Problem Set 03: Linear Classifiers II

Johannes C. B. Dietschreit, Sascha Mausenberger

Due: 28.10.2024

The problems below are meant to be solved using Python functions and libraries. You can do so by either writing short scripts or using Jupyter notebooks. **Report your results in the Quiz on Moodle** to obtain a grade.

Introduction

In this PSet you will get acquainted with `scikit-learn` and make use of some new `rdkit` functions. We will present you with some introductory code for some important packages, however, we expect you to look for functions you might need yourself (the documentations of all mayor Python packages we use in this course are very good, further Google and ChatGPT are of help).

1 `scikit-learn`

`scikit-learn` is a powerful and widely-used Python library for machine learning. It provides simple and efficient tools for data mining and data analysis, built on top of the scientific Python stack, including `NumPy`, `SciPy`, and `matplotlib`.
`scikit-learn` offers a broad range of features, including classification, regression, clustering, dimensionality reduction, model selection, data preprocessing, simple MLPs and more.
One of the main strengths of `scikit-learn` is its user-friendly API, which follows the principle of consistency and reusability. Almost all algorithms can be used with the same workflow.

1.1 Example Workflow

A typical machine learning workflow in `scikit-learn` involves the following steps:

1.1 **Data Loading:** Load the dataset, which can be from built-in datasets, external files, or dataframes.

1.2 **Data Preprocessing:** Preprocess the data by scaling, normalizing, or encoding categorical variables.

1.3 **Model Training:** Initialize and train a machine learning model using the training data.

1.4 **Model Evaluation:** Evaluate the model's performance using a test set and metrics such as accuracy, precision, recall, or mean squared error.

1.5 **Model Tuning:** Optimize the model by tuning hyperparameters using techniques like grid search or randomized search.

1.6 **Prediction:** Use the trained model to make predictions on new data.

1.2 Sample Code

Below is a simple example demonstrating how to use `scikit-learn` to solve a classification problem (we have not covered the classifier used below in class yet):

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Preprocess data (standardization)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

The Problems

1 Non-linear Classification with Linear Classifier

Load the numpy arrays for the 2D data points (`Xs.npy`) and their labels (`ys.npy`). This time, the labels are not -1 and 1 but Boolean instead. `scikit-learn` can handle those and it makes plotting easier.

1.1 Plot the data and use two different colors for the two labels. (1.0 P)

1.2 Use `train_test_split` from scikit-learn with `random_seed=42` and `test_size=0.2` to split your data set, check the online documentation for how to do this. How many examples are in the training set? (1.0 P)

1.3 Instead of the self-written function from last week, use the Perceptron class from `scikit-learn` to classify the data. Feel free to play around with the different settings, but report your results for the **default settings**! Train the perceptron and report the score (1 - missclassification error) of your classifier (the score is an attribute of the classifier, you do not need to program this). Report your result with a precision of **three** digits after the decimal point. (1.0 P)

For your own education, plot the results of the classifier and compare them with the ground truth.

1.4 As you can see in the initial plot, the points are not linearly separable. Hence, the linear classifier from the previous question did not perform well.

Use `PolynomialFeatures` from `sklearn.preprocessing` to obtain polynomial feature of the current data with dimensions (x_1, x_2). Use a polynomial degree of 2. Only apply `fit` or `fit_transform` to the training set, as we do not know the test data at this time. Then apply `transform` to the test data. For polynomial features, this two-step process should not make a difference, but it is important that you get used to it (just like with the standardization shown in the example code in the introduction, do NOT use standardization here). How long is now a feature vector of a training example $\boldsymbol{\phi}(\mathbf{x}^{(i)})$ (the answer is a positive integer)? (0.5 P)

What is the minimum value among all test features (the minimum of the matrix of the size $n_{\text{test}} \times d_{\text{feat}}$)? (0.5 P)

1.5 Create a new perceptron object, train it on the training set with the polynomial features and report the score for the polinomial test set. Report your result with a precision of **three** digits after the decimal point. (1.0 P)

2 Creating Features for Molecules

Load the dataset `solubility_dataset.csv` using `pandas`.

2.1 The table is filled with data for many molecules, but first we need to create our own binary labels. We are interested in whether a molecule is more soluble in an organic solvent than in water or not. The relative solubility is encoded in `MolLogP`.

$$\log P = \log_{10} \frac{c_{\text{organic}}}{c_{\text{water}}}$$

How many molecules are more soluble in a non-polar solvent? (1.0 P)

2.2 Now we need to featurize the molecules in order to train any model. Create `rdkit` mols using `rdkit.Chem.MolFromInchi`. Make sure to use the full InChI not the shorter InChIKeys. Store all mols in a list or array. Next use the following code to set up a Morgan fingerprint generator

```
from rdkit.Chem import rdFingerprintGenerator

# fpSize = length of the feature vector
# radius used to look for substructures
morgan_fp_gen = rdFingerprintGenerator.GetMorganGenerator(radius=2,
    fpSize=2048)

X_OneHotFingerprint = np.array([morgan_fp_gen.GetFingerprint(mol).
    ToList() for mol in mol_list])
X_CountFingerprint  = np.array([morgan_fp_gen.GetCountFingerprint(mol
    ).ToList() for mol in mol_list])
```

Here you generate 2 possible sets of features, one consisting simply of 1 and 0 indicating whether a substructure is present, the other one counting how often one appears. Report the covariance of the first feature between the one-hot and count fingerprints, round your result to 3 digits after the decimal point. (1.0 P)

2.3 For both the normal Morgan fingerprint and the count fingerprints, create two train-test splits, again using `random_seed=42` and `test_size=0.2`, i.e., the same molecules are assigned to train and test, respectively. If you were unable to create the labels in part 2.2.1, load the file `Labels_part2.npy` instead. Train two Perceptron models, one for each set of examples each. Report the scores for both fingerprints, round your result to 3 digits after the decimal point. (each 0.5 P)

2.4 Create CountFingerprints with `GetCountFingerprint` using a feature vector length of 8 instead of 2048 (controlled with `fpSize`). Train two classifiers, one using the 8-dimensional features as is, and another with polynomial features of degree 2 (45 dimensions). Create training and test sets using the usual size and random seed. To determine whether the polynomial features create a classifier that generalizes better, compute the ratio of the scores of these two models (polynomial score divided by normal model). Round your result to 3 digits after the decimal point. (1.0 P)

2.5 Separate the LogP values into bins of width 2.3 ($\approx \ln 10$). Use the model trained on the Count fingerprint of length 2048. Apply the classifier to the data separately in those bins (still classifying whether LogP is smaller/larger 0). For the ease of coding, apply the classifier to the entire data set, not just the test set. Record the performance (score) for the examples in each bin and plot it afterwards against the bin center. What do you observe? (1.0 P)