Natural Language and Information Retrieval
Degree in Data Science
ETSINF. Academic year 2022-2023

Practical Work 3

Session 2: Supervised classification models.

Reynier Ortega Bueno
rortega@prhlt.upv.es

Goal

The main objective is to train and validate several supervised classification models for addressing the tasks proposed in the shared task HUHU. For that, the scikit-learn Python library should be used. Particularly, students have to use supervised methods such as SVM, Logistic Regresión, Decision Tree, Multilayer Perceptron, etc.

Bibliography

- DUDA, Richard O., et al. *Pattern Classification*. John Wiley & Sons, 2006. Chapter 6
- THEODORIDIS, Sergios; KOUTROUMBAS, Konstantinos. *Pattern Recognition*. Elsevier, 2006. Chapter 4
- BISHOP, Christopher M.; NASRABADI, Nasser M. *Pattern Recognition and Machine Learning*. New York: Springer, 2006. Chapters 4, 5 and 7
- FERNÁNDEZ, Alberto, et al. *Learning from imbalanced data sets*. Cham: Springer, 2018.
- https://scikit-learn.org/stable/modules/svm.html
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier
- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifiier
- https://imbalanced-learn.org/stable/user_guide.html#user-guide

AUTOMATIC CLASSIFICATION (supervised models)

Supervised classification is a machine learning algorithm that aims to predict a categorical target variable based on a set of input features. The algorithm is trained on a labeled dataset, where the correct output is already known, to learn the relationship between the input features and the target variable. The trained algorithm can then predict the target variable for new input data it has not seen before. The algorithm's accuracy[1] is measured by comparing the predicted output to the ground truth output for the test data. Some popular algorithms for supervised classification include support vector machines, logistic regression, decision trees, multilayer perceptrons, etc.

Support Vector Machines (SVMs)

SVMs are a type of machine learning algorithms used for classification and regression tasks[2]. The main idea behind SVMs is to find the best possible boundary or hyperplane to separate two or more classes in a high-dimensional space.

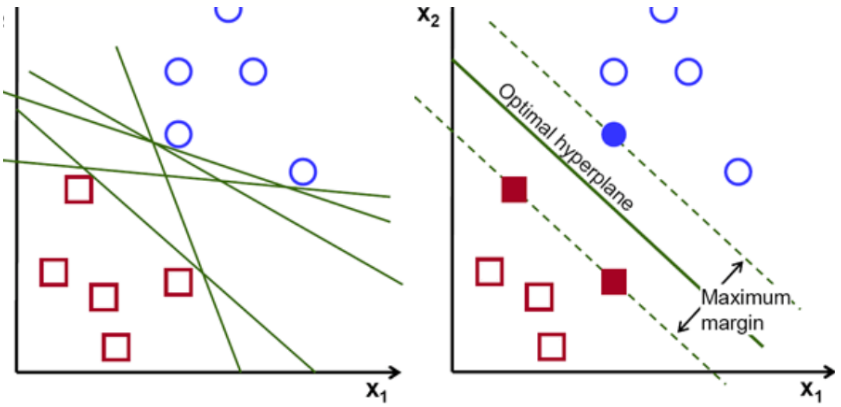

Figure 1: Hyperplane with maximum margin.

[1] The accuracy is defined as the ratio of the number of correctly classified instances to the total number of instances in the dataset.

[2] The main difference between classification and regression tasks lies in the nature of the output variable. Classification tasks predict categorical variables, while regression tasks predict continuous variables.

To do this, SVMs use a technique called *maximum margin classification*, which aims to find the hyperplane with the largest distance between the data points of the different classes (see the Figure 1). The data points closest to the hyperplane on either side are called *support vectors*, and they define the hyperplane.

SVMs work by mapping the input data points into a higher-dimensional space, where finding a hyperplane to separate the data points is more effortless. This is done using kernel functions, which calculate the similarity between data points in the high-dimensional space. Some common kernel functions include Linear, Polynomial, and Radial Basis Functions (RBF).

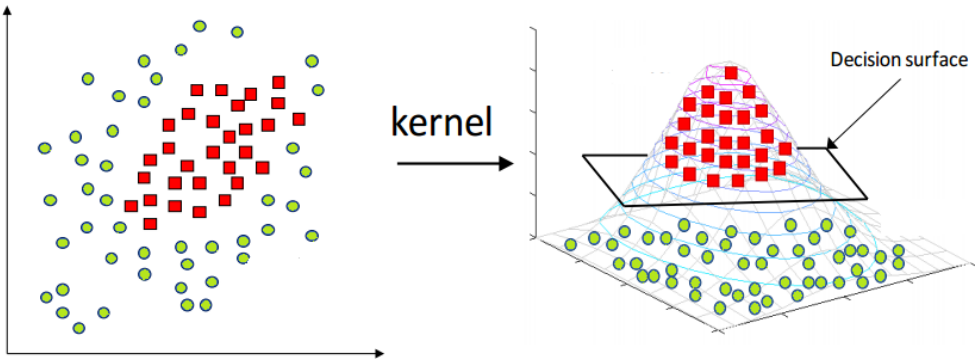

Figure 2: Transformation from a 2D to a 3D space.

Once the hyperplane is found, SVMs can classify new data points based on which side of the hyperplane they fall on. SVMs are helpful for image classification, spam detection, and text classification tasks such as sentiment analysis.

Advantages

- ➢ One of the advantages of SVMs is that they are less prone to overfitting than other classification algorithms, such as decision trees or neural networks.
- ➢ SVMs offer good accuracy in many problems of classification.
- ➢ SVMs use less memory than other classifiers because they use a subset of training points in decision-making.
- ➢ SVMs perform well with a clear separation margin and in high-dimensional spaces.

Disadvantages

- ➢ SVMs are not suitable for large datasets due to their high training time.
- ➢ SVMs are designed to solve binary classification problems.
- ➢ SVMs perform poorly with overlapping classes and are also sensitive to the choice of kernel function and the tuning of hyperparameters. Therefore, selecting these parameters to achieve the best performance is essential.

SVC, NuSVC and LinearSVC are methods to perform binary and multi-class classification on a dataset. SVC and NuSVC are similar but accept slightly different sets of parameters and have different mathematical formulations. On the other hand, LinearSVC is another implementation of Support Vector Classification for the case of a linear kernel. LinearSVC does not accept parameter kernels, which are assumed to be linear.

Hyperparameters

- ➢ *kernel*: Defines the transformation of the input data of the dataset.
- ➢ *regularization*: Reduces overfitting, this hyperparameter reduces the variance of the model parameters. However, it does so at the expense of adding bias to the estimation.
- ➢ *parameter C*: Controls how much error is tolerable in optimization. A lower value of C creates a hyperplane with a small margin, and a higher value of C makes a hyperplane with a more significant margin.

EXAMPLE

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn import svm
# Here, we are creating an artificial dataset for a binary classification problem (similar to the subtask 1 in HUHU).
# In practice, X represents the matrix with text representations from HuHu data, and y represents the corresponding labels of either humorous or non-humorous texts.

X, y = make_classification(n_features=100, n_redundant=10, n_informative=60, n_clusters_per_class=5, n_classes=2)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=1234)
#Using the default parameters
clf = svm.SVC()
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
print("Predicted\t-->\t", predicted)
print("GroundTruth\t-->\t", y_test)
```

What can we do when we have more than two classes in our classification problems?

Multiclass classification problems are a type of machine learning problem where the goal is to classify data points into one of three or more predefined classes. In contrast to binary classification, where the goal is to classify data points in one of two classes, multiclass classification problems require classifiers to identify multiple possible outcomes. For this purpose, Sklearn implements two strategies (**One-vs-Rest** and **One-vs-One**) that allow extending the SVC operation to multiple classes.

- ➢ One-vs-Rest (OvR) approach: In this approach, SVM trains K binary classifiers, where K is the number of classes. Each classifier is trained to distinguish one class from the rest. During testing, each classifier produces a score for each class, and the class with the highest score is selected. This approach can be more efficient than One-v-One, but it may not work well when the classes are imbalanced. In the case of the SVC class, it is sufficient to define the parameter *decision_function_shape*; by default, it has the value *"ovr"* which represents the one-vs-rest approach.
- ➢ One-vs-One (OvO) approach: In this approach, the SVM trains K(K-1)/2 binary classifiers, where K is the number of classes. Each classifier is trained to distinguish between a pair of classes. Each classifier produces a prediction during testing, and the class with the most votes is selected. This approach works well when the number of classes is small. In case of a tie (between two classes with an equal number of votes), the class with the highest average value is selected. Since it requires adjusting K(K-1)/2 classifiers, this approach is usually slower than OvR, due to its quadratic complexity.

Logistic Regression (LR)

Logistic Regression is a binary classification algorithm that models the probability of a binary outcome (e.g., 0 or 1) as a function of one or more input variables. The model's output is a probability score, which is transformed into a binary prediction by applying a decision threshold. The algorithm estimates the coefficients of a linear function that map the input variables to the log-odds of the binary outcome. These coefficients are typically calculated using maximum likelihood estimation.

Advantages

- ➢ LR has several advantages, including its simplicity, interpretability, and ability to handle non-linear relationships between the input variables and the outcome.
- ➢ LR makes no assumptions about class distributions in feature space.
- ➢ LR performs well in many simple datasets, particularly, when the dataset is linearly separable.

Disadvantages

- ➢ LR assumes that the relationship between the input variables and the log-odds of the outcome is linear, and it may not work well when the classes are imbalanced or when the input variables are highly correlated.
- ➢ If the number of data points is smaller than the number of features, LR should not be used; otherwise, it may lead to overfitting.
- ➢ Non-linear problems cannot be solved with logistic regression because it has a linear decision surface.
- ➢ Linearly separable data are rarely encountered in real-world scenarios.

Hyperparameters

penalty: Specify the norm of the penalty by default the model uses l2 norm. Other options are *l1* and *elasticnet*[3].

tol: Tolerance for stopping criteria by default=1e-4

parameter C: Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

solver: Algorithm to use in the optimization problem. Default is *lbfgs*, the other possibilities are *liblinear*, *newton-cg*, *newton-cholesky*, *sag*[4] and *saga*[5].

EXAMPLE

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X, y = make_classification(n_samples=1000, n_features=40,
n_redundant=1, n_informative=20, n_clusters_per_class=2,
n_classes=2)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.01, random_state=1234)
clf = LogisticRegression()
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
print("Predicted\t-->\t", predicted)
print("GroundTruth\t-->\t", y_test)
```

[3] Elastic Net produces a regression model that is penalized with both the **L1-norm** and **L2-norm**. The consequence of this is to effectively shrink coefficients (like in ridge regression) and to set some coefficients to zero (as in LASSO).
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html

[4] Details about the mathematical formulation of the SAG method can be found in
https://link.springer.com/article/10.1007/s10107-016-1030-6

[5] Details about the mathematical formulation of the SAGA method can be found in
https://www.di.ens.fr/~fbach/Defazio_NIPS2014.pdf

Decision Trees (DTs)

Decision tree classifiers are a type of supervised machine learning algorithm that is used for both classification and regression problems. They work by recursively partitioning the input space into regions, each associated with a class label or a predicted value. The algorithm builds a tree-like model where each internal node represents a test on an input variable, and each branch corresponds to the possible outcomes of the test.

Advantages

- ➢ DTs have several advantages, including their interpretability and simplicity. They are used as white box models. Trees can be visualized.
- ➢ DTs have the ability to handle numerical and categorical input variables. Notice that the Sklearn implementation does not support categorical variables.
- ➢ DTs capture non-linear relationships between the input variables and the outcome.
- ➢ DTs can handle missing and noisy data and are robust to outliers.

Disadvantages

- ➢ DTs are prone to overfitting, especially when the tree is deep and complex. Several techniques have been developed to mitigate overfitting, including pruning, ensemble methods (such as random forests and gradient boosting), and regularization.
- ➢ DTs can be unstable because slight variations in the data can generate a completely different tree.
- ➢ DTs learning algorithms rely on heuristic algorithms to make locally optimal decisions at each node. Such algorithms cannot guarantee a globally optimal decision tree.

Hyperparameters

criterion: The function to measure the quality of a split. Supported criteria are *gini*, *entropy*, *log_loss* by default, the criterion used is *gini*[6].
splitter: The strategy used to choose the split at each node. Supported strategies are *best* to choose the best partition and *random* to choose the best random split.
max_depth: The maximum depth of the tree. If *None*, nodes are expanded until all leaves are pure or until all leaves contain less than *min_samples_split* samples.
min_samples_split: The minimum number of samples required to split an internal node, the default value is set to 2.

min_samples_leaf : The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least *min_samples_leaf* training samples in each left and right branch. This may have the effect of smoothing the model, especially in regression.
max_features: The number of features to consider when looking for the best split. It can be set to an integer value, float value or *auto*, *sqrt*, and *log2, by default it is set to None.*

EXAMPLE

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
X, y = make_classification(n_samples=1000, n_features=40,
n_redundant=1, n_informative=20, n_clusters_per_class=2,
n_classes=2)
```

[6] The mathematical formulation for the gini criterion can be found in the link https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.01, random_state=1234)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
print("Predicted\t-->\t", predicted)
print("GroundTruth\t-->\t", y_test)
```

Multilayer Perceptron (MLP)

A Multilayer Perceptron (MLP) is a type of Artificial Neural Network (ANN) that consists of multiple layers of interconnected nodes (or neurons) that process and transform input data into output. The input data is fed into the input layer, which passes the data through one or more hidden layers before producing the outcome in the final output layer. Each neuron in the MLP receives input from the neurons in the previous layer, calculates a weighted sum of the inputs, applies an activation function, and passes the output to the next layer. The weights and biases of the neurons are learned through a training process that adjusts them to minimize the error between the predicted output and the true output. MLPs are commonly used in classification and regression tasks and are prevalent in applications such as image recognition, speech recognition, and natural language processing.

Advantages

- ➢ MLPs are capable of learning non-linear relationships between inputs and outputs, making them useful for complex problems where linear models may not be sufficient.
- ➢ MLPs can be used for both classification and regression tasks and can handle a variety of input.
- ➢ MLPs can handle large amounts of data and can be scaled up by adding more layers or neurons, allowing them to handle more complex tasks.

- ➢ MLPs are capable of generalizing from training data to unseen data, making them suitable for tasks such as pattern recognition and prediction.
- ➢ MLPs can adapt to changing inputs and can learn from new data without requiring retraining from scratch.
- ➢ MLPs can be trained and run on parallel computing systems, allowing for faster processing of large amounts of data.

Disadvantages

- ➢ MLPs can take a long time to train, especially for large and complex datasets. The training process can be computationally expensive and may require a lot of resources.
- ➢ MLPs can be prone to overfitting, where the model learns to fit the training data too closely and does not generalize well to new data. Regularization techniques can be used to address this issue.
- ➢ MLPs can get stuck in local optima during the training process, where the model converges to a suboptimal solution instead of the global optimum. This can be addressed by using various optimization algorithms and techniques.
- ➢ MLPs can be difficult to interpret and understand, making it challenging to explain how the model makes predictions.

Hyperparameters

hidden_layer_sizes : array-like of shape(*n_layers - 2*,), *default=(100,).* The *ith*-element represents the number of neurons in the *ith*-hidden layer.

activation : Activation function for the hidden layer, *default=relu*. Other activation functions are *identity*, *logistic*, *tanh* and *relu*.

solver : The solver for weight optimization, *default=adam*. Other solvers are *lbfgs* and *sgd.*

batch_size: Size of minibatches for stochastic optimizers, *default=auto*. If the solver is *lbfgs*, the classifier will not use minibatch. When set to *auto*, *batch_size=min(200, n_samples)*.

learning_rate: Learning rate schedule for weight updates, *default=constant*. Other schedules are *invscaling* and *adaptive.*
learning_rate_init: The initial learning rate used, *default=0.001*. It controls the step-size in updating the weights. Only used when solver=*sgd* or *adam*.
max_iter: Maximum number of iterations, *default=200*. The solver iterates until convergence. For stochastic solvers (*sgd* and *adam*), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.
tol: Tolerance for the optimization, *default=1e-4*. When the loss or score is not improving by at least *tol* for *n_iter_no_change* consecutive iterations, unless *learning_rate* is set to adaptive, convergence is considered to be reached and training stops.
early_stopping: Whether to use early stopping to terminate training when validation score is not improving, default=False. If set to True, it will automatically set aside 10% of training data as validation and terminate training when validation score is not improving by at least *tol* for *n_iter_no_change* consecutive epochs. The split is stratified, except in a multilabel setting. If early stopping is False, then the training stops when the training loss does not improve by more than tol for *n_iter_no_change* consecutive passes over the training set. It is only effective when solver=*sgd* or *adam*.
validation_fraction: The proportion of training data to set aside as validation set for early stopping, *default=0.1*. Must be between 0 and 1. Only used if *early_stopping* is True.
n_iter_no_change: Maximum number of epochs to not meet *tol* improvement, *default=10*. Only effective when solver=*sgd* or *adam*.

EXAMPLE

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
X, y = make_classification(n_samples=1000, n_features=40,
n_redundant=1, n_informative=20, n_clusters_per_class=2,
n_classes=2)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.01, random_state=1234)
clf = MLPClassifier(random_state=1, max_iter=300)
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
print("Predicted\t-->\t", predicted)
print("GroundTruth\t-->\t", y_test)
```

The Problem of Imbalanced Classes

The imbalanced classes problem in machine learning refers to a situation where the distribution of classes in the training data is not balanced, i.e., one or more classes have significantly fewer examples than others. This can lead to biased models that perform poorly on minority classes and result in inaccurate predictions. For example, in a binary classification problem where the target variable is *prejudice* or *no-prejudice* if the dataset contains only a few examples of "prejudice" cases, the model may be biased towards predicting *no-prejudice* cases and perform poorly on detecting messages that convey prejudice.

If a binary classifier is given 990 data points of one class and 10 of another, it is challenging to differentiate the minority class from the other class. It will most likely always classify a data point in the majority class.

The imbalance classes problem can be addressed by several techniques, including:

- Resampling methods involve either *oversampling* the minority class or *undersampling* the majority class to balance the

distribution of classes in the training data. There is a library (imbalanced-learn) in Python that implements these functionalities (https://imbalanced-learn.org/stable/user_guide.html#user-guide).

- ➢ Cost-sensitive learning: This approach involves assigning different misclassification costs to different classes. The idea is to penalize the majority class during training. Parameter in Sklearn: *class_weight=balanced* can be used for this purpose.
- ➢ Make transformation of the training data.

ACTIVITY

Students have to train and validate classifiers for the HUrtful HUmour (HUHU) task, which involves identifying messages that spread prejudice using humor as opposed to those that express prejudice in a straightforward manner. **At least two different classifiers** must be evaluated, and compared their results. The code and a report explaining the chosen classification models, the hyperparameters considered, and the results obtained should be delivered before the next session.

The Google collaborative environment (Google Colab) can be used to implement the machine learning models:
https://colab.research.google.com/notebooks/welcome.ipynb#scrollTo=5fCEDCU_qrC0