2024 Project

Context

This year's project will tackle the Feature Selection Problem in Machine Learning, which is an NP-Complete problem, requiring the use of optimisation metaheuristics.

We have chosen to give you a use-case on genomics data, where you start with a data file with a very large number of columns, and where you have to select the most informative subset of features. You will find below a case study report describing a competitive approach to solving the problem using evolutionary optimisation and a direct supervised evaluation performance criterion as the fitness function (evaluating the function requires training and evaluating a classifier!).

This approach has the drawback of leading to costly fitness function evaluations, which means that population based metaheuristics will be heavily penalized and run slowly.

Objectives

You will be working in groups of 4 students

For this project, we want you to consider two settings:

- An unsupervised setting, where the criterion to minimise will be the mutual information score (you may use the sklearn implementation). You may consider other pertinant criteria of you wish.
- A Supervised setting where you will use a score based on the AUC of an actual classifier (the exact score defined in the case study report)

The objective will be to compare various optimisation metaheuristics (you can reimplement the approached from the case study, or propose other metaheuristics) across both the unsupervised and supervised setting.

For both settings, you are asked to evaluate the machine learning performance (AUC) on a set of classifiers with sklearn and record the execution time.

You will provide a table showing all the metaheuristics considered in both settings, with their average and standard deviation of AUC across classifiers, and runtimes for each of the problem instances that we will supply. Take care to use cross validation and to properly segment the instances for the evaluation.

You will also have to provide a script that given a dataset as input produces all the results to fill the tables.

Data:

We will use data instances from the Gene Expression Omnibus (GEO). We provide you with a consolidated version of the dataset containing the classes, please download it directly through Campus.

Data format

The data is in CSV format. The first column gives you the class corresponding to each sample, and the remaining columns give the features and their corresponding values.

Please look at the file description below for more precise information.

Evaluation criteria and expected outcomes:

You shall submit the following elements:

- The code for your project, with a script that take the path to the dataset and runs all the evaluations and comparisons. **No hardcoded paths!**
- A short report (max 3 pages!) describing your approaches in one page, giving the result table and an analysis of your result. Please highlighting which approach offers the best speed-performance tradeoff.

The grading will follow this scale:

1. Code quality and elegance of proposed approaches [8 marks]
2. Report [8 marks]
   - System description [2 marks]
   - Results table [2 mark]
   - Analysis [4 marks]
3. You ranking compared to the other teams [4 marks]

Submission instructions

You will submit a single zip archive, contraining the last name of all the students in the group on the Campus deposit space **before the 10/11/2024 at 23:59 CET.**

Additional indications:

Unsupervised score computation:

Regarding the unsupervised setting, the mutual_info_score operates on probability density estimates of joint probability distributions between the sequences, whether computed on discrete or continuous data.
It could be reimplemented using kernel density estimation, but typically what's done is to compute an approximation of the density through an histogram of the distribution. You can do that with numpy https://numpy.org/doc/stable/reference/generated/numpy.histogram2d.html and give this as input to the mutual_info_score from sklearn (a similarity, symmetric, but not a metric). The related Jensen-Shanon divergence (from scipy) can also be used (it's a distance and a proper metric) in exactly the same way if you prefer a minimisation problem. The width of the bins is a parameter, the smaller they are, the better the approximation but the slower the computation. Also, too small, would possibly yield bins with single samples, which is to be avoided. You can adjust by visual observation of the histograms.

Integrating your feature extractor to scikit-learn pipelines:

You may integrate your feature selection approaches to scikit-learn by extending some of the base classes used to implement transformation components (TransformerMixin):

```
from sklearn.base import BaseEstimator, TransformerMixin

class CustomFeatureSelector(BaseEstimator, TransformerMixin):
    def __init__(self, param1=None, param2=None):
        # Initialize your parameters here
        self.param1 = param1
        self.param2 = param2

    def fit(self, X, y=None):
        # Learn from the data, e.g., compute feature importances
        # Store any attributes you need for transformation
        return self

    def transform(self, X):
        # Select and return the desired features from X
        return X_transformed
```

Here you can find an example random feature selector:

```
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

class RandomFeatureSelector(BaseEstimator, TransformerMixin):
    def __init__(self, n_features_to_select=10, random_state=None):
        """
        Parameters:
        - n_features_to_select: int, default=10
            Number of features to select.
        - random_state: int or None, default=None
            Seed for the random number generator.
        """
        self.n_features_to_select = n_features_to_select
        self.random_state = random_state

    def fit(self, X, y=None):
        # Check if n_features_to_select is valid
        n_features = X.shape[1]
        if self.n_features_to_select > n_features:
            raise ValueError("n_features_to_select must be <= the number of features")

        # Initialize random state
        rng = np.random.RandomState(self.random_state)

        # Randomly select feature indices
        self.selected_indices_ = rng.choice(
            n_features, self.n_features_to_select, replace=False
        )
        return self

    def transform(self, X):
        # Select columns based on the stored indices
        return X[:, self.selected_indices_]
```

And finally an usage example in a pipeline:

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris

# Load sample data
X, y = load_iris(return_X_y=True)

# Create an instance of the RandomFeatureSelector
selector = RandomFeatureSelector(n_features_to_select=2, random_state=42)

# Build a pipeline
pipeline = Pipeline([
    ('feature_selection', selector),
    ('classification', LogisticRegression())
])

# Fit the pipeline
pipeline.fit(X, y)

# Make predictions
predictions = pipeline.predict(X)
```