Lab3 - Evaluation and k-Nearest Neighbors

Artificial Intelligence and Pattern Recognition

In this exercise we will be focusing on model evaluation and the k-Nearest Neighbors classifier. The first part is meant to teach how to do proper machine learning evaluation using separate training, validation, and testing sets. The second part will focus on the k-NN classifier and how to adapt it to different datasets as well as to figure out when it does and doesn't work.

Evaluation

In RapidMiner you train a model by hooking data up directly to it and then by running the process you will get a fully trained (or otherwise prepared) model to use for prediction. However, often you also want to evaluate how good your model is at predicting new data. For this purpose you can use the "Apply Model" operator, that takes a trained model and a dataset as input and gives a labeled dataset as output. To see the performance of the model we can use one of the Performance operators (Validation -> Performance). Which one you use depends on what the task is. For these exercises we're doing classification, so we'll use the "Performance (classification)" operator. Then you can simply plug the output performance into the results and check how good your classifier is.

Now the question comes, what does it mean to be good?
To perform well on the data that was used for testing or to perform well on new data that was not seen before?

Task 1

- Train a k-NN models accuracy on the Iris dataset (or another dataset).
- Test that model on the data it was trained on as well as some previously unseen data (the "Split Data" operator can be used to make a previously unseen dataset)
- Which of the two accuracies are higher? Why? Which of the accuracies is most likely closer to how the model would perform as a predictive model on new data? Why?

Another aspect of training a model is to change its hyperparameters. Maybe different hyperparameters (like the value of k) perform significantly better. However, when we test many different values of the hyperparameters maybe we will find the values that are best for the test set, and not for actually new data. Do we have to make another split of the data to use for testing many different hyper parameters

Task 2

- Make another split of the data (so there are three splits in total which we will call the training, validation, and test sets)

- Change the value of k of the k-NN model, for each value tested, train the model on the training set and get the performance on both the training and the validation set
- Then take the model with the best performance on the validation set and also evaluate its performance on the test set.
- Which accuracy of the final model is most likely to be closer to new unseen data? Why? What is the problem with testing different hyperparameters directly on the test set?

k-Nearest Neighbors

The k-Nearest Neighbors classifier is one of the simplest to understand. All the data is stored in the model and to predict the class of a new data point it is simply checked which the k nearest data points are. So, if k is three and two out of three nearest data points belong to class X, then the new datapoint will be predicted to belong to class X.

Task 3

- Load the Iris dataset and visualize it using the 3D-scatterplot (put label as color and the attributes as X, Y, and Z). Does it look like k-Nearest Neighbors would get good performance on the dataset? What would be a good value for k?
- Decide on a value for k and then train and test the performance of the k-NN model.

Under this veil of simplicity, though, hides a few more complex issues: What does it mean to be near another data point? How is distance supposed to be measured? What if one attribute has values between 1 and 1000 and another has values between 0 and 1? What if one attribute isn't numerical? What is the distance between "sunny" and "rain" (typically the distance between different non-numerical attributes is 1)?

Task 4

- Load the Titanic Training dataset and split it into training, validation, and test set.
- Train and validate a k-NN model on the Titanic Training data (no hyperparameter tuning needed) and note the accuracy.
- Visualize the Titanic Training dataset using 3D-scatterplot and using a normal scatter/bubble plot (in the latter add some jitter to see how different non-numerical attributes affect the Play-label).
    - Think of ways to turn the non-numerical attributes into attributes (using the map operator)
    - Think of ways to balance the numerical attributes so that the distances used are larger where they are more relevant to survival (You can use the "generate attributes"-operator to create new balanced attributes as mathematical functions of the old ones and then use the "select attributes"-operator, to select those you want to use for training and evaluation).
- Implement these strategies and train and validate them.
- Change and update your different strategies and validate each.
- Finally evaluate the best strategy on the test set.

Extra Tasks (optional)

Manually iterating through all potential hyperparameters is a chore, thankfully RapidMiner like most machine learning frameworks has a way to automate this process with the “Optimize Parameters”-operator. There are several different strategies, but for these tasks the “Grid” version will suffice.

Task 5

- Add the “Optimize Parameters”-operator to any previous pipeline you’ve made.
    - By double clicking the operator you will be given a view of its interior, in there you can construct the training and validation steps that will be done with each parameter.
    - As input you can send anything, but the training and validation data needs to be there.
    - As output you can send anything, but the trained model and its performance are required.
    - The parameters of the operator contain an “Edit Parameter Settings”-button. Click it to decide which hyperparameters to iterate over and what values to use (for example then k-NN k value).
    - From the outside view the model output will we be the one with the highest performance and if you plug the performance into the process results you can view the performance and specific hyperparameters of the many different models tested.
- Run a training and validations over some hyperparameters and then evaluate the best model on the test set. Is there a discrepancy between the performance on the validation and the test sets? Why?

The k-NN model works by calculating the distance from one data point to another, but this means that some attributes with large value differences will be taken into account more than others, regardless of its correlation to the labels. Additionally, as the number of dimensions increase, the average distance between data points becomes more similar, regardless of their labels. This is a weakness that has to be considered during preprocessing.

Task 6

- Create a k-NN prediction pipeline for the Iris dataset.
- Use the “Generate Attibutes”-operator to add additional attributes to the dataset.
    - These attributes can be generated randomly between 0 and 1 using the rand() function, and their values can be increased with multiplication.
- Check how adding many different dimensions with different values affects the accuracy of the k-NN model. Can you make it break down completely?
- Think of (and perhaps try to implement) an automated way to balance or remove attributes that are bad for prediction. Perhaps you can come back to this task later in the course.