Aim

For any of the classifiers from past homework, obtain a quality score based on the F and F1 metrics. For the F-metric, evaluate changes in the quality of the model when changing the weighting coefficient "beta" (optional task). Provide conclusions in the report.

Dataset

It was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, 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

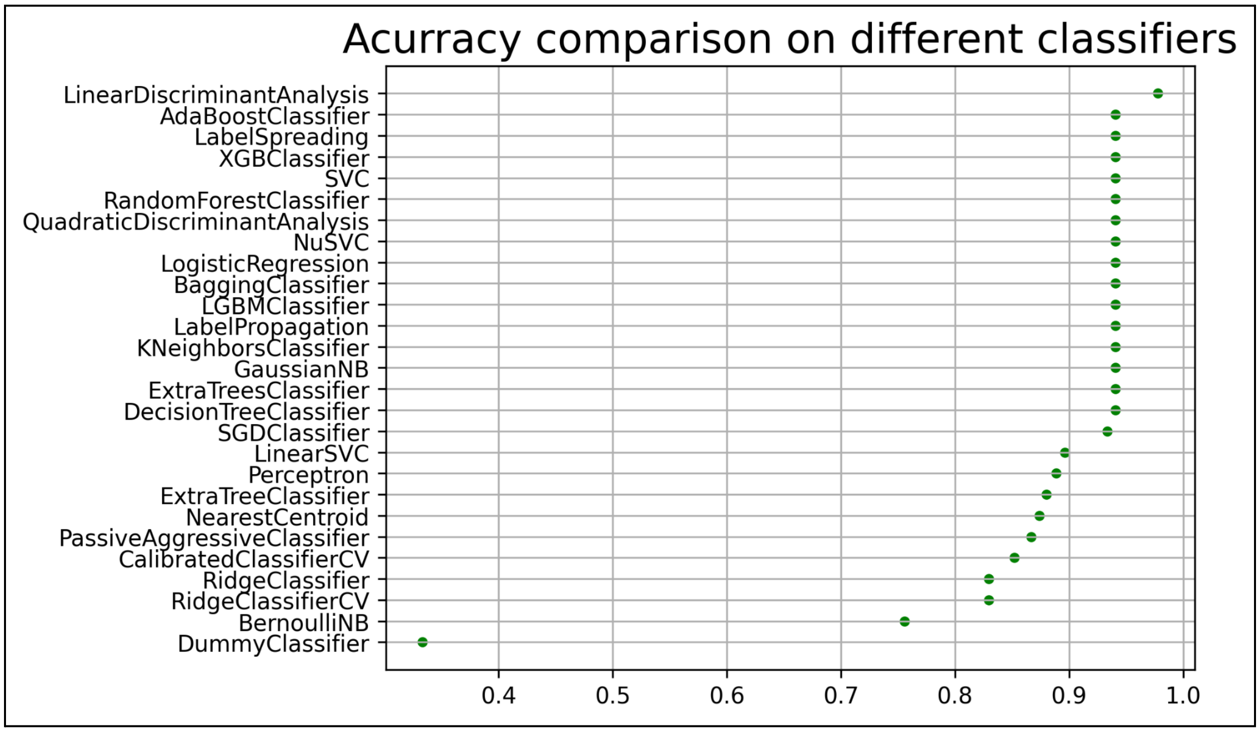
```
data_iris = load_iris()
iris_target = data_iris.target
iris_data = data_iris.data
train_data, test_data, train_labels, test_labels = train_test_split(iris_data,
                                                                    iris_target,
                                                                    random_state=1488)
```

Prepairings

First of all it’s worth it to chose the most suitable classifier although we had one in previous researches. Lazy Predict library is able to this kind of a problem.

```
# choosing a classifier
clf = LazyClassifier(verbose=0, ignore_warnings=True)
models_initial, predictions = clf.fit(train_data, test_data, train_labels, test_labels)
print(predictions)
```

On the result graph we can observe that Linear Discriminant Analysis performed the best, so that type of model will be chosen as today's.



```
model = LinearDiscriminantAnalysis()
model.fit(train_data,train_labels)
```

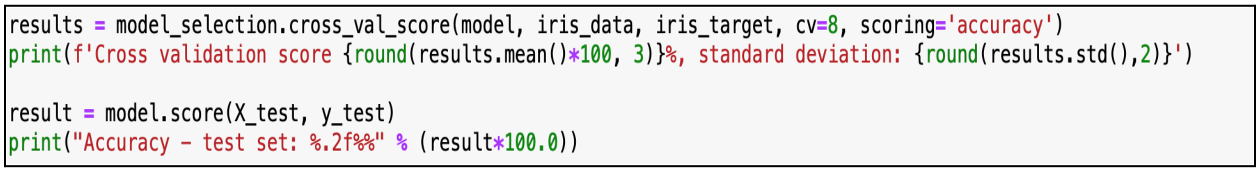
Evaluation

Sklearn library will be used to provide helper functions for our task

Accuracy and cross validation

The very basic metric is an accuracy. Its value coulb be more precise if the extraction of accuracy is done with cross-validation technique. Basically it can be described in next steps:

1. Divide the dataset into N parts: one for testing, other for training
2. Train the model on the training set, validate the model on the test set. Get the accuracy.
3. Repeat steps 1-2 A times, where A is possible number of combinations.



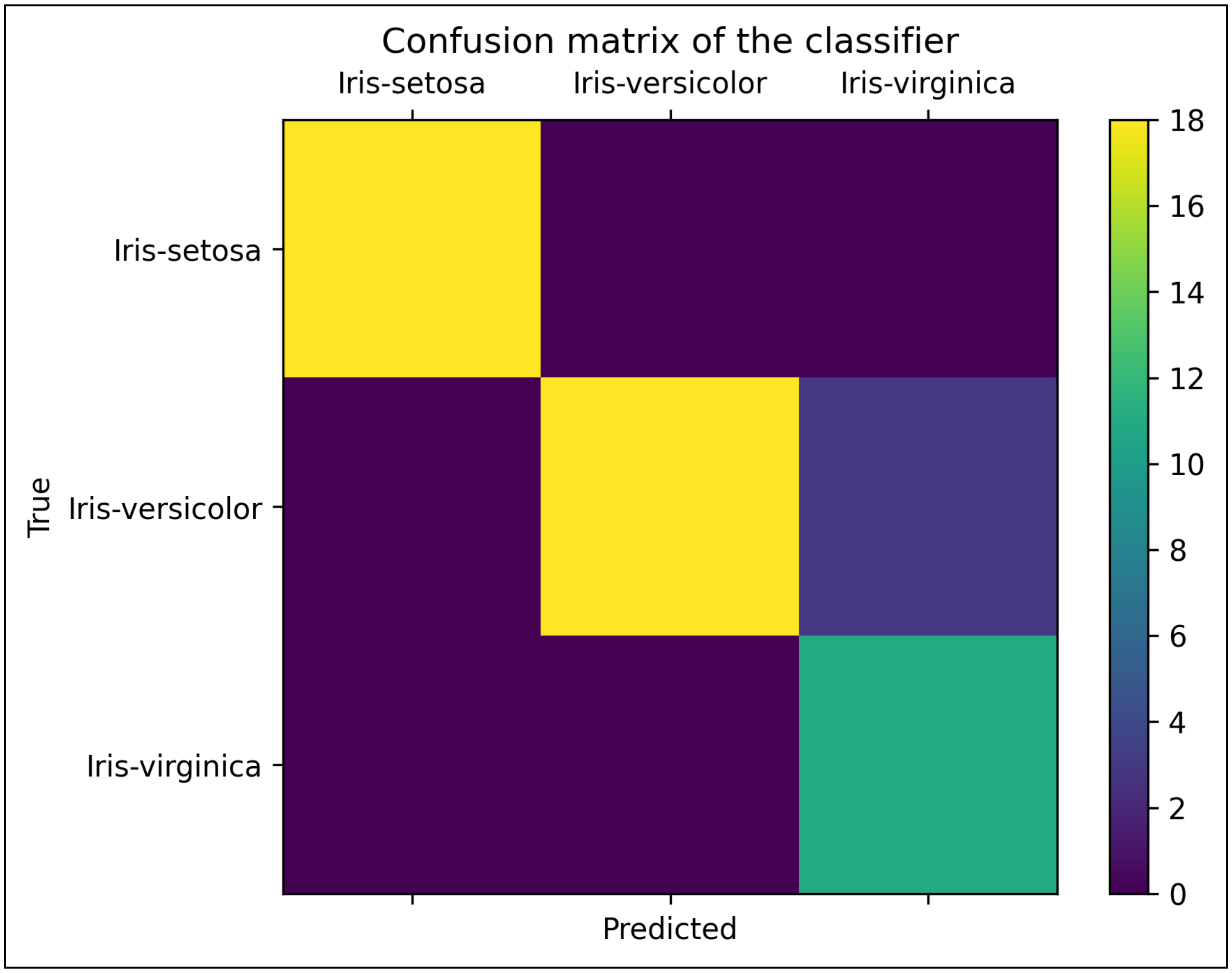
```
results = model_selection.cross_val_score(model, iris_data, iris_target, cv=8, scoring='accuracy')
print(f'Cross validation score {round(results.mean()*100, 3)}%, standard deviation: {round(results.std(),2)}')

result = model.score(X_test, y_test)
print("Accuracy - test set: %.2f%%" % (result*100.0))
```

So after calculations we get an array with accuracy of every 8 (in our case) combinations. The average is ~97.368% while the standard deviation from the mean is 0.04. However if we calculate individual accuracy, giving our test data as input (proportion is 0.33/0.67), the its value will be ~94%.

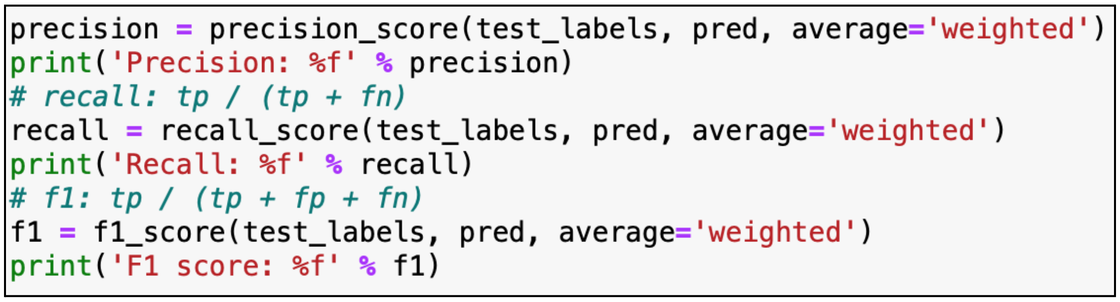
Confusion Matrix

A confusion matrix provides a more detailed breakdown of correct and incorrect classifications for each class. Numbers (colours) of squares represent a amount of correct or incorrect predictions in dependency of ‘Reality’. For example we can see that 18 of 18 of Iris-versecolor were predicted correctly (the center square), however 4 Iris-virginica were predicted as Iris versicolor. It is obvious that the ideal case can be represented by identity matrix, so values are only present on the diagonal. In our case the model performed so well that only 4 errors of 50 are present.



F-measure

F1 is an overall measure of a model's accuracy that combines precision and recall, in that weird way that addition and multiplication just mix two ingredients to make a separate dish altogether. That is, a good F1 score means that you have low false positives and low false negatives, so you're correctly identifying real threats and you are not disturbed by false alarms. An F1 score is considered perfect when it's 1, while the model is a total failure when it's 0. Precision is the number of correct positive results divided by the total predicted positive observations. Recall, on the other hand, is the number of correct positive results divided by the number of all relevant samples (total actual positives). We could have had an f-beta score but our classification problem is **not a binary one,** so recall and precision have the same weight. ROC and AUC also couldn't be used because of the specificity of our model.



```
precision = precision_score(test_labels, pred, average='weighted')
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(test_labels, pred, average='weighted')
print('Recall: %f' % recall)
# f1: tp / (tp + fp + fn)
f1 = f1_score(test_labels, pred, average='weighted')
print('F1 score: %f' % f1)
```

With the code above we have an output:

```
Precision: 0.952
Recall: 0.94
F1 score: 0.941
```

A **precision** says that ~95% of 'positively' predicted values were actually correct, while **recall** tells us that 94% of 'positive' values were predicted truly. The **F1 score** of ~0.941 lies between the previous two measures which is quite a good index.

Ideally, the estimated performance of a model tells us how well it performs on unseen data. Making predictions on future data is often the main problem we want to solve. It's important to understand the context before choosing a metric because each machine learning model tries to solve a problem with a different objective using a different dataset.