Feature Selection for Machine Learning

This section lists 4 feature selection recipes for machine learning in Python

This post contains recipes for feature selection methods.

Each recipe was designed to be complete and standalone so that you can copy-and-paste it directly into you project and use it immediately.

Recipes uses the Pima Indians onset of diabetes dataset to demonstrate the feature selection method . This is a binary classification problem where all of the attributes are numeric.

Dataset File. Dataset Details.

Univariate Selection

Statistical tests can be used to select those features that have the strongest relationship with the output variable.

The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features.

Many different statistical test scan be used with this selection method. For example the ANOVA F-value method is appropriate for numerical inputs and categorical data, as we see in the Pima dataset. This can be used via the f_classif() function. We will select the 4 best features using this method in the example below.

In [ ]:
```
# Feature Selection with Univariate Statistical Tests
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
# load data
filename = './pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cla
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=f_classif, k=4)
fit = test.fit(X, Y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
```

```
# summarize selected features
print(features[0:5,:])
```

```
[ 39.67  213.162   3.257   4.304  13.281  71.772  23.871  46.141]
[[  6.  148.   33.6  50. ]
 [  1.   85.   26.6  31. ]
 [  8.  183.   23.3  32. ]
 [  1.   89.   28.1  21. ]
 [  0.  137.   43.1  33. ]]
```

For help on which statistical measure to use for your data, see the tutorial:

How to Choose a Feature Selection Method For Machine Learning Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

You can see the scores for each attribute and the 4 attributes chosen (those with the highest scores). Specifically features with indexes 0 (preq), 1 (plas), 5 (mass), and 7 (age).

Recursive Feature Elimination

The Recursive Feature Elimination (or RFE) works by recursively removing attributes and building a model on those attributes that remain.

It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

You can learn more about the RFE class in the scikit-learn documentation.

The example below uses RFE with the logistic regression algorithm to select the top 3 features. The choice of algorithm does not matter too much as long as it is skillful and consistent.

In [ ]:
```
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
url = "./pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cla
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = LogisticRegression(solver='lbfgs', max_iter=1000)
rfe = RFE(model, n_features_to_select=3)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

```
Num Features: 3
Selected Features: [ True False False False False  True  True False]
Feature Ranking: [1 2 4 6 5 1 1 3]
```

You can see that RFE chose the the top 3 features as preg, mass and pedi.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

These are marked True in the support_ array and marked with a choice “1” in the ranking_ array.

Principal Component Analysis

Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form.

Generally this is called a data reduction technique. A property of PCA is that you can choose the number of dimensions or principal component in the transformed result.

In the example below, we use PCA and select 3 principal components.

Learn more about the PCA class in scikit-learn by reviewing the PCA API. Dive deeper into the math behind PCA on the Principal Component Analysis Wikipedia article.

In [ ]:
```
# Feature Extraction with PCA
import numpy
from pandas import read_csv
from sklearn.decomposition import PCA
# load data
url = "./pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cla
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

```
Explained Variance: [0.889 0.062 0.026]
[[-2.022e-03  9.781e-02  1.609e-02  6.076e-02  9.931e-01  1.401e-02
   5.372e-04 -3.565e-03]
 [-2.265e-02 -9.722e-01 -1.419e-01  5.786e-02  9.463e-02 -4.697e-02
  -8.168e-04 -1.402e-01]
 [-2.246e-02  1.434e-01 -9.225e-01 -3.070e-01  2.098e-02 -1.324e-01
  -6.400e-04 -1.255e-01]]
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

You can see that the transformed dataset (3 principal components) bare little resemblance to the source data.

Feature Importance

Bagged decision trees like Random Forest and Extra Trees can be used to estimate the importance of features.

In the example below we construct a ExtraTreesClassifier classifier for the Pima Indians onset of diabetes dataset. You can learn more about the ExtraTreesClassifier class in the scikit-learn API.

In [ ]:
```
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# load data
url = "./pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cla
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = ExtraTreesClassifier(n_estimators=10)
model.fit(X, Y)
print(model.feature_importances_)
```

```
[0.106 0.244 0.105 0.085 0.08  0.141 0.109 0.129]
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

You can see that we are given an importance score for each attribute where the larger score the more important the attribute. The scores suggest at the importance of plas, age and mass.

Summary

In this post you discovered feature selection for preparing machine learning data in Python with scikit-learn.

You learned about 4 different automatic feature selection techniques:

Univariate Selection. Recursive Feature Elimination. Principle Component Analysis. Feature Importance. If you are looking for more information on feature selection, see these related posts:

Feature Selection with the Caret R Package Feature Selection to Improve Accuracy and Decrease Training Time An Introduction to Feature Selection Feature Selection in Python with Scikit-Learn Do you have any questions about feature selection or this post? Ask your questions in the comment and I will do my best to answer them.

Feature importance

This notebook is part of the following blogpost: https://medium.com/bigdatarepublic/feature-importance-whats-in-a-name-79532e59eea3

Install depencencies

```
#Install non-standard packages (assuming jupyter notebook)
!pip install shap
!pip install lime
!pip install eli5
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 KB 4.4 MB/s eta 0:00:
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages

Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.9/
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.9/
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/l
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.9/dist
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dis
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.9
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.9/d
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/di
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/py
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/di
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/d
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packa
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) ... done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283859 s
  Stored in directory: /root/.cache/pip/wheels/ed/d7/c9/5a0130d06d6310bc6cbe55
Successfully built lime
```

```
Successfully built lime
Installing collected packages: lime
Successfully installed lime-0.2.0.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting eli5
  Downloading eli5-0.13.0.tar.gz (216 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.2/216.2 KB 4.6 MB/s eta 0:00:
  Preparing metadata (setup.py) ... done
Requirement already satisfied: attrs>17.1.0 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: jinja2>=3.0.0 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.9/
Requirement already satisfied: graphviz in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.9/dis
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dis
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-
Building wheels for collected packages: eli5
  Building wheel for eli5 (setup.py) ... done
  Created wheel for eli5: filename=eli5-0.13.0-py2.py3-none-any.whl size=10774
  Stored in directory: /root/.cache/pip/wheels/7b/26/a5/8460416695a992a2966b41
Successfully built eli5
Installing collected packages: eli5
```

Import packages

```
#Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import itertools

#scikit-learn package (https://pypi.org/project/scikit-learn)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc, roc_auc
from sklearn.metrics import make_scorer

#eli5 package (https://eli5.readthedocs.io/en/latest)
import eli5
from eli5.sklearn import PermutationImportance

#lime package (https://github.com/marcotcr/lime)
import lime
import lime.lime_tabular

#shap package (https://github.com/slundberg/shap)
import shap
```

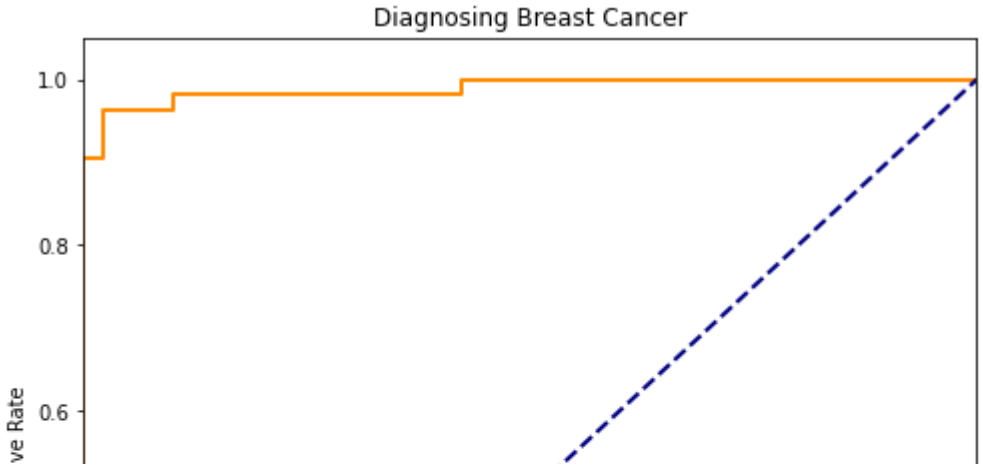
Load data and train model

```
#Load Wisconsin diagnostic breast cancer data from UCI #

url_prefix = "https://archive.ics.uci.edu/ml/machine-learning-databases/"
#if url url does not work you can use this mirror instead:
#url_prefix = "http://mlr.cs.umass.edu/ml/machine-learning-databases/"

data_url = url_prefix + "breast-cancer-wisconsin/wdbc.data"

#Define column labels
id_status = ["ID", "diagnosis"]

#Mean, standard error and maximum ('worst') values are available for features
#computed on a collection of cells in a tissue
column_labels = itertools.product(["radius", "texture", "perimeter", "area",
                                   "smoothness", "compactness", "concavity",
                                   "concave_points", "symmetry", "fractal_dim"],
                                  ["mean","std","max"])

column_labels = id_status + [f"{t}_{f}" for f, t in column_labels]

#Read into pandas DataFrame
df = pd.read_csv(data_url, header=None, names = column_labels)

#Define feature set
X = df.drop('ID', axis=1).drop('diagnosis', axis=1)

#Define diagnosis as integer: malignant (1) or benign (0)
y = (df['diagnosis'] == "M")*1

#Split train and test set.
RANDOM_STATE = 123
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    stratify=y,
                                                    random_state=RANDOM_STATE)

#Train random forest classification model
model = RandomForestClassifier(max_depth=4, random_state=RANDOM_STATE)
model.fit(X_train, y_train)

# Diagnosis prediction
y_predict = model.predict(X_test)

# Probability of malignant tissue produced by the model
y_prob = [probs[1] for probs in model.predict_proba(X_test)]
```

Evaluate model

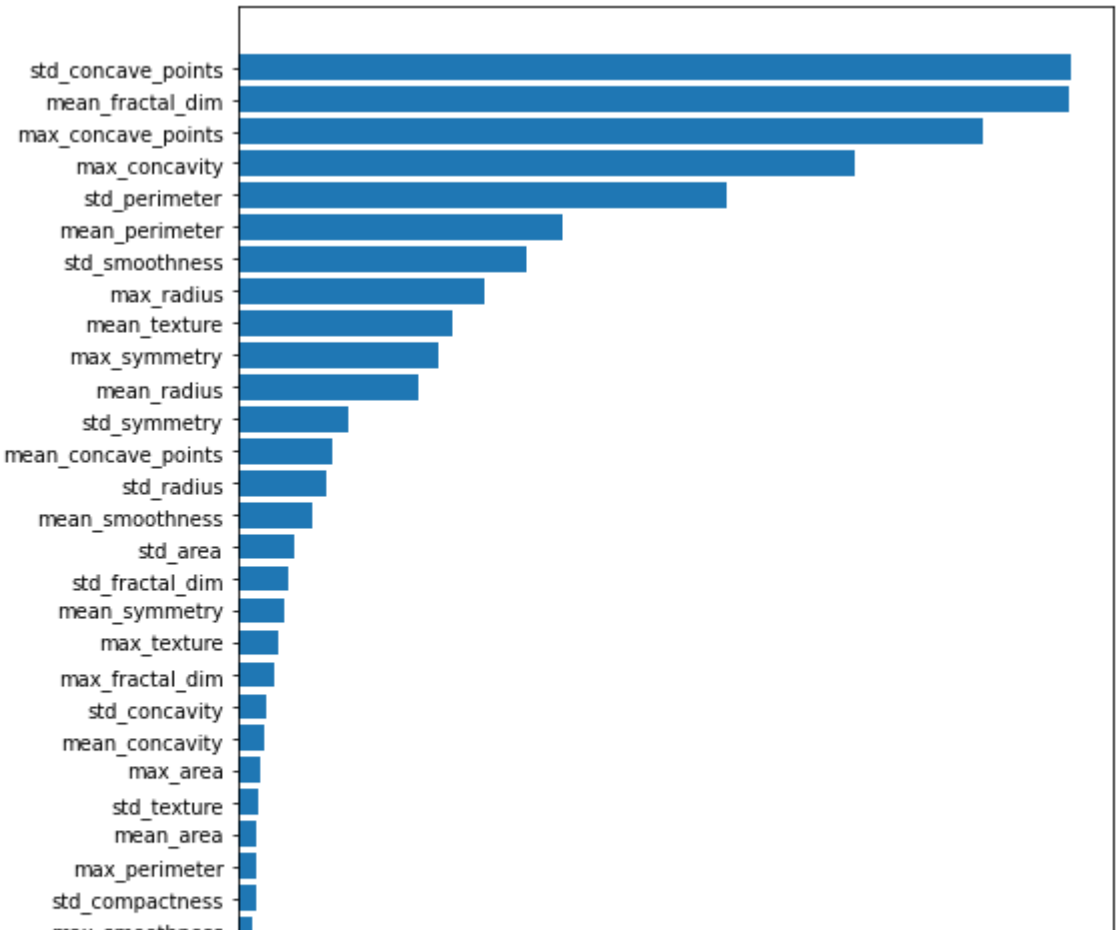
```
#Accuracy on test set
print(f"Test accuracy: {accuracy_score(y_test, y_predict).round(2)}")

# Confusion matrix test set
pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted Benign', 'Predicted Malignant'],
    index=['Benign', 'Malignant']
)

# Compute area under the curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

#Set default figure size
plt.rcParams['figure.figsize'] = (8,8)

# Plot ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("Diagnosing Breast Cancer")
plt.legend(loc="lower right")
plt.show()
```

```
Test accuracy: 0.97
```



Model-specific feature importance

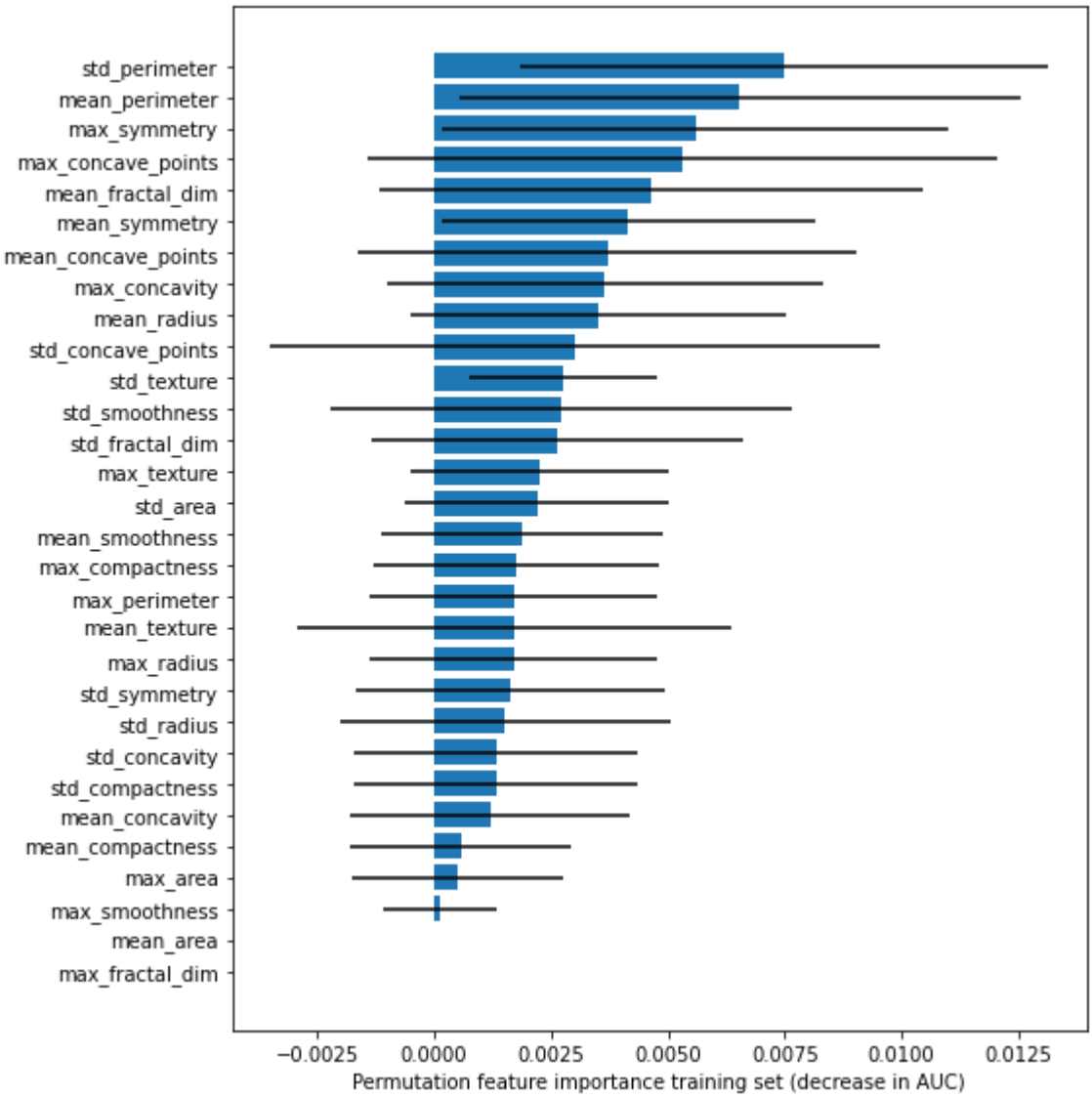
```
# Feature importance dataframe
imp_df = pd.DataFrame({'feature': X_train.columns.values,
                       'importance': model.feature_importances_})

# Reorder by importance
ordered_df = imp_df.sort_values(by='importance')
imp_range=range(1,len(imp_df.index)+1)

## Barplot with confidence intervals
height = ordered_df['importance']
bars = ordered_df['feature']
y_pos = np.arange(len(bars))

# Create horizontal bars
plt.barh(y_pos, height)

# Create names on the y-axis
plt.yticks(y_pos, bars)

plt.xlabel("Mean reduction in tree impurity in random forest")

plt.tight_layout()
# Show graphic
plt.show()
```



Permutation feature importance

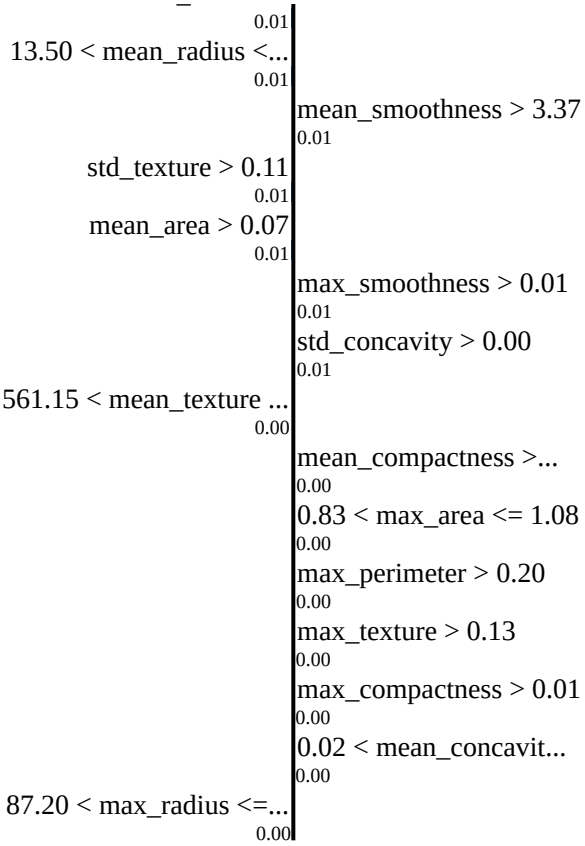

```
# Feature importance based on TRAINING set

perm_test = PermutationImportance(model, scoring=make_scorer(roc_auc_score),
                                  n_iter=50, random_state=RANDOM_STATE, cv="prefit")

# fit and see the permuation importances
perm_test.fit(X_train, y_train)

imp_df = eli5.explain_weights_df(perm_test)
label_df = pd.DataFrame({'feature': [ "x" + str(i) for i in range(len(X_test.columns)
imp_df = pd.merge(label_df, imp_df, on='feature', how='inner', validate="one_to_one")

# Reorder by importance
ordered_df = imp_df.sort_values(by='weight')
imp_range=range(1,len(imp_df.index)+1)


## Barplot with confidence intervals

height = ordered_df['weight']
bars = ordered_df['feature_name']
ci = 1.96 * ordered_df['std']
```

```
y_pos = np.arange(len(bars))

# Create horizontal bars
plt.barh(y_pos, height, xerr=ci)

# Create names on the y-axis
plt.yticks(y_pos, bars)

plt.xlabel("Permutation feature importance training set (decrease in AUC)")
plt.tight_layout()

# Show graphic
plt.show()
```

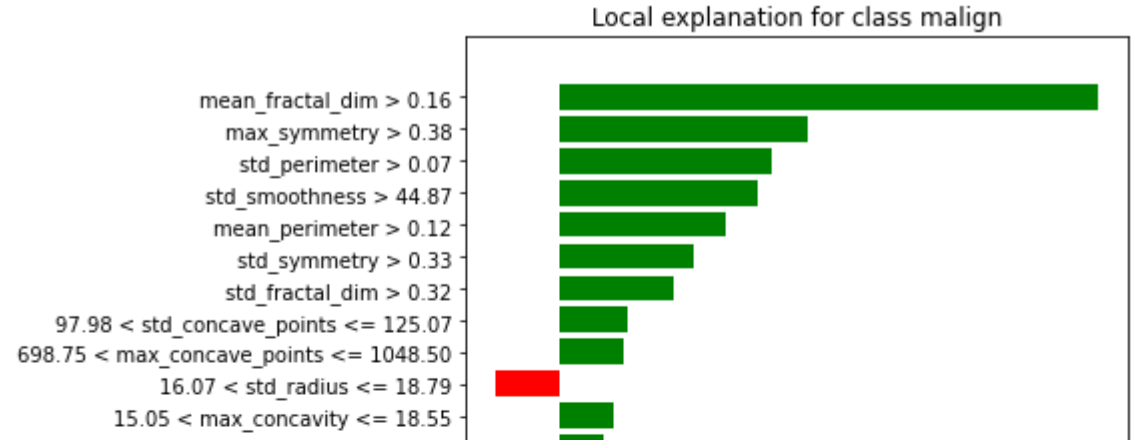


LIME

```
#Explain samples in test set
X_explain = X_test
explainer = lime.lime_tabular.LimeTabularExplainer(training_data=X_train.values,
                                                   feature_names=X_train.columns.valu
                                                   discretize_continuous=True,
                                                   class_names=["benign", "malign"],
                                                   mode="classification",
                                                   verbose=True,
                                                   random_state=RANDOM_STATE)

#Explaining first subject in test set using all 30 features
exp = explainer.explain_instance(X_explain.values[0,:],model.predict_proba,
                                 num_features=30)
#Plot local explanation
plt = exp.as_pyplot_figure()
plt.tight_layout()
exp.show_in_notebook(show_table=True)
```

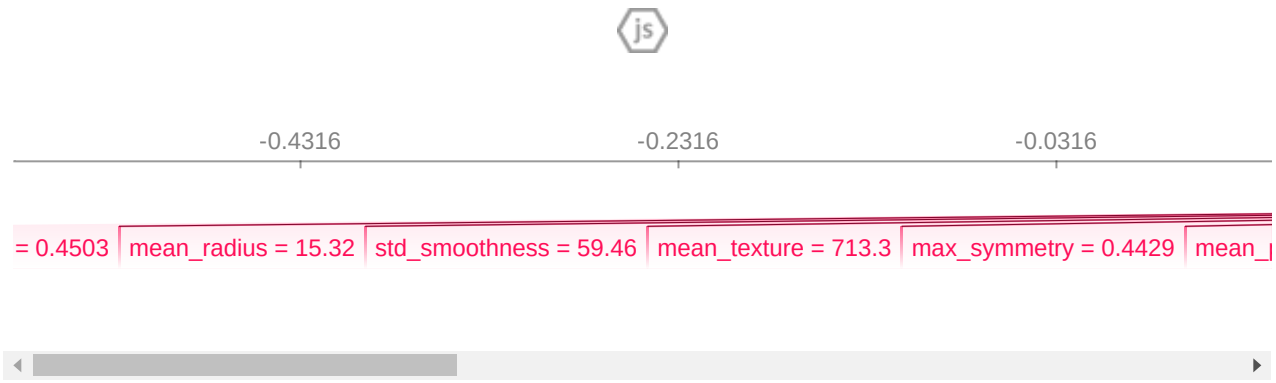


Feature	Value
mean_fractal_dim	0.22
max_symmetry	0.44
std_perimeter	0.12
std_smoothness	59.46
mean_perimeter	0.24
std_symmetry	0.45
std_fractal_dim	0.33
std_concave_points	119.80
max_concave_points	928.80
std_radius	17.27
max_concavity	17.73
std_area	0.66
std_compactness	0.05

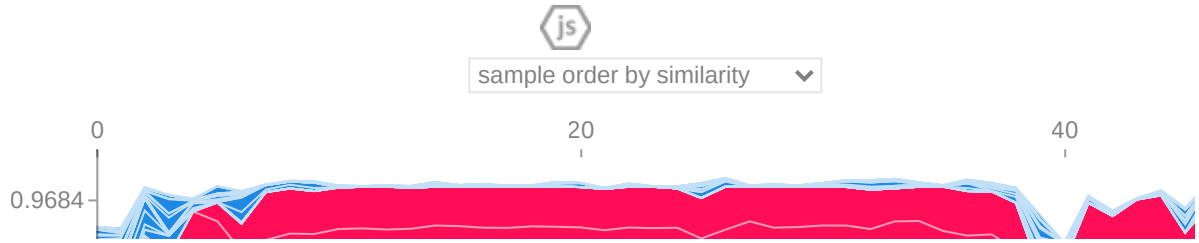



```
# explain the model's predictions on test set using SHAP values
```

```
# same syntax works for xgboost, LightGBM, CatBoost, and some scikit-learn models
explainer = shap.TreeExplainer(model)

# shap_values consists of a list of two matrices of dimension samplesize x #features
# The first matrix uses average nr of benign samples as base value
# The second matrix which is used below uses average nr of malignant samples as base
shap_values = explainer.shap_values(X_explain)


# Interactive visualization of the explanation of the first subject
# in the test set (X_explain).
# It shows the relative contribution of features to get from the base value of malign
# samples(average value)
# to the output value (1 in case of malignant sample)
# the numbers at the bottom show the actual values for this sample.
shap.initjs() #initialize javascript in cell
shap.force_plot(explainer.expected_value[1], shap_values[1][0,:], X_explain.iloc[0,:]
```

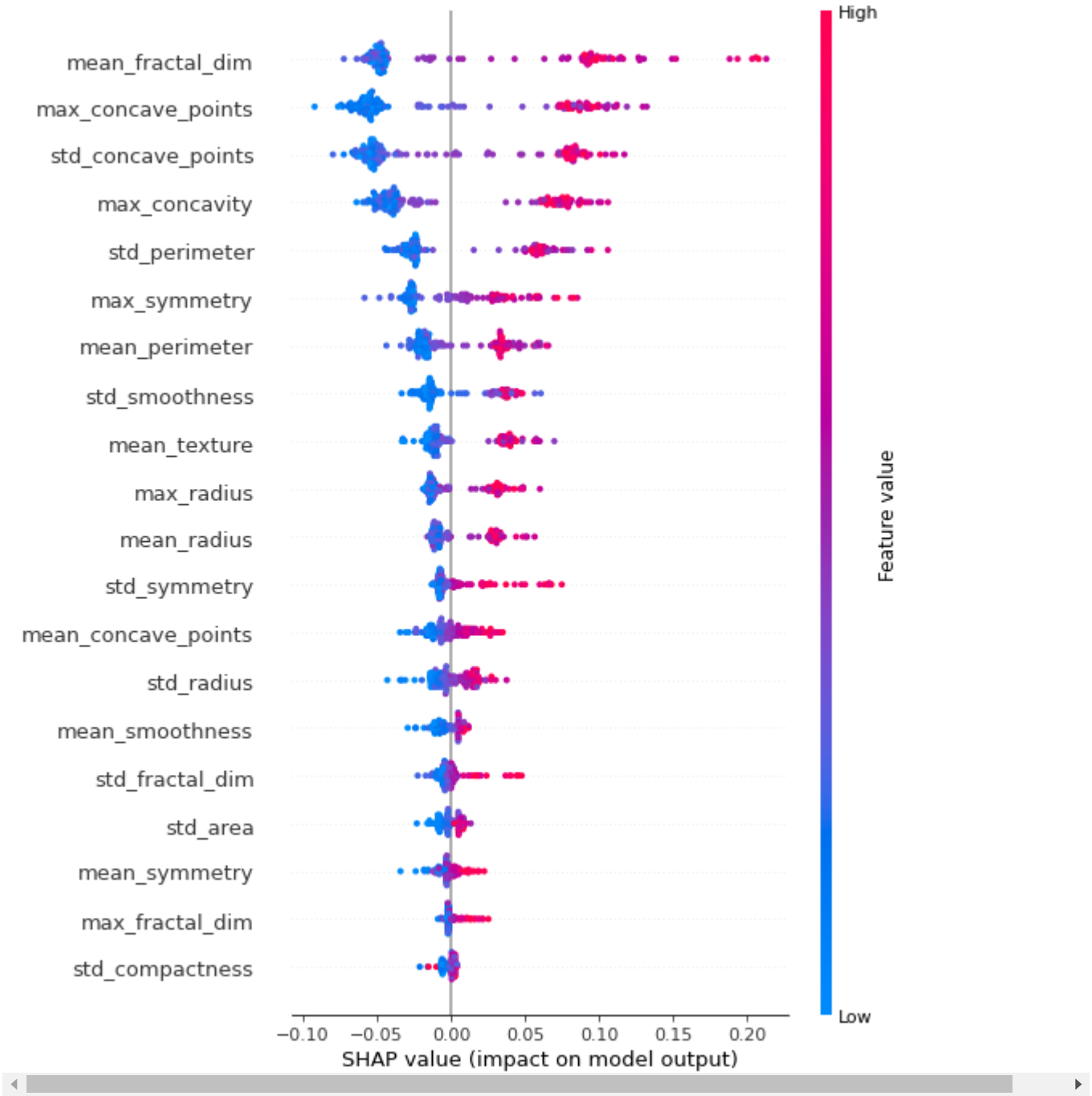



```
#Interactive visualization of all sample/feature Shapley values
#It is possible to show the relative contribution of individual features for all
# samples on the y-axis as well.
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], X_explain)
```



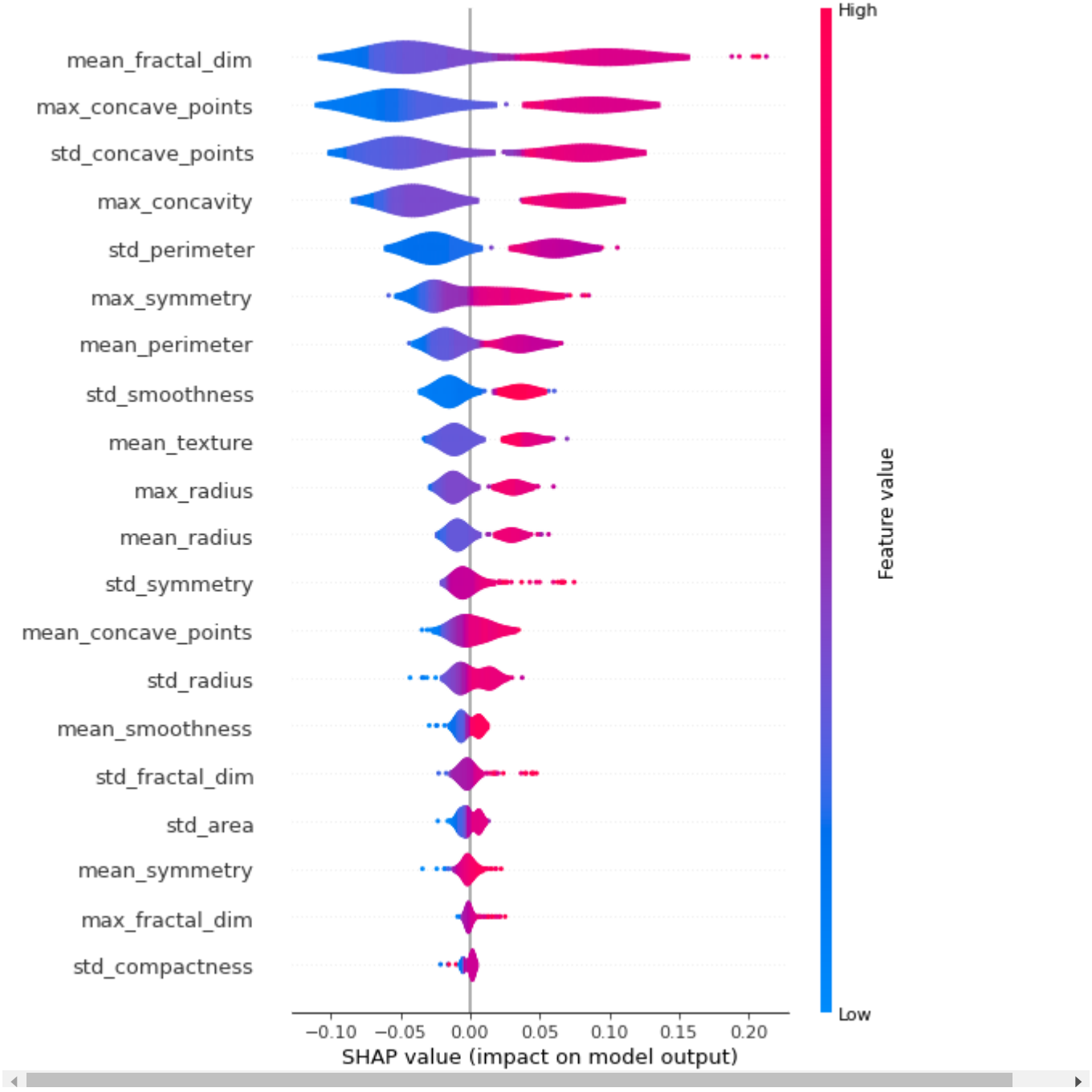

```
#A summary plot with the shapley value (feature importance)
shap.summary_plot(shap_values[1], X_explain)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ign
```




```
#Same as above, but with violin plots to better see the distribution of shapley value
shap.summary_plot(shap_values[1], X_explain, plot_type="violin")
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ign
```



In [ ]:
```
#Requirements

!pip install numpy
!pip install pandas
!pip install sklearn
!pip install seaborn
!pip install rfpimp
!pip install eli5
!pip install xgboost
!pip install matplotlib
```

```
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: numpy in /home/deepcode/.local/lib/python3.
9/site-packages (1.24.2)
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: pandas in /home/deepcode/.local/lib/python3.
9/site-packages (1.5.3)
Requirement already satisfied: numpy>=1.20.3 in /home/deepcode/.local/lib/p
ython3.9/site-packages (from pandas) (1.24.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/deepcode/.lo
cal/lib/python3.9/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /home/deepcode/.local/lib/py
thon3.9/site-packages (from pandas) (2022.7.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages
(from python-dateutil>=2.8.1->pandas) (1.16.0)
Defaulting to user installation because normal site-packages is not writeab
le
Collecting sklearn
  Using cached sklearn-0.0.post1.tar.gz (3.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0.post1-py3-none-any.whl si
ze=2342 sha256=c32631bb64cc7e59f09b292901f6c16e70cbdd4f2b6c9030d874059649a0
9f22
  Stored in directory: /home/deepcode/.cache/pip/wheels/f8/e0/3d/9d0c2020c4
4a519b9f02ab4fa6d2a4a996c98d79ab2f569fa1
Successfully built sklearn
Installing collected packages: sklearn
Successfully installed sklearn-0.0.post1
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: seaborn in /home/deepcode/.local/lib/python
3.9/site-packages (0.12.2)
Requirement already satisfied: pandas>=0.25 in /home/deepcode/.local/lib/py
thon3.9/site-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/lib/python3/
dist-packages (from seaborn) (3.3.4)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /home/deepcode/.loca
l/lib/python3.9/site-packages (from seaborn) (1.24.2)
Requirement already satisfied: pytz>=2020.1 in /home/deepcode/.local/lib/py
thon3.9/site-packages (from pandas>=0.25->seaborn) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/deepcode/.lo
cal/lib/python3.9/site-packages (from pandas>=0.25->seaborn) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages
(from python-dateutil>=2.8.1->pandas>=0.25->seaborn) (1.16.0)
Defaulting to user installation because normal site-packages is not writeab
le
Collecting rfpimp
  Downloading rfpimp-1.3.7.tar.gz (10 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/lib/python3/dist-packages
(from rfpimp) (3.3.4)
Requirement already satisfied: numpy in /home/deepcode/.local/lib/python3.
9/site-packages (from rfpimp) (1.24.2)
```

```
Requirement already satisfied: pandas in /home/deepcode/.local/lib/python3.
9/site-packages (from rfpimp) (1.5.3)
Requirement already satisfied: scikit-learn in /home/deepcode/.local/lib/py
thon3.9/site-packages (from rfpimp) (1.2.2)
Requirement already satisfied: pytz>=2020.1 in /home/deepcode/.local/lib/py
thon3.9/site-packages (from pandas->rfpimp) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/deepcode/.lo
cal/lib/python3.9/site-packages (from pandas->rfpimp) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/deepcode/.loca
l/lib/python3.9/site-packages (from scikit-learn->rfpimp) (3.1.0)
Requirement already satisfied: joblib>=1.1.1 in /home/deepcode/.local/lib/p
ython3.9/site-packages (from scikit-learn->rfpimp) (1.2.0)
Requirement already satisfied: scipy>=1.3.2 in /usr/lib/python3/dist-packag
es (from scikit-learn->rfpimp) (1.6.0)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages
(from python-dateutil>=2.8.1->pandas->rfpimp) (1.16.0)
Building wheels for collected packages: rfpimp
  Building wheel for rfpimp (setup.py) ... done
  Created wheel for rfpimp: filename=rfpimp-1.3.7-py3-none-any.whl size=106
66 sha256=263e4b317e46897c05ee0a7bb89677d4bd6a89ce0b8f6421f38e359b0c63e72c
  Stored in directory: /home/deepcode/.cache/pip/wheels/d9/f2/53/6d8c73011f
73fc347598d683ff8b2343605ad43474ae083816
Successfully built rfpimp
Installing collected packages: rfpimp
Successfully installed rfpimp-1.3.7
Defaulting to user installation because normal site-packages is not writeab
le
Collecting eli5
  Downloading eli5-0.13.0.tar.gz (216 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.2/216.2 kB 2.1 MB/s eta 0:
00:00a 0:00:01
  Preparing metadata (setup.py) ... done
Requirement already satisfied: attrs>17.1.0 in /usr/lib/python3/dist-packag
es (from eli5) (20.3.0)
Collecting graphviz
  Downloading graphviz-0.20.1-py3-none-any.whl (47 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.0/47.0 kB 4.6 MB/s eta 0:0
0:00
Requirement already satisfied: jinja2>=3.0.0 in /home/deepcode/.local/lib/p
ython3.9/site-packages (from eli5) (3.1.2)
Requirement already satisfied: numpy>=1.9.0 in /home/deepcode/.local/lib/py
thon3.9/site-packages (from eli5) (1.24.2)
Requirement already satisfied: scikit-learn>=0.20 in /home/deepcode/.local/
lib/python3.9/site-packages (from eli5) (1.2.2)
Requirement already satisfied: scipy in /usr/lib/python3/dist-packages (fro
m eli5) (1.6.0)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from
 eli5) (1.16.0)
Requirement already satisfied: tabulate>=0.7.7 in /usr/lib/python3/dist-pac
kages (from eli5) (0.8.7)
Requirement already satisfied: MarkupSafe>=2.0 in /home/deepcode/.local/li
b/python3.9/site-packages (from jinja2>=3.0.0->eli5) (2.1.2)
Requirement already satisfied: joblib>=1.1.1 in /home/deepcode/.local/lib/p
ython3.9/site-packages (from scikit-learn>=0.20->eli5) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/deepcode/.loca
l/lib/python3.9/site-packages (from scikit-learn>=0.20->eli5) (3.1.0)
```

```
Building wheels for collected packages: eli5
  Building wheel for eli5 (setup.py) ... done
  Created wheel for eli5: filename=eli5-0.13.0-py2.py3-none-any.whl size=10
7727 sha256=2697ee93142ad720efb5c93c959d69388971a2760199906adab4da583c92048
1
  Stored in directory: /home/deepcode/.cache/pip/wheels/7b/26/a5/8460416695
a992a2966b41caa5338e5e7fcea98c9d032d055c
Successfully built eli5
Installing collected packages: graphviz, eli5
Successfully installed eli5-0.13.0 graphviz-0.20.1
Defaulting to user installation because normal site-packages is not writeab
le
Collecting xgboost
  Downloading xgboost-1.7.4-py3-none-manylinux2014_x86_64.whl (193.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 193.6/193.6 MB 2.3 MB/s eta 0:
00:0000:0100:02
Requirement already satisfied: scipy in /usr/lib/python3/dist-packages (fro
m xgboost) (1.6.0)
Requirement already satisfied: numpy in /home/deepcode/.local/lib/python3.
9/site-packages (from xgboost) (1.24.2)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.4
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: matplotlib in /usr/lib/python3/dist-packages
(3.3.4)
```

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBClassifier
from xgboost import plot_importance
from matplotlib import pyplot as plt

np.random.seed(seed = 42)
```

In [ ]:
```
# Read CSV
task_file = 'task_data.csv'
with open(task_file, mode = 'rb') as f:
    data = pd.read_csv(f)

# Extract names of each column (using pandas)
headers = np.array(list(data.columns.values))
names = headers[2:]
y = data.class_label
X = pd.DataFrame(data,columns=names)
```

In [ ]:
```
print(headers)
```

```
['sample index' 'class_label' 'sensor0' 'sensor1' 'sensor2' 'sensor3'
 'sensor4' 'sensor5' 'sensor6' 'sensor7' 'sensor8' 'sensor9']
```

In [ ]:
```
print(names)
```

```
['sensor0' 'sensor1' 'sensor2' 'sensor3' 'sensor4' 'sensor5' 'sensor6'
 'sensor7' 'sensor8' 'sensor9']
```

In [ ]:

As per problem statement, the emphasis is on identifying the importance of a particular feature in the machine learning model. This is a classification problem wherein we need to identify the important feature and we need to justify the reasons by showing why we have chosen a specific feature and how it turns out to be important feature. Identifying the important features is very important due to the followig reasons: 1. It will help us improve the model's by concentrating on important features (variables) 2. It will help us get rid of features that are not relevant or which don not contribute the model's performance.

Feature importance can be implemented using various model's the popular among them being Random forest, XGBoost, Linear Regression. My idea here is that the better the model is, the more we can rely on the feature importance and how to interepret the problem. I will explain the reason for chosing the model as we proceed.

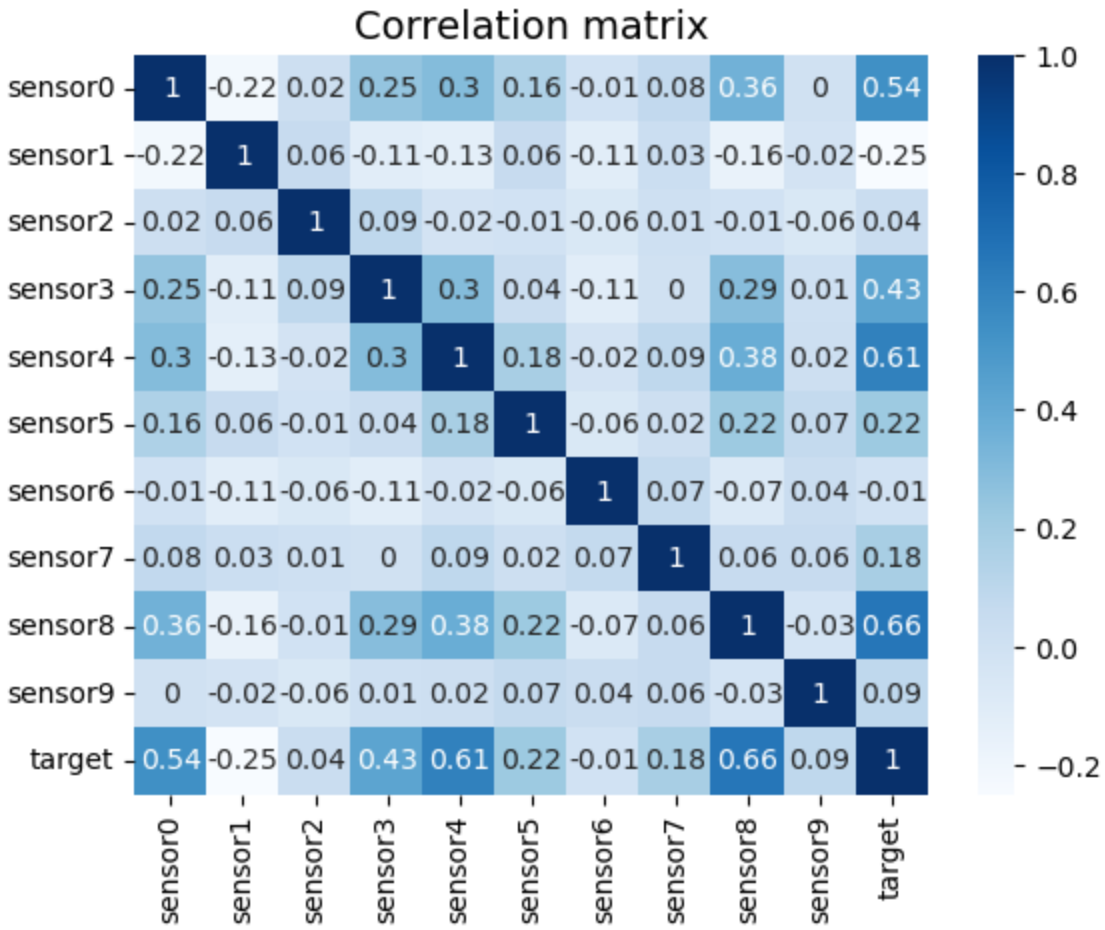
Exploratory data analysis

In [ ]:
```
print(data.describe())
```

```
       class_label     sensor0     sensor1     sensor2     sensor3  \
count   400.000000  400.000000  400.000000  400.000000  400.000000
mean      0.000000    0.523661    0.509223    0.481238    0.509752
std       1.001252    0.268194    0.276878    0.287584    0.297712
min      -1.000000    0.007775    0.003865    0.004473    0.001466
25%      -1.000000    0.299792    0.283004    0.235544    0.262697
50%       0.000000    0.534906    0.507583    0.460241    0.510066
75%       1.000000    0.751887    0.727843    0.734937    0.768975
max       1.000000    0.999476    0.998680    0.992963    0.995119

          sensor4     sensor5     sensor6     sensor7     sensor8     senso
r9
count  400.000000  400.000000  400.000000  400.000000  400.000000  400.0000
00
mean     0.497875    0.501065    0.490480    0.482372    0.482822    0.5419
33
std      0.288208    0.287634    0.289954    0.282714    0.296180    0.2724
90
min      0.000250    0.000425    0.000173    0.003322    0.003165    0.0004
52
25%      0.249369    0.269430    0.226687    0.242848    0.213626    0.3212
64
50%      0.497842    0.497108    0.477341    0.463438    0.462251    0.5783
89
75%      0.743401    0.738854    0.735304    0.732483    0.740542    0.7689
90
max      0.999412    0.997367    0.997141    0.998230    0.996098    0.9994
65
```

In [ ]:
```
sns.heatmap(X.assign(target = y).corr().round(2), cmap = 'Blues', annot = Tr
```

Out[ ]:
```
Text(0.5, 1.0, 'Correlation matrix')
```



From the correlation matrix we can note that there is no correlation between the features.

Models:

a) Random Forest

Since Decision trees can be used for better interpretability of the model, the downside is that it works well if the depth is very small. Since the current dataset has only 400 data points, it might be easy to interpret the model using Decision tree, but if the aspect of scalebility is considered it might not be a best approach and this is where Random Forest comes into picture. In Random forest the subset of data is chosen which is also called as bootstrap. These bootstrap are fed to decision trees with large depths randomly. The final result is obtained by counting the majority of the vote from all the decision trees. This technique is called as boot strapping or bagging. This also helps in overcoming bais.

In order to validate the Random forest model, I have used `oob_score`. Out of bag score(oob_score) is used when the dataset is smaller in size. Since validation score requires

us to keep apart certain part of da ta, it is good to use oob_score since it works only on subset of decision trees and while the validation score is calculated on all the decision trees.

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

rf =  RandomForestClassifier(n_estimators = 100,
                             n_jobs = -1,
                             oob_score = True,
                             bootstrap = True, random_state=42)

rf.fit(X, y)
```

Out[ ]:
```
RandomForestClassifier(n_jobs=-1, oob_score=True, random_state=42)
```

In [ ]:
```
#print('Score :', rf.score(X_test,y_test))

print('Out of bag error:', rf.oob_score_)
```

```
Out of bag error: 0.985
```

As we can obersve that the model performed well on the training set. The `oob_score` obtained is also good. I will now check which feature is more important and try to visulaize it. This will help us understand which feature has to be given more importance.

In Scikit learn, we can use the feature importance by just using the decision tree which can help us in give some prior intution of th features. Decision Tree is one of the machine learning algorithms that is used for a range of classification and regression problems and more specifically in the decision analysis problems. Decision tree provides a tree-like structure which comprises od decisions that can be made based on a hypothesis. A decision tree is represented in an upside-down where each node represents a feature also called attribute and each brach also called link to the nodes is represents a decision or condition for selecting the specific node. The top node of the tree is called the root node and the bottom-most nodes are called the leaf nodes. The main idea behind using a decision tree is to represent the whole data in the form of a tree.
It has some advantages such as it is faster and it is easier to implement. The downside is it suffers from bias problem due to high cardinality and inflate the importance of continous featues.

In [ ]:
```
# function for creating a feature importance dataframe
def imp_df(column_names, importances):
    df = pd.DataFrame({'feature': column_names,
                       'feature_importance': importances}) \
           .sort_values('feature_importance', ascending = False) \
           .reset_index(drop = True)
    return df
```

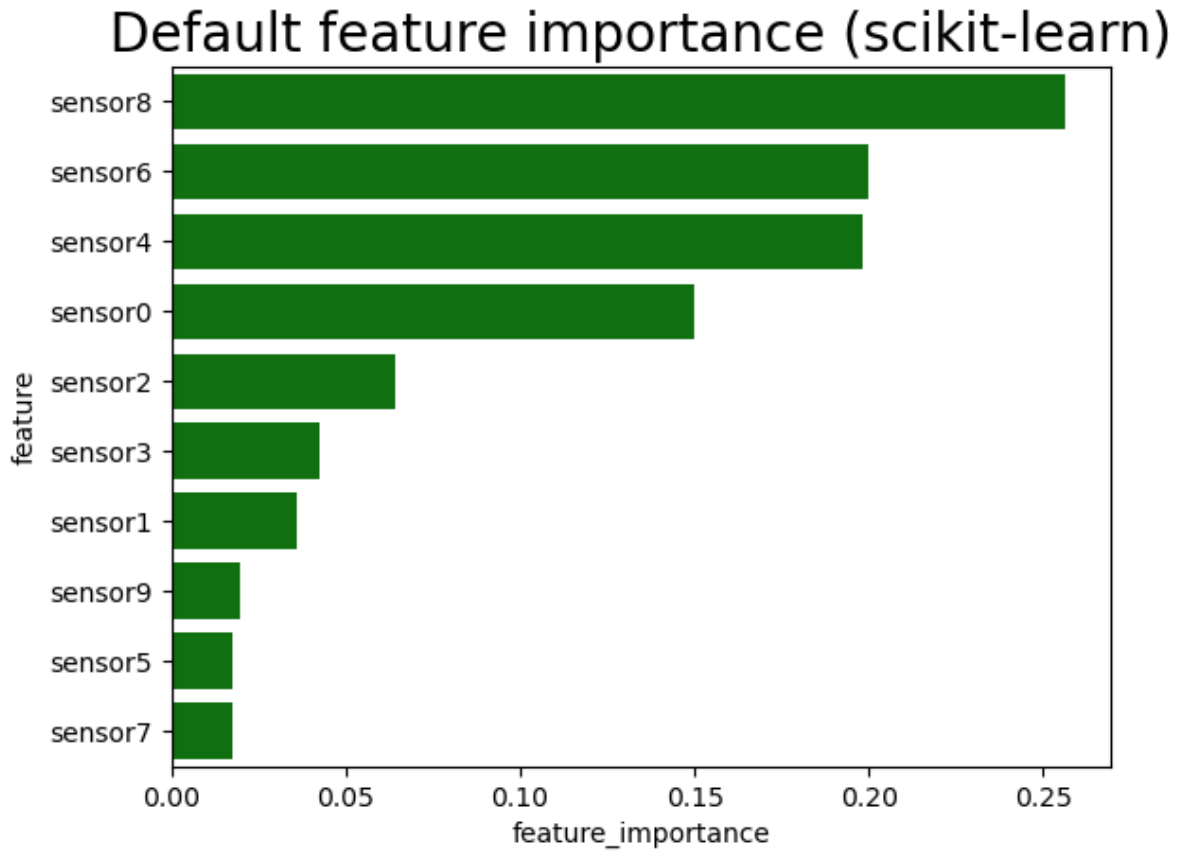
```
# plotting a feature importance dataframe (horizontal barchart)
def var_imp_plot(imp_df, title):
    imp_df.columns = ['feature', 'feature_importance']
    sns.barplot(x = 'feature_importance', y = 'feature', data = imp_df, orie
       .set_title(title, fontsize = 20)
```

In [ ]:
```
base_imp = imp_df(X.columns, rf.feature_importances_)
base_imp
```

Out[ ]:

	feature	feature_importance
0	sensor8	0.256427
1	sensor6	0.199622
2	sensor4	0.198249
3	sensor0	0.149611
4	sensor2	0.063839
5	sensor3	0.042247
6	sensor1	0.036030
7	sensor9	0.019462
8	sensor5	0.017295
9	sensor7	0.017217

In [ ]:
```
var_imp_plot(base_imp, 'Default feature importance (scikit-learn)')
```



Using the default feature importance of Scikit-learn we can get the below mentioned graph. As we can notice that the top features are from `Sensor8, Sensor4 and Sensor6` .

[Reference : https://scikit-learn.org/stable/modules/feature_selection.html]

This default feature importance doesn't gives a proper picture of feature importance. This is because the there is bias involved in this method .[Reference https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#varimp]. Another issue is the collinearity and the inflation towards the continous values.

Hence we need to find a better method.

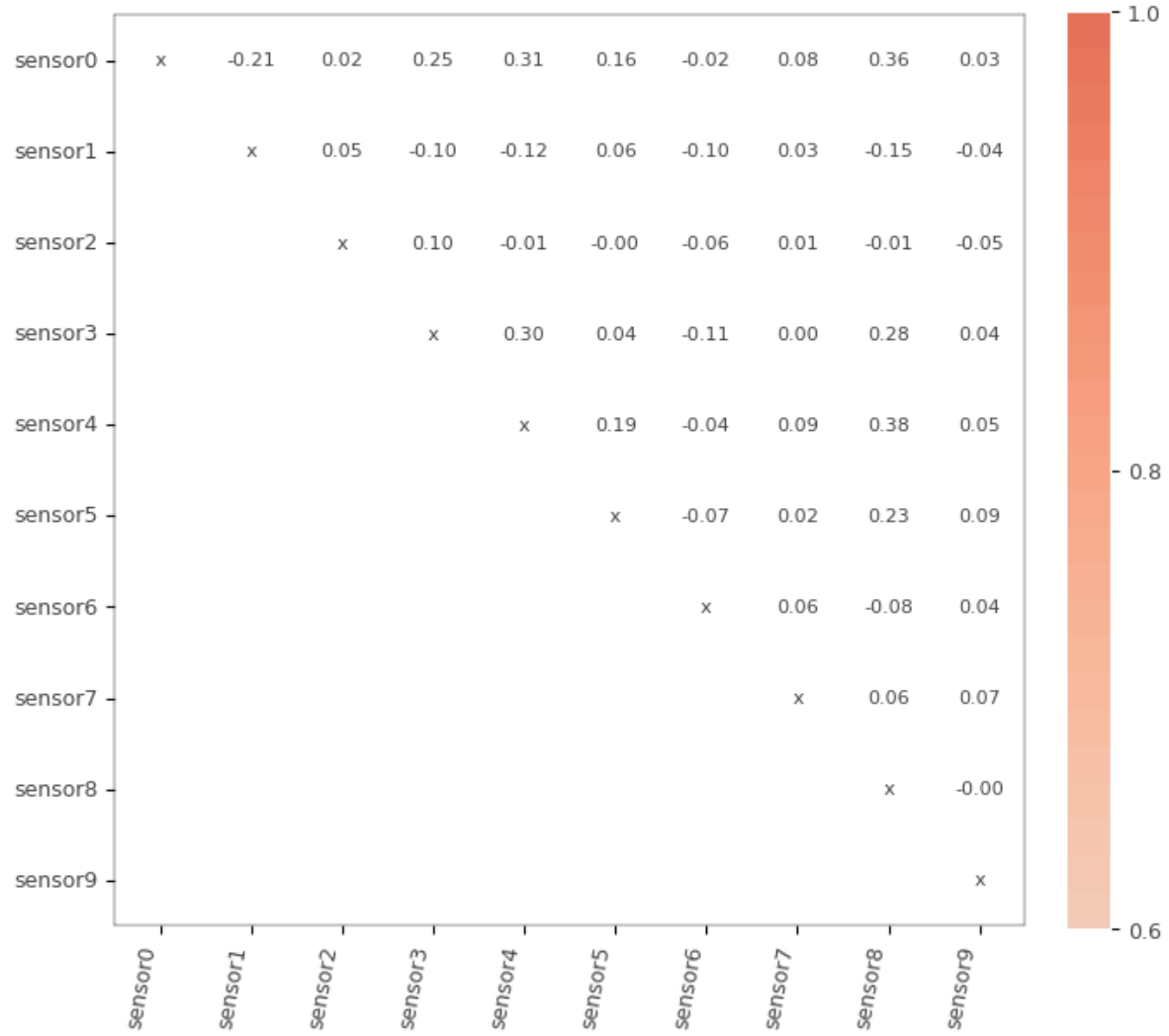
Other method to check the feature importance.

There are other ways to check the feature importance and I have chosen the following:

1. Permutation feature importance :

```
    In this method we first train the model and calculate the
score by passing validation set (or OOB set in case of Random
Forest). We call this score as benchmark result. We then re
shuffle the values from one feature in the dataset and then
pass it to the model to calcualte the score again. We then
calculate the feature importance score which the the
difference between the score obtained after shuffling the
data and benchmark result. We repeat this process for all the
features.
    This method has advantages like it is applicable to any
model, reasonably efficient, reliable technique and no need
to retrain the model at each modification of the dataset. The
downside is it is more computationally expensive than default
feature_importances permutation importance overestimates the
importance of correlated predictors - Strobl et al (2008)


    [ Reference :https://explained.ai/rf-
importance/index.html#5 ]
```

In [ ]:
```
from rfpimp import plot_corr_heatmap
viz = plot_corr_heatmap(X, figsize=(7,7))
viz.view()
```



In [ ]:
```
#Referring the code from https://explained.ai/rf-importance/index.html#5
from sklearn.metrics import r2_score
from rfpimp import permutation_importances

def r2(rf, X, y):
    return r2_score(y, rf.predict(X))

perm_imp_rfpimp = permutation_importances(rf, X, y, r2)
perm_imp_rfpimp.reset_index(drop = False, inplace = True)
```

In [ ]:
```
#Referring the code from https://explained.ai/rf-importance/index.html#5

from rfpimp import *
rf_pimp = RandomForestClassifier(n_estimators=100, n_jobs=-1)
%time I = oob_importances(rf_pimp, X, y)
```

```
CPU times: user 504 ms, sys: 28.2 ms, total: 532 ms
Wall time: 482 ms
```

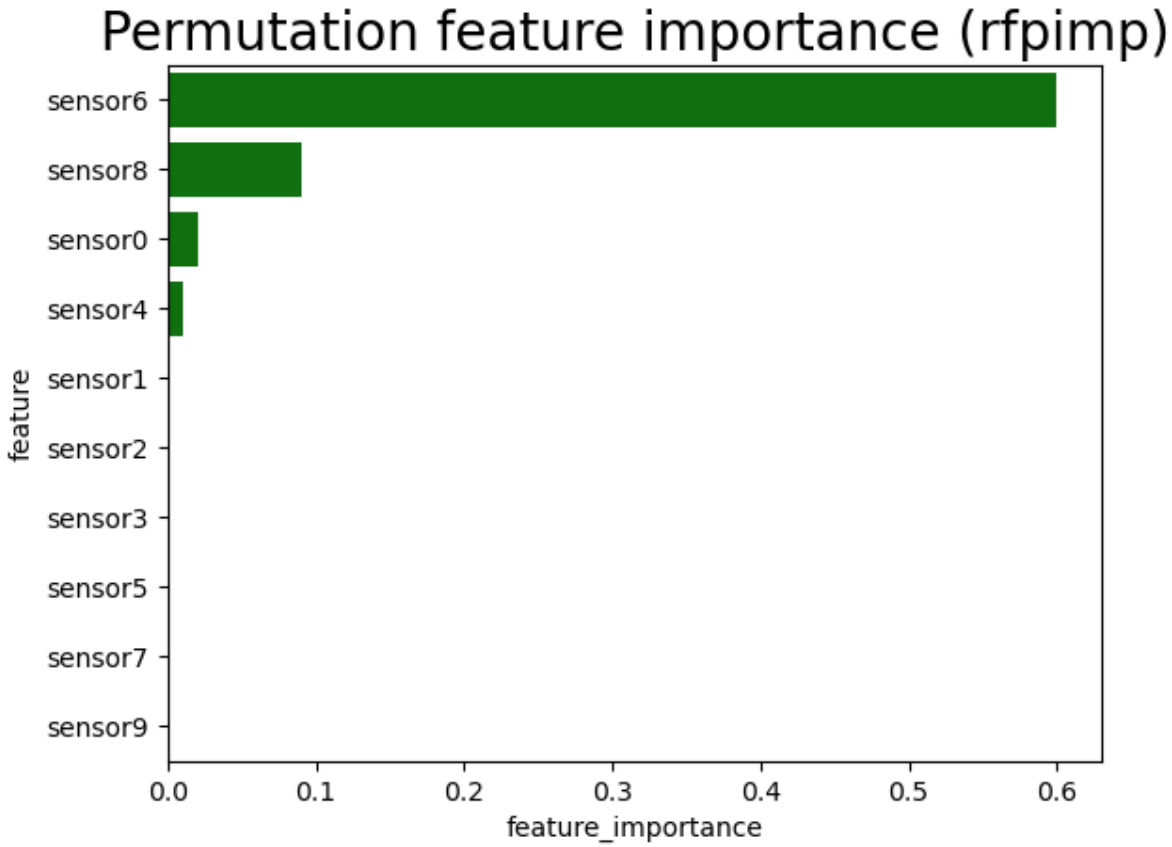
In [ ]:
```
%time I = cv_importances(rf_pimp, X, y, k=5)
```

```
CPU times: user 4.42 s, sys: 414 ms, total: 4.83 s
Wall time: 6.28 s
```

In [ ]:
```
%time I = dropcol_importances(rf_pimp, X, y)
```

```
CPU times: user 1.92 s, sys: 203 ms, total: 2.13 s
Wall time: 1.58 s
```

In [ ]:
```
var_imp_plot(perm_imp_rfpimp, 'Permutation feature importance (rfpimp)')
```



As we can notice that the permutation method gives a different result when compared to the Scikit learn's default feature importance. In this result the `sensor 6` turned out to be more important. Even though `sensor 8` is still an important feature and is among the top 4 values, this helped us to overcome the issue of collinearity which the default feature importance method suffer's with

As we can notice that the time take to get the feature importance using permutation method is much less. This will be beneficial when the dataset is very small as in this case. However this method will suffer badly when compared to default feature importance method when the size of the data increases. Hence its not scalable.

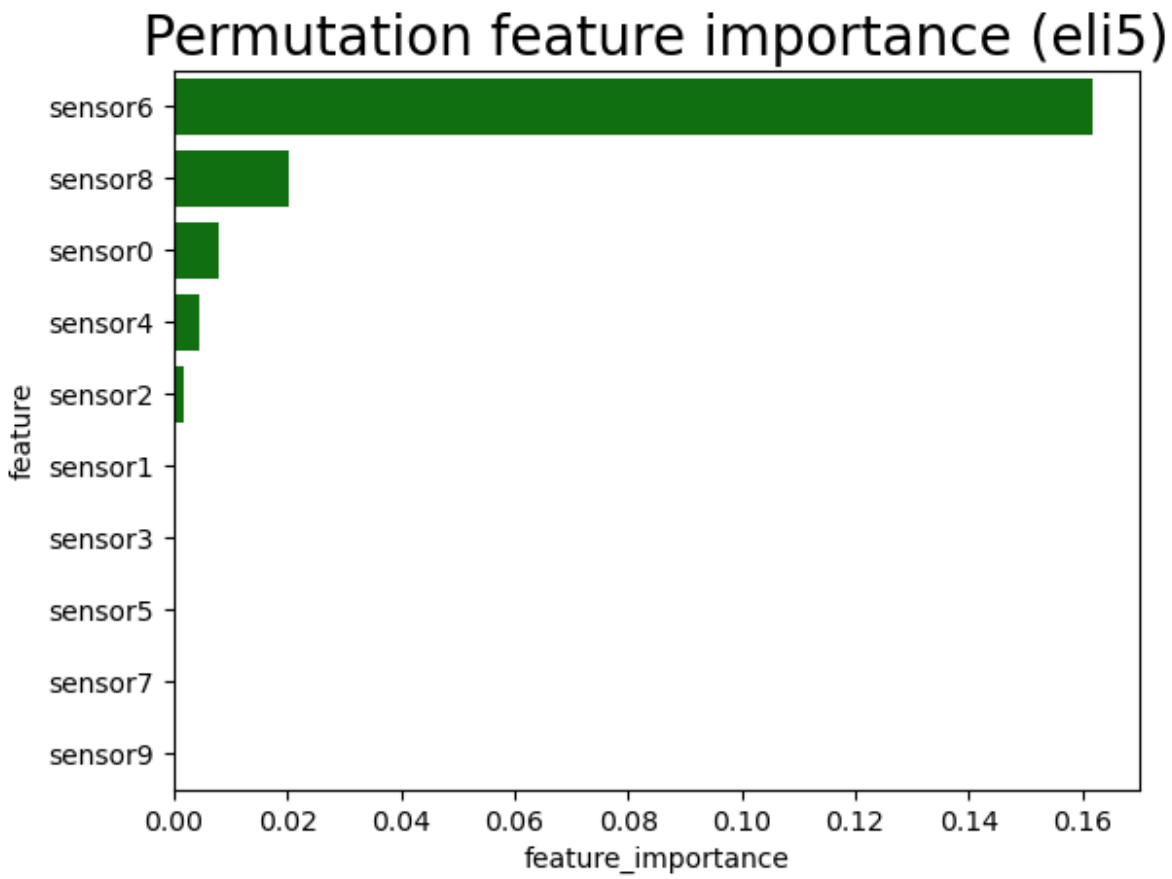
Another feature importance method is using eli5 which is another option present in scikit learn. [Reference : http://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html]

This method shows us a way where in we can check how the score will reduce if a particular feature is not present. Since the number of features present is not more for this problem, this

method is not power intensive(resource consumption), otherwise it might suffer if the number of features increase.

In [ ]:
```
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rf, cv = None, refit = False, n_iter = 50).fit(
perm_imp_eli5 = imp_df(X.columns, perm.feature_importances_)
var_imp_plot(perm_imp_eli5, 'Permutation feature importance (eli5)')
```

```
2023-03-19 15:25:08.935289: I tensorflow/core/platform/cpu_feature_guard.c
c:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-03-19 15:25:09.088763: W tensorflow/compiler/xla/stream_executor/platf
orm/default/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.
11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such
file or directory
2023-03-19 15:25:09.088784: I tensorflow/compiler/xla/stream_executor/cuda/
cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set
up on your machine.
2023-03-19 15:25:09.721479: W tensorflow/compiler/xla/stream_executor/platf
orm/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.s
o.7'; dlerror: libnvinfer.so.7: cannot open shared object file: No such fil
e or directory
2023-03-19 15:25:09.721543: W tensorflow/compiler/xla/stream_executor/platf
orm/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_pl
ugin.so.7'; dlerror: libnvinfer_plugin.so.7: cannot open shared object fil
e: No such file or directory
2023-03-19 15:25:09.721550: W tensorflow/compiler/tf2tensorrt/utils/py_util
s.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you woul
d like to use Nvidia GPU with TensorRT, please make sure the missing librar
ies mentioned above are installed properly.
```



In [ ]:
```
eli5.show_weights(perm)
```

Out[ ]:

Weight	Feature
0.1619 ± 0.0242	x6
0.0203 ± 0.0100	x8
0.0078 ± 0.0054	x0
0.0046 ± 0.0047	x4
0.0016 ± 0.0034	x2
0 ± 0.0000	x9
0 ± 0.0000	x7
0 ± 0.0000	x5
0 ± 0.0000	x3
0 ± 0.0000	x1

As we can notice that by using this method we got the similar result as we obtained via rfpimp. This gives us a better undertanding that the `sensor 6` is an important feature. It has more weightage.

There are other feature selection methods mentioned in Scikit learn such as `lasso` but I haven't used them.

[Reference : https://scikit-learn.org/stable/auto_examples/feature_selection/plot_select_from_model_boston.html#sphx-glr-auto-examples-feature-selection-plot-select-from-model-boston-py]

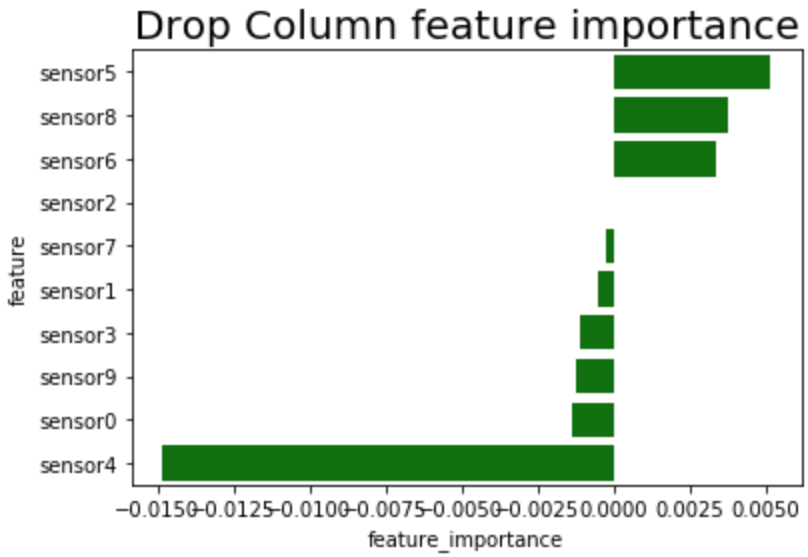
2. Check the feature importance by dropping a column

I will now check how the model behaves if we drop certain coloumn features for training. This helps us justify our previous benchamarks and results that we have obtained. We will check if the `sensor6` feature turns out to be the important one which we have see above.

In [ ]:
```
from sklearn.base import clone

def drop_col_feat_imp(model, X_train, y_train, random_state = 42):

    # clone the model to have the exact same specification as the one initia
    model_clone = clone(model)
    # set random_state for comparability
    model_clone.random_state = random_state
    # training and scoring the benchmark model
    model_clone.fit(X_train, y_train)
    benchmark_score = model_clone.score(X_train, y_train)
    # list for storing feature importances
    importances = []

    # iterating over all columns and storing feature importance (difference
    for col in X_train.columns:
        model_clone = clone(model)
        model_clone.random_state = random_state
        model_clone.fit(X_train.drop(col, axis = 1), y_train)
        drop_col_score = model_clone.score(X_train.drop(col, axis = 1), y_tr
        importances.append(benchmark_score - drop_col_score)

    importances_df = imp_df(X_train.columns, importances)
    return importances_df
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.8, r

rf_1 =  RandomForestRegressor(n_estimators = 100,
                              n_jobs = -1,
                              oob_score = True,
                              bootstrap = True, random_state=42)

rf_1.fit(X_train, y_train)
%time drop_imp = drop_col_feat_imp(rf_1, X_train, y_train)
print(drop_imp)
var_imp_plot(drop_imp, 'Drop Column feature importance')
```

```
Wall time: 2.69 s
   feature  feature_importance
0  sensor5            0.005149
1  sensor8            0.003756
2  sensor6            0.003359
3  sensor2           -0.000010
4  sensor7           -0.000256
5  sensor1           -0.000508
6  sensor3           -0.001101
7  sensor9           -0.001242
8  sensor0           -0.001343
9  sensor4           -0.014849
```



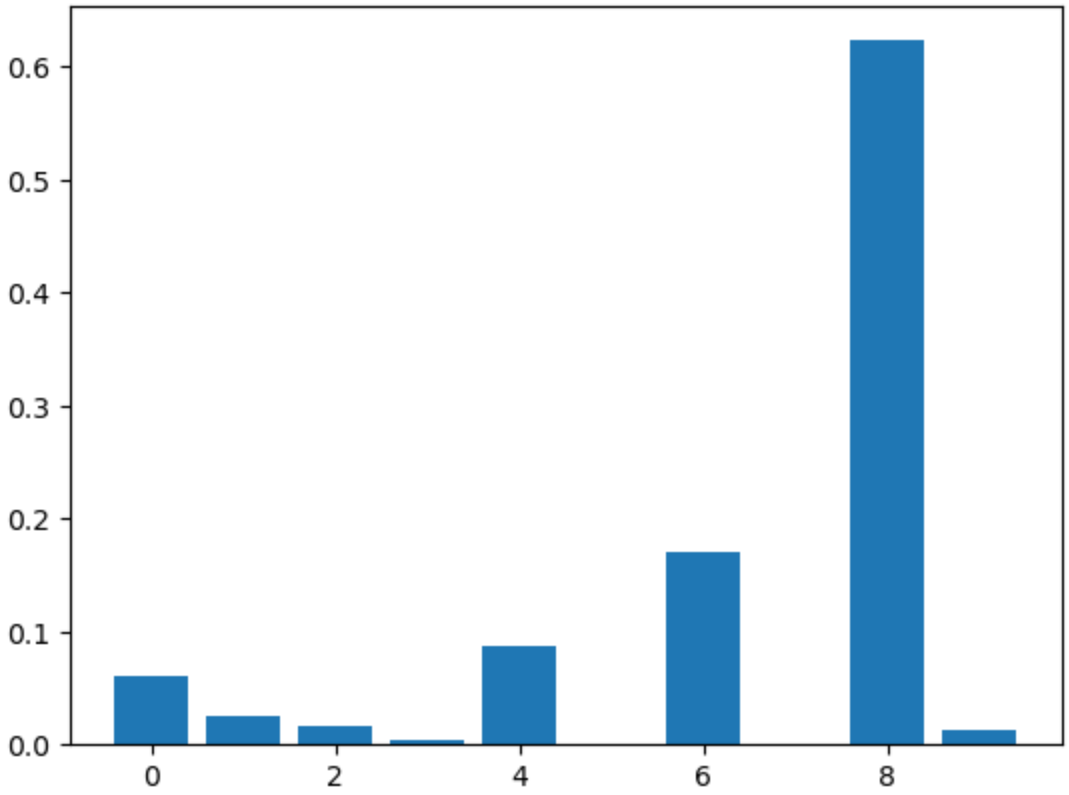
Now as we can see that certain features if dropped causes high impact both positively and negatively. As we can see sensors `8,5,6` and sensors `4,0,9` causes high impact on the overall model performance. This is one of the most important and accuracte feature selection method. The negative importance in this case means the model performance will improve if we remove these features. Another surprising thing to note is that `sensor 4` which was one of the important features in observed previously, turns out to be a negative impacting feature.

This method has one downfall which is comptutation cost. The reason for chosing random forest regressor in this method was that I wanted to see how dropping each feature causes the impact on the model. If we use the random forest classification, it will simply classify either 1 or -1 and we cannot derive any meaning out of it.

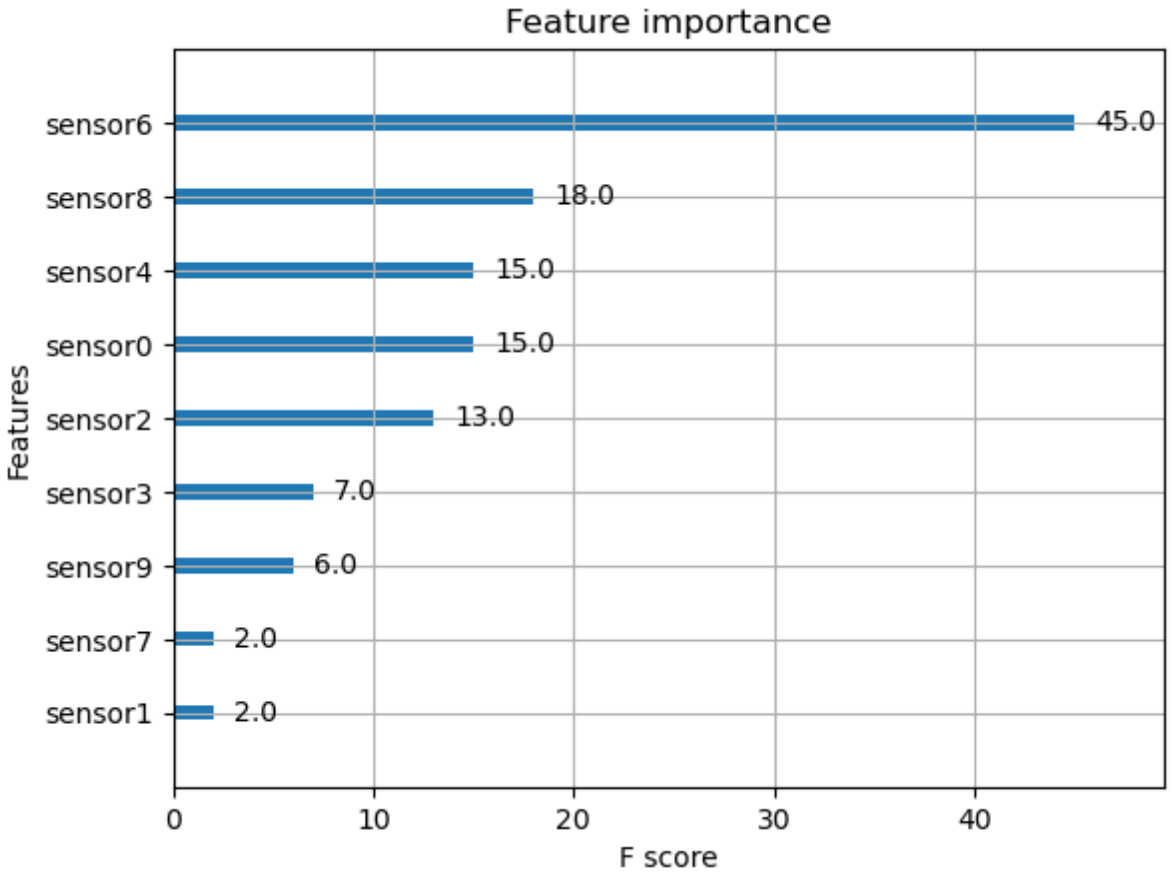
b) XGBoost classifier

In [ ]:
```
xgb = XGBClassifier(n_estimators = 100,
                            n_jobs = -1,
                            oob_score = True,
                            bootstrap = True, random_state=42)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y)
xgb.fit(X, y_train)
# feature importance
print(xgb.feature_importances_)

# plot using default scikit learn feature selection
plt.bar(range(len(xgb.feature_importances_)), xgb.feature_importances_)
plt.show()
```

```
[15:28:50] WARNING: ../src/learner.cc:767:
Parameters: { "bootstrap", "oob_score" } are not used.

[0.06006592 0.02580716 0.01665014 0.00399834 0.08641388 0.
 0.17037797 0.00122901 0.6231604  0.01229716]
```



In [ ]:
```
# plot feature importance using XGBOOST
plot_importance(xgb)
plt.show()
```



Conclusion :

From the above I can conclude that sensor 6 turns out to be an important feature. I have also explained the reasons for chosing the models and methods. The permutation method helps us to identify the feature importance by eliminating bias, on the other hand I explained the downside of it as well.

BAlternatively, I have used XGBoost feature importance and it turned out that the scikit learn default feature_importance gives the same result as previous ones but surprisingly, XGBOOST feature importance gives similar results as seen in the permuatation method because in boosting as we saw there is not much correlation between the features , boosting will use the feature that is not correlated since correlated feature will not be used in the splitting process.

In [ ]: