Aim

1. Using the Lazy Predict package, compare the performance of various Scikit-Learn classifiers. To assess the quality, use the Balanced Accuracy metric. Select one or more classifiers with good quality indicators, write out their full names.
2. Using the SHAP (SHapley Additive exPlanations) package, analyse the impact of each feature on the target variable.
3. Discard low influence features, repeat classifier comparison using Lazy Predict package. Evaluate how the neglect of features with low influence affects the quality of the work of classifiers.
4. Reduce the dimension of the original dataset using any dimensionality reduction method (PCA, UMAP). Conduct an assessment of the influence of features for the obtained dataset. And using the Lazy Predict package, repeat the comparison of classifiers.
5. Draw conclusions about 1) How different ways of transforming a dataset (rejecting features with low influence, lowering the dimension) affect the quality of the classifiers. 2) How the influence of features is redistributed when the dimension is reduced. Is the influence of new signs becoming more uniform?

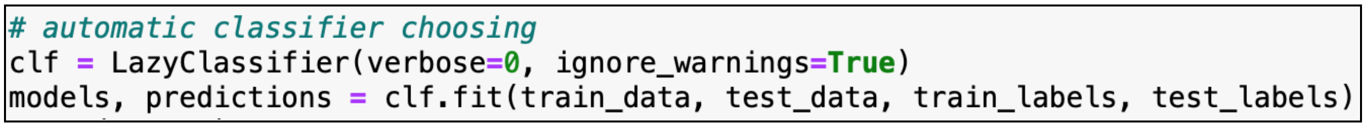
Dataset

It was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository. It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other. The columns in this dataset are:

- Id
- SepalLengthCm (S.L.)
- SepalWidthCm (S.W.)
- PetalLengthCm (P.L.)
- PetalWidthCm (P.W.)
- Species

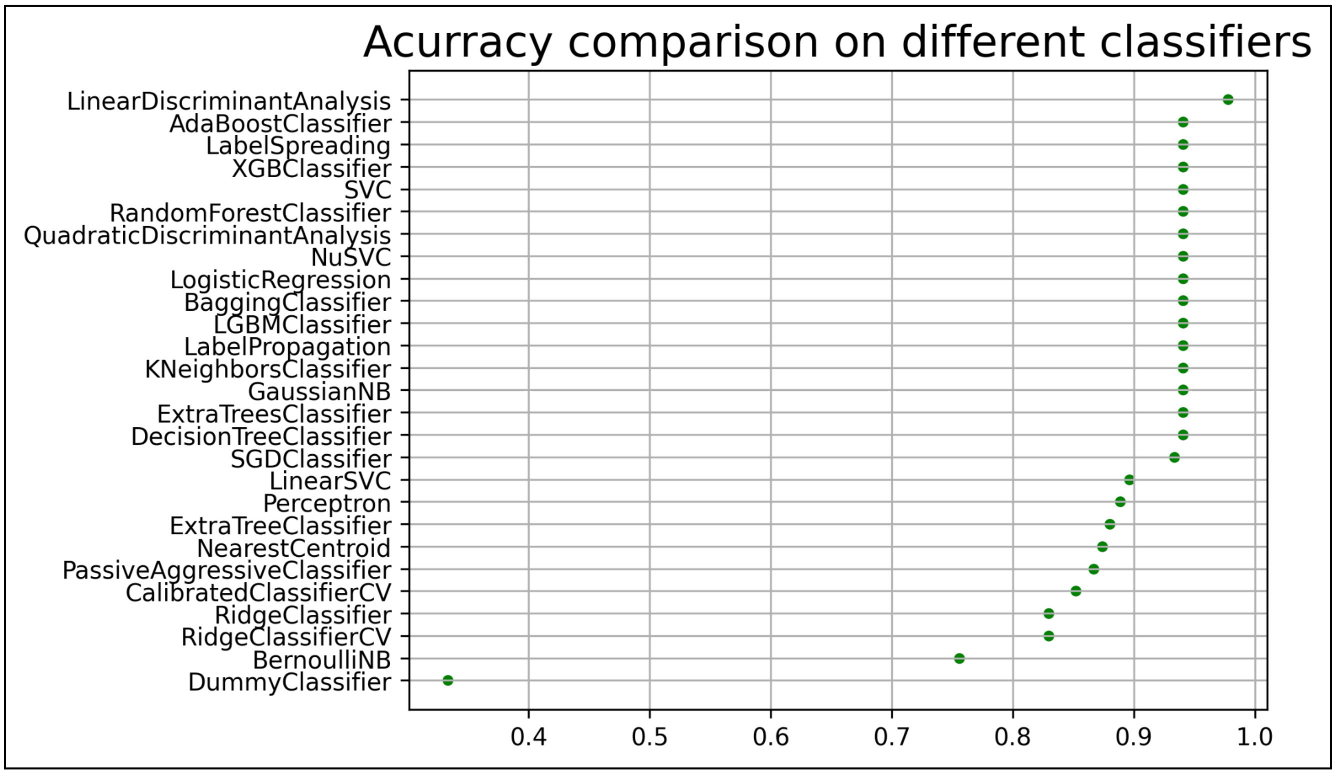
Finding a classifier

LazyPredict will be used as a tool, which is an open-source python library that helps you to semi-automate a Machine Learning Task. It can build multiple models without writing much code and helps understand which models work better for the processed dataset without requiring any parameter tuning. As mentioned before, to evaluate, will use the ‘Balanced Accuracy’ metric. After some simple code, we are able to build a comparison graph.

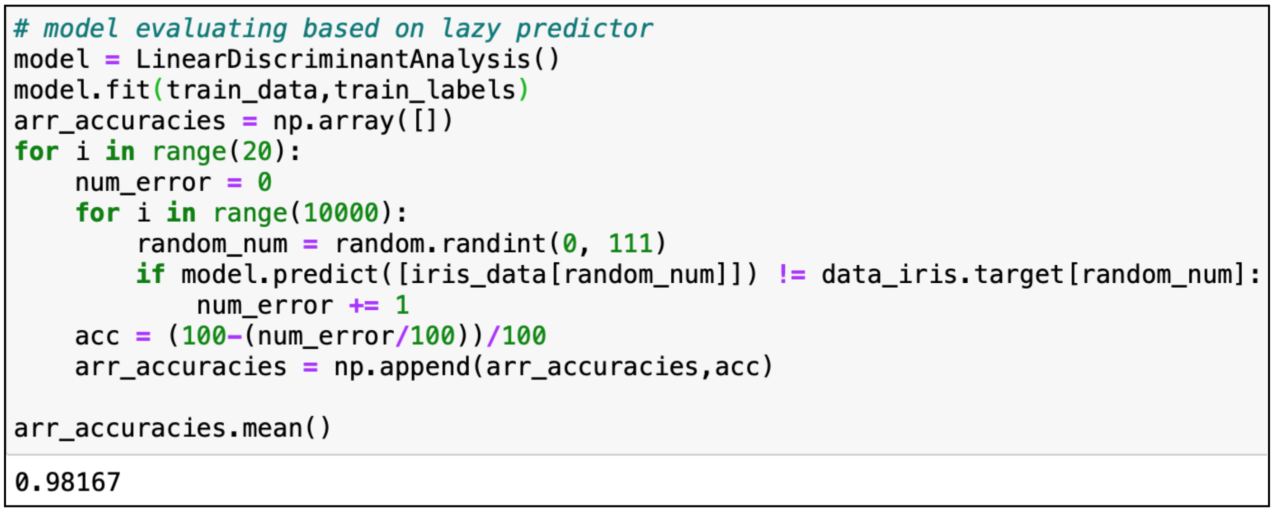


```python
# automatic classifier choosing
clf = LazyClassifier(verbose=0, ignore_warnings=True)
models, predictions = clf.fit(train_data, test_data, train_labels, test_labels)
```

It’s clearly noticeable that we have the best model called ‘Linear Discriminant Analysis’ with its performance value of 0.98. It can also be distinguished that the mode in this data is 0.94. The worst one is ‘Dummy Classifier’ performance of which can be evaluated in 0.24. The average accuracy for all the models is ~0.889.



Linear Discriminant Analysis or **Normal Discriminant Analysis** or **Discriminant Function Analysis** is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. After constructing a model by ourselves, the accuracy calculated by ‘LazyPredictor’ can be confirmed, which is between 0.981 and 0.983 on average.



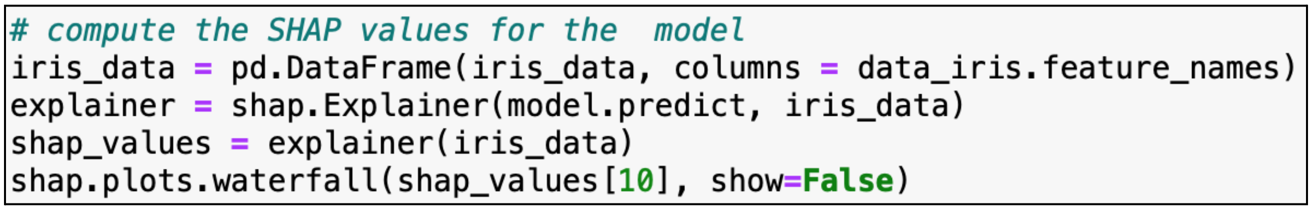
```
# model evaluating based on lazy predictor
model = LinearDiscriminantAnalysis()
model.fit(train_data,train_labels)
arr_accuracies = np.array([])
for i in range(20):
    num_error = 0
    for i in range(10000):
        random_num = random.randint(0, 111)
        if model.predict([iris_data[random_num]]) != data_iris.target[random_num]:
            num_error += 1
    acc = (100-(num_error/100))/100
    arr_accuracies = np.append(arr_accuracies,acc)

arr_accuracies.mean()
```

```
0.98167
```

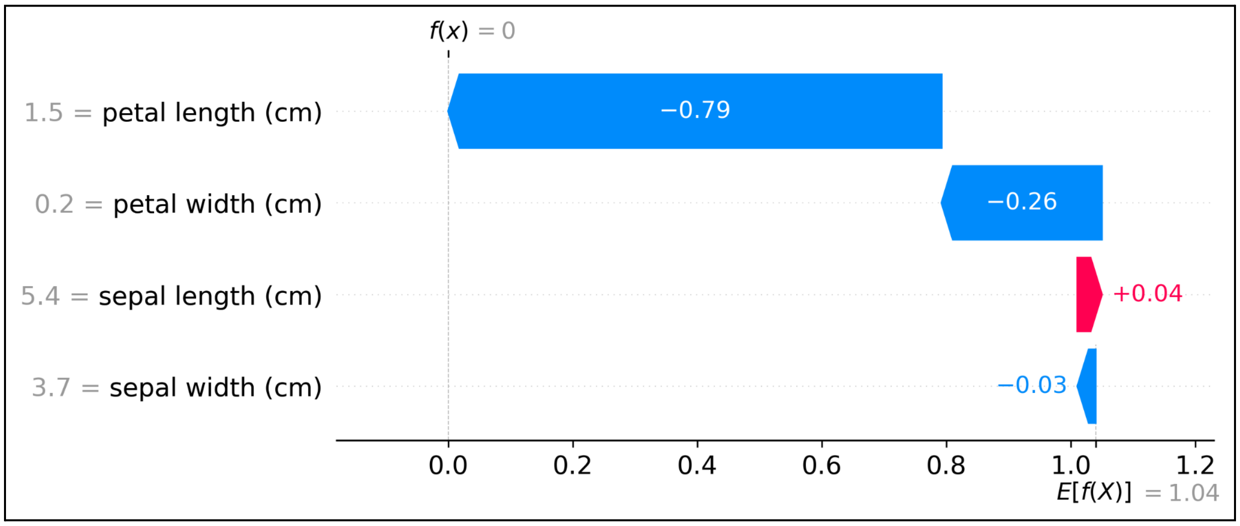
Shapley Additive Explanations

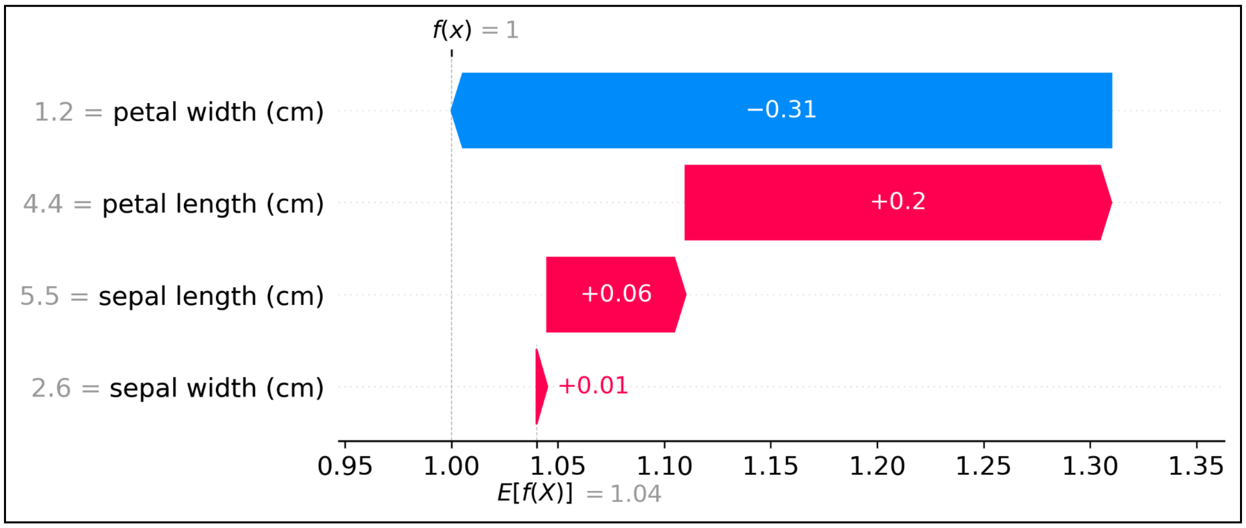
Next, we need to make an analysis on the importance of each feature of our dataset unit. SHAP (SHapley Additive exPlanations) will be used. It is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions.
After a bit of code, we have our shap values and build a descriptive graphs.

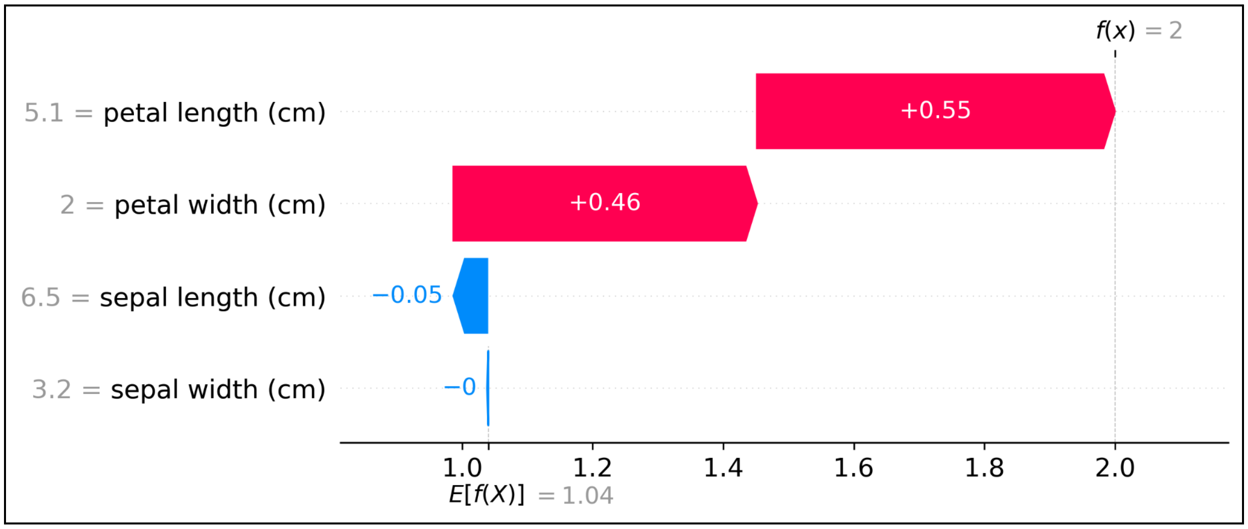


```
# compute the SHAP values for the  model
iris_data = pd.DataFrame(iris_data, columns = data_iris.feature_names)
explainer = shap.Explainer(model.predict, iris_data)
shap_values = explainer(iris_data)
shap.plots.waterfall(shap_values[10], show=False)
```

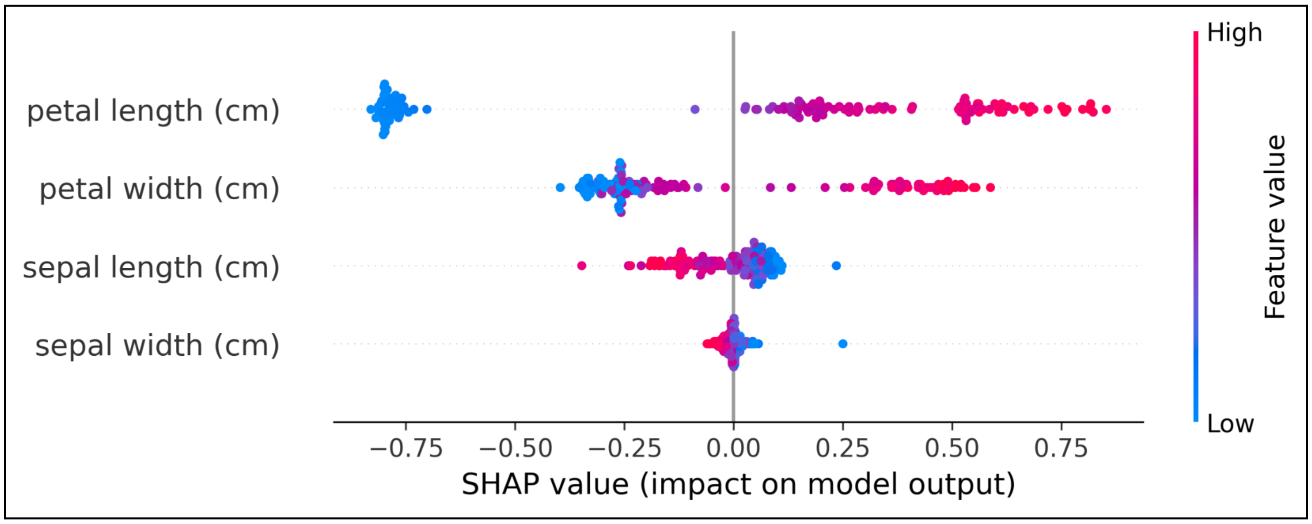
The ‘waterfall’ type of a plot can describe an impact of features for individual input (vector). The x-axis has the values of the target variable which is the numerical variant of the Iris type. x is the chosen observation, f(x) is the predicted value of the model, given input x and E[f(x)] is the expected value of the target variable, or in other words, the mean of all predictions. The absolute SHAP value shows us how much a single feature affected the prediction. On the next visualisation we’ve taken 3 random samples from the IRIS dataset.







We are definitely able to say that for both three cases (graphs), a petal length plays the most important role with the highest values. The second one is petal width. Sepal width doesn't participate at all. Although we can see some patterns with the help of those graphs, we can't really say anything about the general contribution of each input feature. The ‘bee swarm’ plot from Python SHAP library is able to solve the problem.

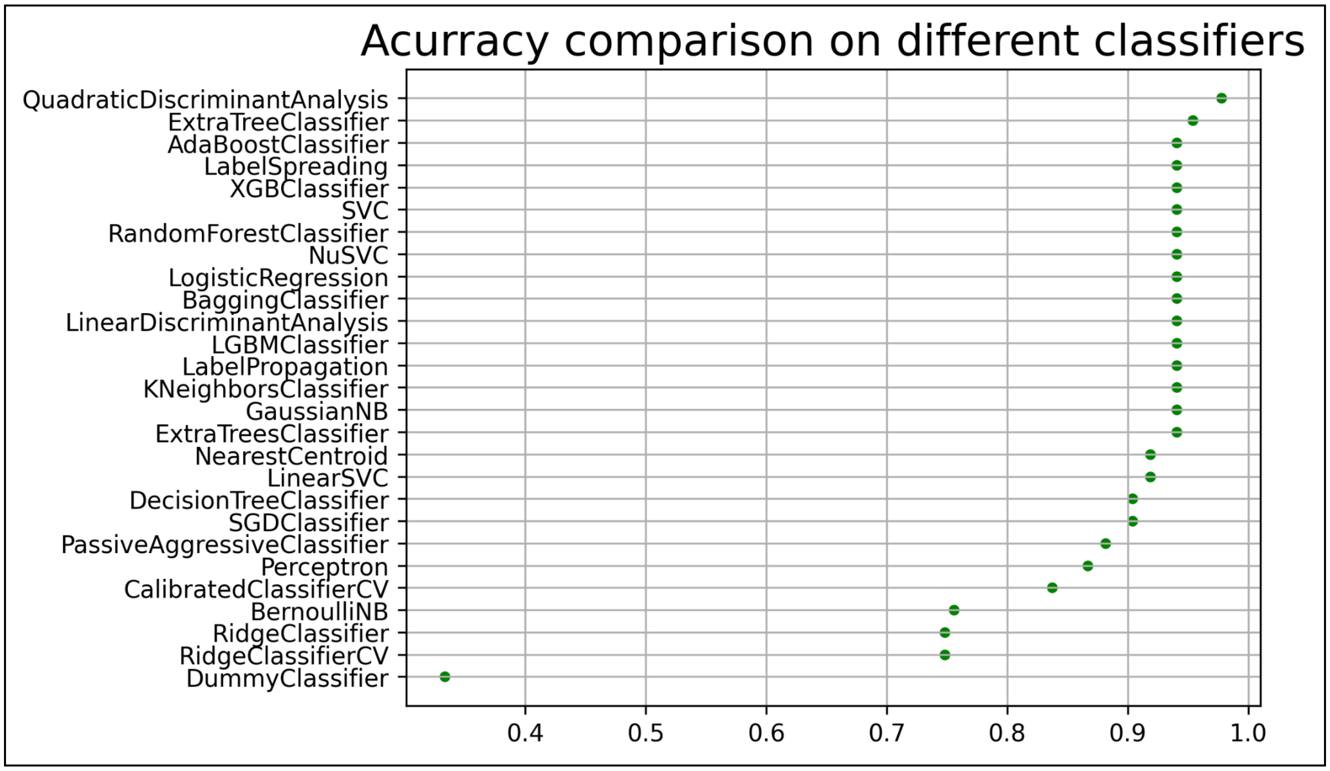


Having 150 data units here, the picture hasn’t changed. For example, low input values (blue colour) of the petal length variable have a perceptible negative contribution to the prediction (SHAP value of ~-0.75), while higher an input value from zero, more contribution it has. A sequence of importance remained the same: PL, PW, SL, SW. It can be observed that for the lower Y (feature name), points are distributed closer to zero.
The next step is to compare the effect of deleting a feature column from a dataset and reducing dataset dimensionality.

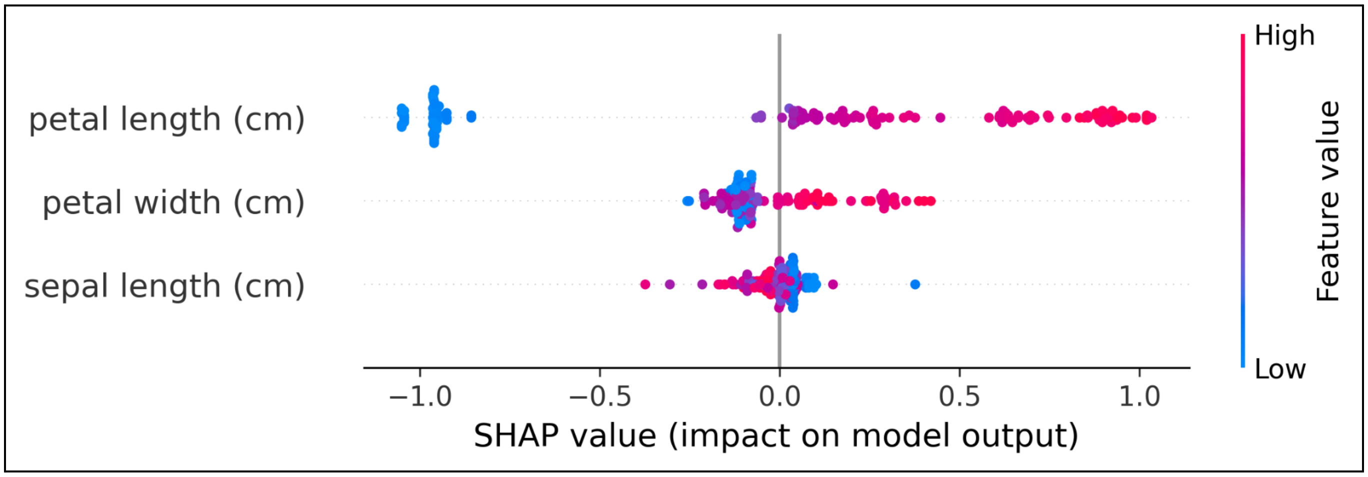
Discarding low valuable features

As mentioned before we have some features, impact of which is very low, in our case it’s a sepal width variable. Now, we can experiment with their removal and compare the old and new results of ‘Lazy Predictor’.

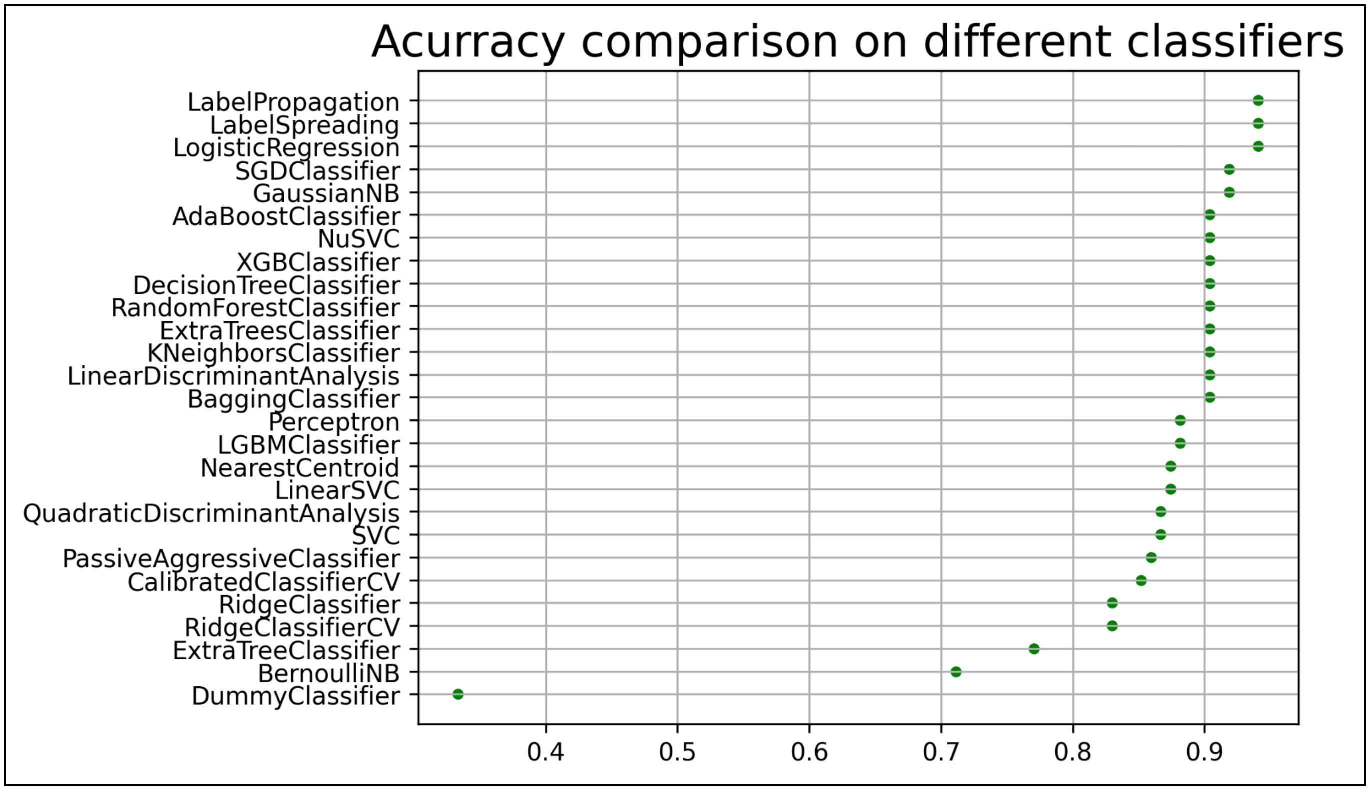
We can see that after the sepal width feature drop, ‘**Quadratic Discriminant Analysis**’ classifier performed the best with the value of 0.98, while Linear Discriminant Analysis accuracy decreased down to 0.94. The mean for all the models now is ~0.885, which is 0.004 lower than the initial one, - no significant difference.



After building a model based on 3 input features and dropping a ‘sepal width’ column, almost nothing has changed. The main pattern looks similar to the one we had with four features featured.



It is also important to see what will happen after getting rid of the most valuable variable - P.L., We can observe that overall performance dropped to an average of ~0.86 and the best one is 0.94 by the ‘**Label Propagation**’ model.



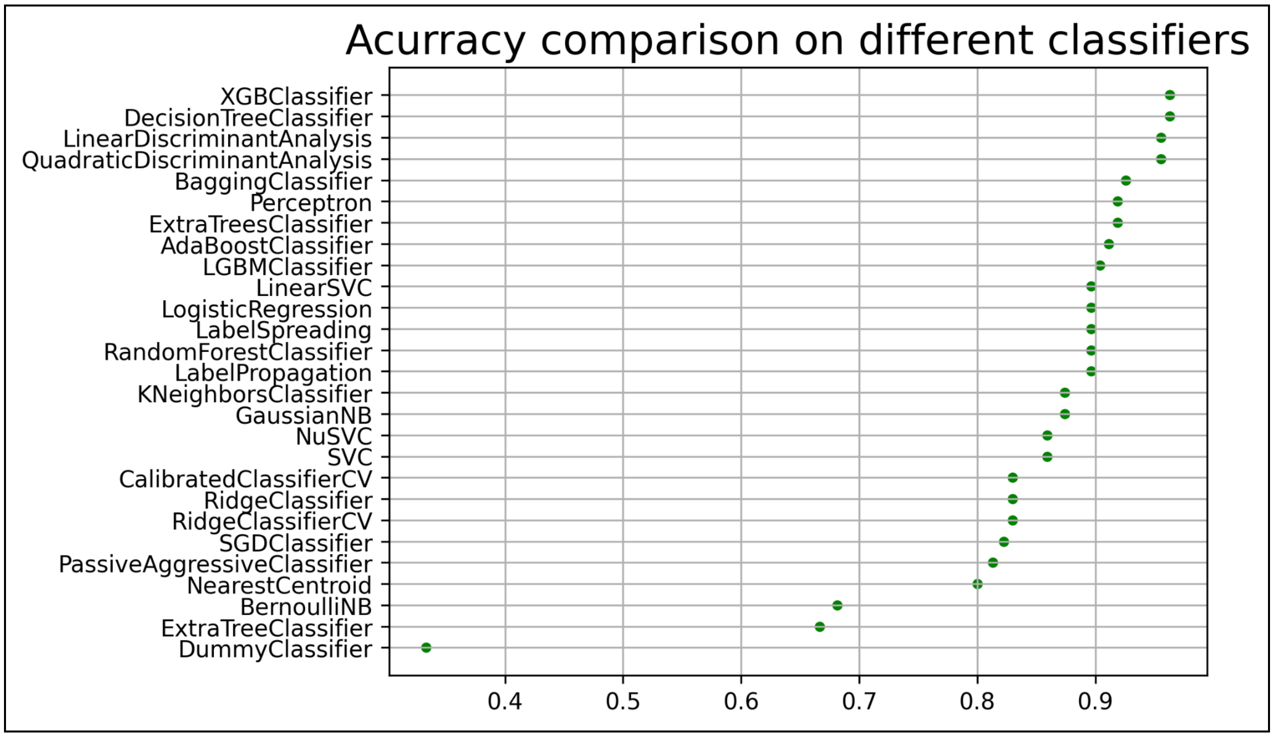
The difference is noticeable in comparison with the previous removal of less important feature: 0.45 % loss in the first case and 3.2 % loss in the second one.

PCA dimension reduction

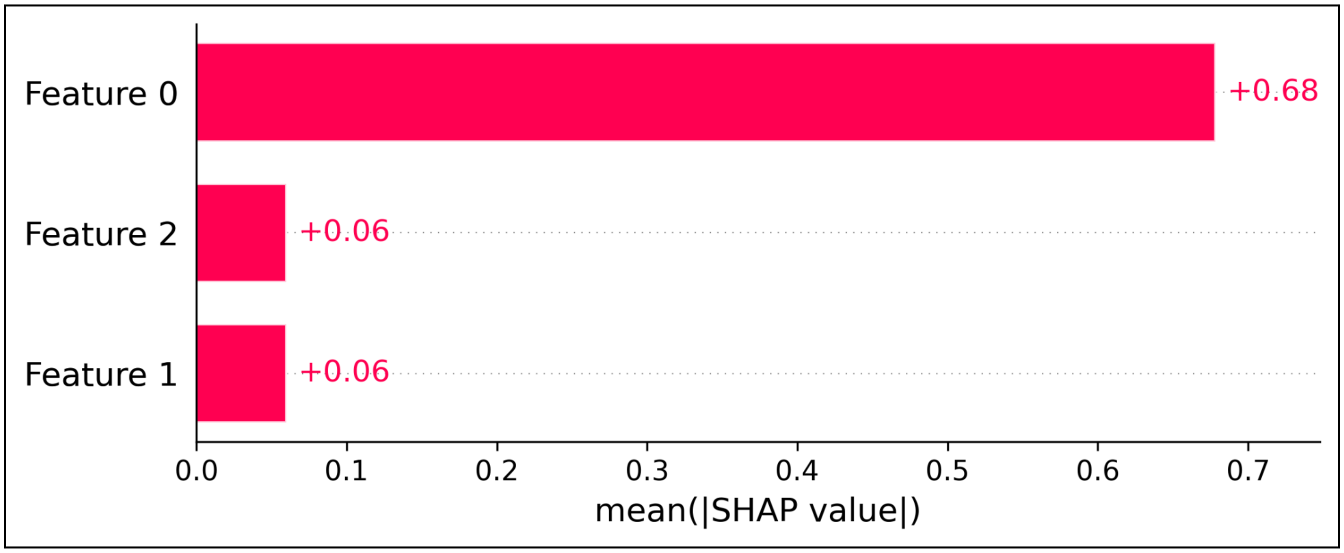
The second way of reducing our data is to use the PCA algorithm. The dimension will be reduced down to three main components.

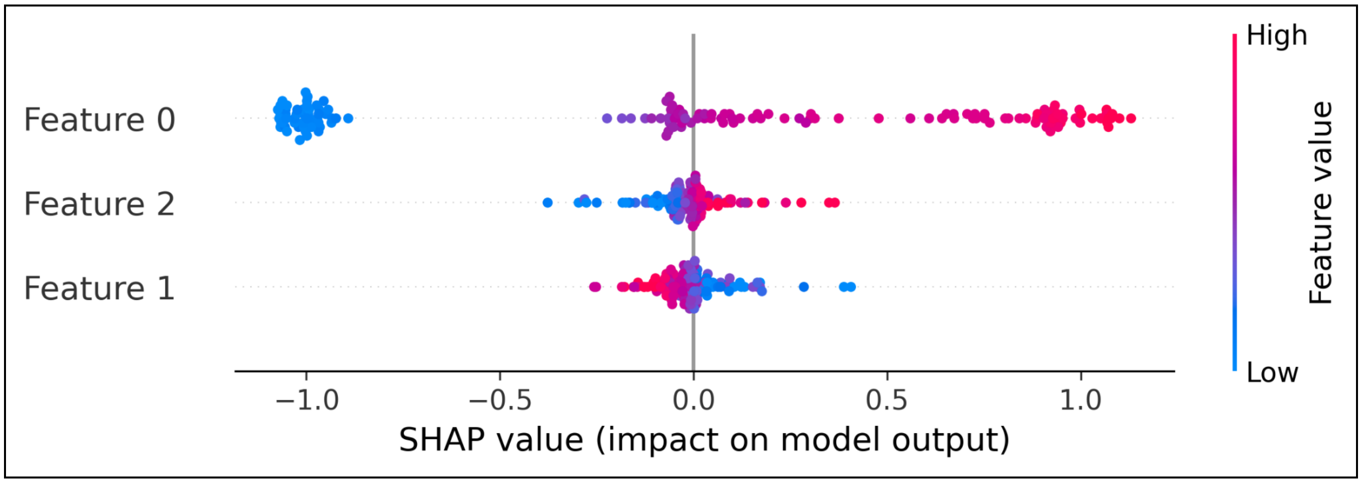
```
pca = PCA(n_components=3)
pca.fit(iris_data)
iris_data_pca = pca.transform(iris_data)
```

'Lazy predictor' says that mean accuracy is 0.86 which is 0.03 lower as an index from reduction by removing one column from the dataset. This time, the best models are **XGBClassifier** and **DecisiontreeClassifier** with performance of 0.95.



So now we need to rebuild our neural network with a new model and analyse how SHAP values have changed. Two next graphs show the importance of each feature after using PCA.





We can’t longer interpret our features because of PCA usage, but it can be seen that feature 2 and feature make exactly the same impact on the model results, while feature 0 now makes more contribution than earlier and has a similar pattern (as Petal Length). Some visual mirroring (negative-positive values) can be noticed in feature 2 and 0.

Conclusion

To sum up, we can say that the removal of non important components doesn’t play any role, and the performance remains the same. We were able to see a difference between accuracy change after removal of least valuable and most valuable: minus 0.45% and minus 3.2% of the initial one. Though some of the SHAP values of uninterpretable features after PCA became similar, overall performance on the dataset has dropped by 0.03 which is more than the effect or ‘column drop’. SHAP is a powerful tool that can help track the value of features and after, make a rough removal of a non important one, while you are still able to describe columns unlike PCA or UMAP algorithms.