = True, cmap = 'Blues')
```

Validation curve:



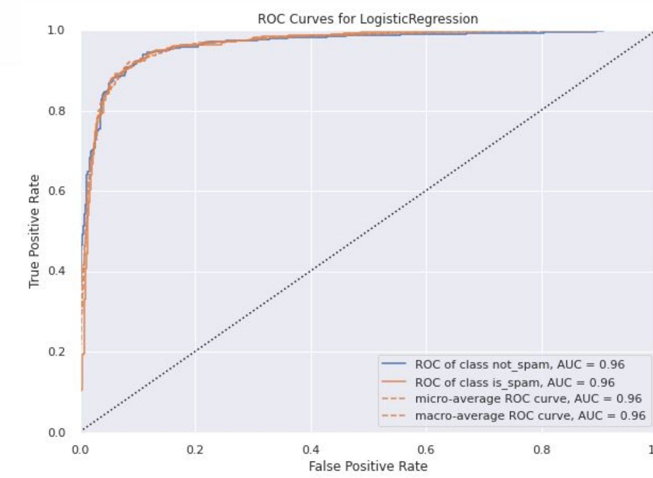

```
visualizer =
yellowbrick.model_selection.ValidationCurve(m
odel, param_name = "max_depth", param_range =
np.arange(1, 11), scoring = "r2", size =
(700, 500))
visualizer.fit(X, y)
visualizer.show()
```

Learning curve:



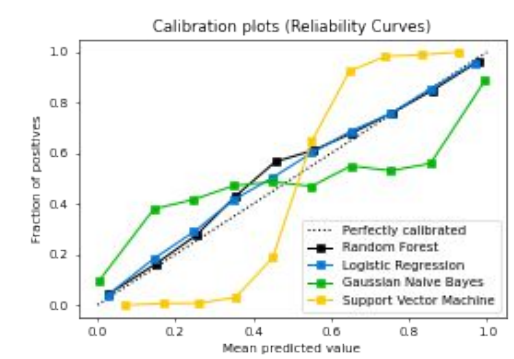

```
visualizer =
yellowbrick.model_selection.LearningCurve(
model, scoring = 'f1_weighted', size = (700,
500))
visualizer.fit(X, y)
visualizer.show()
```

ROC curves:




```
classes = ["not_spam", "is_spam"]
visualizer =
yellowbrick.classifier.ROCAUC(model, classes
= classes, size = (700, 500))
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```

Calibration plots:




```
rf_probas = RandomForestClassifier().fit(X_train,
y_train).predict_proba(X_test)
lr_probas = LogisticRegression().fit(X_train,
y_train).predict_proba(X_test)
nb_probas = GaussianNB().fit(X_train,
y_train).predict_proba(X_test)
svm_scores = LinearSVC().fit(X_train,
y_train).decision_function(X_test)

probas_list = [rf_probas, lr_probas,
nb_probas, svm_scores]
clf_names = ['Random Forest',
'Logistic Regression', 'Gaussian Naive Bayes',
'Support Vector Machine']
skplt.metrics.plot_calibration_curve(y_test,
probas_list, clf_names)
```