184.702 Machine Learning – Exercise 3

The third exercise allows you to chose from different topics, all dealing with advanced and particular topics in machine learning. The exercise shall be done in groups of 3 students, which may be the same as in the first two exercises. Please arrange with your colleagues, and then register for a group **and** for your topic in TUWEL.

If you have an interesting idea of any other topic, please don't hesitate to contact me (mayer@ifs.tuwien.ac.at) and discuss it with us. If we see it fitting to the topics of the course, you'll get our ok to work on your idea.

There is only an online submission of the report and other artefacts that you created during this exercise (code, data, ...), but **no presentation**.

The descriptions of the different topics are likely not a complete specifications, they just summarise the general idea and goals. If there is an unclarity, please do not hesitate to ask in the forum.

General comments for the exercise

This exercise is supposed to be done using an machine learning environment that (I) provides deep learning capabilities in the form of convolutional neural networks (for Exercise types like 3.1), or (iI) allows you to script your experiments, as you will need to run more setups – a GUI-only environment is likely more difficult to utilise (if you worked with WEKA until now, the WEKA API is an option).

Your submission should contain

- a zip file with all needed files (your source code, your code compiled, data sets used (but **NOT** the ones we provide to you), a build script that resolved dependencies, or include any libraries you are using. Your submission needs to be self-contained!
    - If applicable, provide a means to compile your classes, preferably with a Maven or ANT build file.
    - If your build file allows for automatically obtaining the dependencies, then they don’t need to be included, otherwise please include them.
    - Provide a short how-to explaining the way to start your program (which is the main class/file, which command-line options does it expect, ..).
    - Please also state clearly what is the version of the software package(s) you use; unless for a specific reason, please work with the latest versions (you can of course e.g. work in python 2, but don’t work with old versions of libraries etc.)
- A report, describing

    - Your task
    - Your solutions (also describe failed approaches!)
    - Your results, evaluated on different datasets, parameters, ...
    - An analysis of the results
- Where applicably, your program shall be configurable via command-line options or a configuration file, to modify parameters, evaluation types, etc… i.e. it should not be needed to modify the code to change these options. There are many framework for command-line-options available, you don't need to code this yourself (e.g. Apache Commons CLI (http://commons.apache.org/cli/, http://martiansoftware.com/jsap/ for Java).

You need to chose ONE topic, i.e. one of Topic 3.1 – 3.7.

Some of the datasets mentioned in the various task descriptions:

- AT&T (Olivetti) Faces: https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html, https://scikit-learn.org/0.19/datasets/olivetti_faces.html
- PubFig http://www.cs.columbia.edu/CAVE/databases/pubfig/
- MNIST: http://yann.lecun.com/exdb/mnist/, https://scikit-learn.org/0.19/datasets/mldata.html
- CIFAR (10, 100): https://www.cs.toronto.edu/~kriz/cifar.html, https://github.com/ivanov/scikits.data/blob/master/datasets/cifar10.py

Computing resources

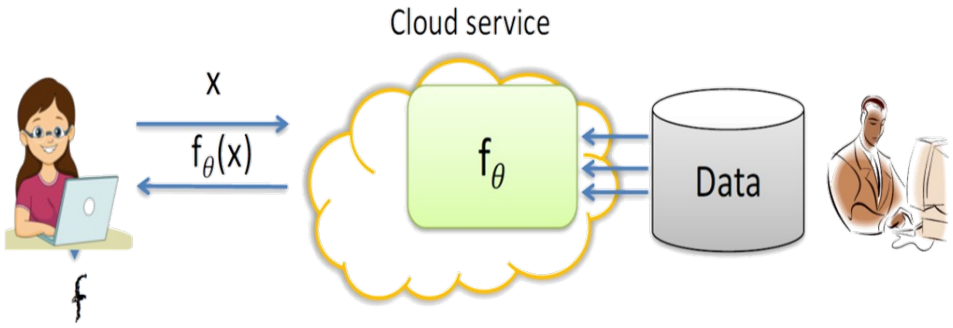
If you are in the need of computing resources, you can contact me (mayer@ifs.tuwien.ac.at) especially for CPU power on a Linux server @TU.

Further, you can try infrastructure-as-a-service offers, as some Cloud operators have some free starter credit. E.g the Google Cloud Computing platform gives you 300 USD if you register (you need to provide your credit card, but they don't charge) which you can use to create remote machines with various OS preinstalled. Also other providers have similar offers.

Topic 3.1: Adversarial Machine Learning

In this exercise, you shall pick one “discipline” of adversarial machine learning, and perform experiments with it.

Topic 3.1.1: Model Stealing (Extraction)



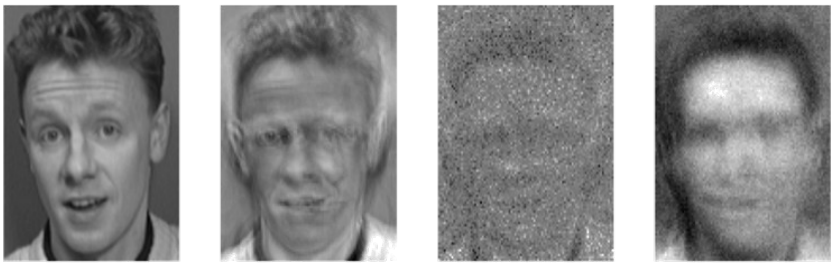
Model stealing means to obtain a trained model from a source that generally only provides a “prediction API”, i.e. a means to obtain a prediction from the model when providing a specific input, but does NOT disclose the actual model (i.e. “ML-as-a-service”). This might be because the model may be deemed confidential due to their sensitive training data, commercial value, or use in security applications. A user might train models on potentially sensitive data, and then charge others for access on a pay-per-query basis.

In model stealing/model extraction, an adversary with black-box access (but no knowledge of the model parameters or training data), aims to duplicate the functionality of the model.

In this task, you shall try to experiment with such attacks, using the code provided at https://github.com/ftramer/Steal-ML, with similar attacks as in the corresponding paper (Tramèr et al. Stealing Machine Learning Models via Prediction APIs. USENIX Security 2016. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_tramer.pdf)

Specifically, you shall experiment with these model stealing attacks for three different types of classifiers (can be from the paper) on four different datasets (have to be different than in the paper). You can either use a model available locally (as black-box, i.e. just for querying & confidence values), or one of the mentioned online service. While the model is “extracted”, record how the accuracy of the duplicated model changes, and how many queries (and how much time) it takes to train the model; plot these, in similar fashion as in the paper, for each dataset/classifier.

Topic 3.1.2: Model Inversion Attack



Given access to a model, an inversion attack tries to extract the training data used for obtaining that model. A prominent example is for a face recognition system, where the model tries to identify to which of the known users a newly taken image (e.g. from an access control camera) corresponds to. This means that each class in the dataset corresponds to one specific individual, and the training data for each class is generally a number of images of that individual (taken with potentially different angles, zoom, light, background, and appearance such as hair style, ...).

For the attack, in most cases, this means picking a specific class, and then trying to ***generate*** input patterns that return the highest confidence of being classified as that class. This normally means access to the model (also in white-box, i.e. being able to read the model parameters; potentially building on top of a model stealing/extraction attack, but you can assume to have white-box access).

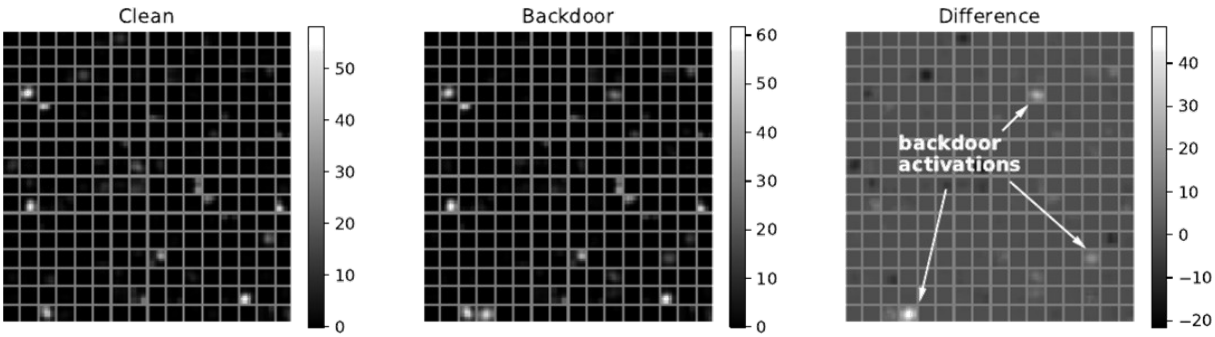
In this task, you shall re-create experiments in the style of the paper by Fredrikson et al: “Model inversion attacks that exploit confidence information and basic countermeasures” (http://www.cs.cmu.edu/~mfredrik/papers/fjr2015ccs.pdf) (or a more recent paper if you find something), but it is enough to focus on the most successful (i.e. softmax layer) approach. You can work with the same dataset, or chose a different dataset for a similar task (e.g. a subset from the PubFig dataset from http://www.cs.columbia.edu/CAVE/databases/pubfig/, which has a higher resolution and more pictures per person (but might also be with higher variance between the images))

In your evaluation, compare how well the inversion works for each of the individuals, and also evaluate what is the “cost” of generating the images – i.e. how many iterations of generating and asking the model are required, and how is the efficiency of this (in runtime). You can also evaluate the quality of the generated images by computing a numerical similarity to the target image (e.g. the most representative of the training images per class).

Topic 3.1.3: Backdoor/poisoning Attacks & defence

Poisoning/backdoor attacks modify the training data to **embed a specific pattern** in some images (e.g. a yellow block on a STOP sign), and falsely label that image as a different class (e.g. as a speed limit 50 sign) to make the classifier learn this pattern for wrong predictions. These attacks generally work because a certain number of neurons in the network can be dedicated to learn these patterns, often because the number of neurons is larger than what is required to actually represent the pattern in the “clean” training data.

In this task, you shall recreate the experiments performed using two defence mechanisms: pruning and fine-tuning, respectively a combination of both, as in the paper by Liu et al: “Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks.” (https://arxiv.org/abs/1805.12185), or you can also utilise other defence mechanisms published in the literature.



We will provide you with a number of datasets with traffic signs, which already have poisoned images prepared, so that you do not need to do this manually yourself.

You shall then evaluate, in a similar manner to the cited paper, how well you can embed backdoors on the models, and how effective the defence mechanisms work.

Topic 3.1.4: Backdoor/poisoning Attacks: CNN vs Feature Extraction

It has been postulated that Deep Learning (specifically Convolutional Neural Networks) are much more prone to poisoning/backdoor attacks (as well as evasion attacks in the form of adversarial inputs), as compared to “traditional” approaches for image classification that relied on a feature extraction step (e.g. SIFT, HOG, …) and a subsequent model such as SVMs or RFs.

However, there is rather little evidence that this is true. In this task, you shall thus compare CNNs and traditional approaches, on the same dataset that is available also for Topic 3.1.3 (traffic sings with pre-created backdoor images).

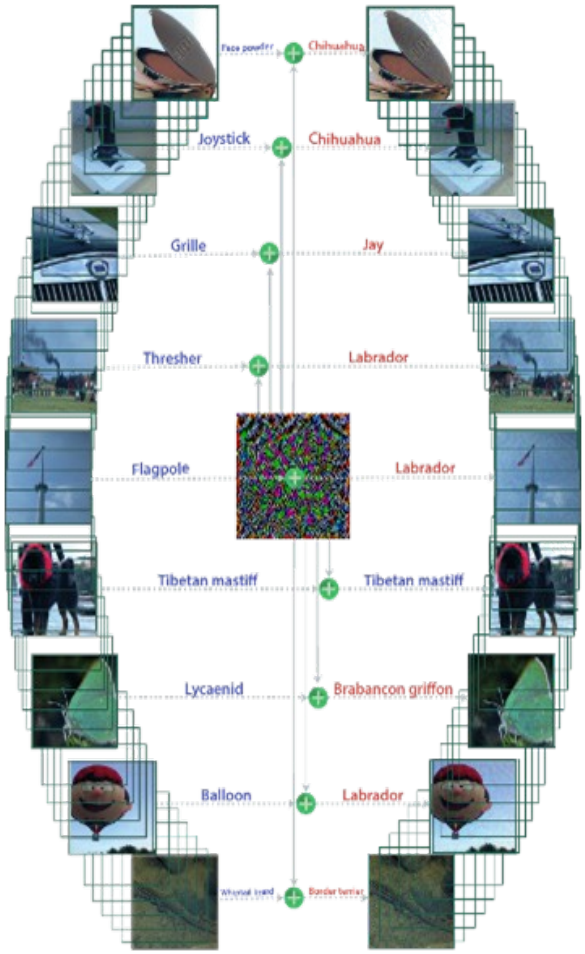
To this end, train a (simple, but depending on your compute power) CNN model that shall contain the backdoor, and evaluate the (impact on the) accuracy on clean image vs. the percentage of the model being tricked into predicting the desired target class.

Then, try the same experiment with first extracting features (similar to the ones in Task 3.8, i.e. SIFT or something equally powerul (SURF, HOG, ..)) and then learning a traditional model, such as an SVM or a Random Forest.

Then, compare the accuracy and success rate for triggering the backdoor on test samples, both from the CNN approach and the “traditional” approach.

Topic 3.1.5: Model Evasion Attack

Evasion attacks are less targeted than backdoors, as they “just” try to avoid a specific class to be predicted.



In this task, you shall evaluate, e.g. on the AT&T Faces dataset (https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html, https://scikit-learn.org/0.19/datasets/olivetti_faces.html) or a subset of the PubFig dataset, how well different methods for generating adversarial images work, by measuring the distortion needed in a metric similarity, complemented with an occasional manual inspection of the perceived difference. Also measure efficiency

of the attacks. Specifically of interest are also methods that generate universal perturbations (i.e. not specific to one instance/image).

Then, try 2 simple methods for defending against these attacks (e.g. by adversarial training). You can utilise e.g. the IBM Adversarial Robustness Toolbox (https://github.com/IBM/adversarial-robustness-toolbox), or also Google#s Cleverhans (https://github.com/tensorflow/cleverhans) or Foobox (https://github.com/bethgelab/foolbox)

Topic 3.2: Distributed Computation

Distributed computation in general addresses privacy / data protection issues by not centralising data that is initially already partitioned between several parties, and either computing functions securely w/o revealing the input (secure multi-party computation), or by exchanging only aggregated values, such as gradients or other model parameters (federated learning).

In this task, you shall evaluate the effectiveness and efficiency of these approaches. An evaluation on the quality of the models, as compared to having a model trained on all the different partitions of the data, shall be performed. - i.e. observe the effectiveness and efficiency on the centralised vs. the distributed approach.

Topic 3.2.1: Secure Multi-party computation

Utilising e.g. the library mpyc for Python (https://github.com/lschoe/mpyc) and the implementation of a secure computation for a binarised multi-layer perceptron, first try to recreate the results reported in Abspoel et al, “Fast Secure Comparison for Medium-Sized Integers and Its Application in Binarized Neural Networks”, i.e. train a baseline CNN to estimate a potential upper limit of achievable results, and then train the binarized network, as a simplified but still rather performant version, in a secure way. If needed, you can use a subset of the MNIST dataset.

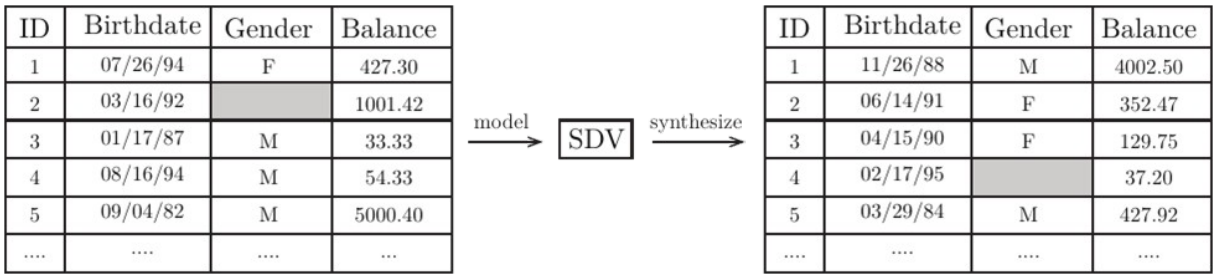
Then, try to perform a similar evaluation on another small dataset, either already available in greyscale, or converted to greyscale, e.g. using (a subset of) the AT&T faces dataset.

Topic 3.2.2: Federated learning

Utilising e.g. PySyft (https://github.com/OpenMined/PySyft), based on PyTorch, or TensorFlow federated (https://www.tensorflow.org/federated/), evaluate the effectiveness (and efficiency) of federated learning vs. the centralised learning on a small image database (e.g. AT&T faces, PubFig, CIFAR; if needed a subset of each) while learning neural networks. Specifically compare the difference in effectiveness when you vary some parameters, e.g. the number of nodes, or how often federation steps are performed.

Topic 3.3: Generation and evaluation of structured synthetic datasets

Besides data sanitisation and secure computation, a further approach to work with data that contains sensitive information is to generate synthetic data. Synthetic data would be generated by obtaining some kind of model from existing data, and then letting this model generate new instances. The "model" can be very simple, e.g. it could estimate the distribution for each variable independently, and then generate synthetic data accordingly. Or, it could be more complex models that take correlations between variables into account – if the value of the n-th attribute is to be generated, it takes into account the already generated values for the attributes (1 .. n-1). The more information is obtained from the underlying dataset, the better synthetic data is generated.



Your task is thus to develop a synthetic data set generator, and evaluate how well that works on for trainings models. Implement a very simple (independent) approach, and one slightly more sophisticated version.

You shall work on 2-3 structured datasets (you can use e.g. the adult/census income dataset as one starting point).

From each dataset, take a part and use it as training data. From this training data, generate the model, and then synthetic instances using the model. Keep one part of the original data aside as test data, and use your model (trained on synthetic data) to predict the classes of the test data.

Then compare these results with a model that you would train directly on the data used to estimate the synthetic data generator, to estimate the usefulness of the synthetic data. Provide also an analysis e.g. of the confusion matrix, to test whether one can observe differences for certain classes or instances.

Tools that you can utilise include

- https://github.com/DataResponsibly/DataSynthesizer
- https://hdi-project.github.io/SDV/
- https://cran.r-project.org/web/packages/synthpop/index.html

Topic 3.4.: Generation and evaluation of unstructured synthetic datasets

This is a rather experimental topic, but maybe therefore even more interesting. The idea is similar to the one above, but with the difference that you are not going to generate structured data, but image data. One approach could be similar to the one used in generative adversarial networks (GANs), though you can potentially utilise also a simpler approach.

Then in general, the approach shall be similar to the one for structured data – obtain a dataset of images, use a training set to learn the generator, generate data, learn a model from it, and predict the images on the test set. Compare this to the baseline of using the training set directly. You can test your approach e.g. on the car and fruits datasets linked above for the Deep Learning topic. Another evaluation would be on a medical dataset with a clear classification task.

As this topic is more experimental and novel than others, you can focus on the problem and implementation, and perform a less detailed evaluation than on the structured dataset. Also, we will honour your effort in the evaluation, and less your results.

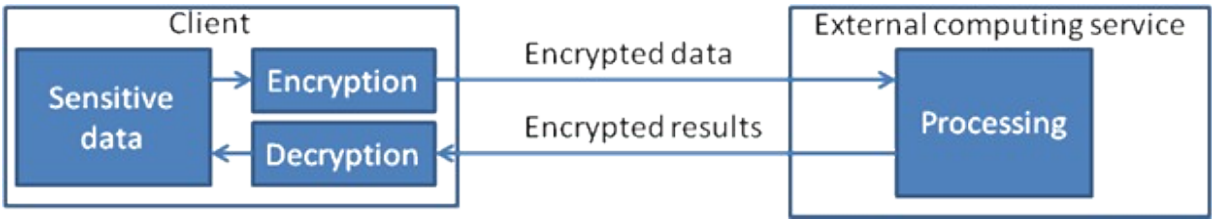
For GANs in python, plenty of examples are available, e.g. https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f

One project that you can maybe draw inspiration from the medical domain is https://www.fda.gov/MedicalDevices/ScienceandResearch/ResearchPrograms/ucm477418.htm

One data set to use (with a classification task) could be e.g. https://www.kaggle.com/c/diabetic-retinopathy-detection/data

Topic 3.5.: Machine learning on encrypted data

One approach to conceal sensitive information in data is to only provide encrypted data to the machine learning algorithm, and directly compute on encrypted data. The returned model is also encrypted, but can be decrypted with the same key as the data. Thus, only the owner of the data can utilise the model, while the computation can be outsourced to an untrusted third party.



To achieve computation on encrypted data, homomorphic encryption schemes are utilised. These will achieve operations on the encrypted data without any loss, but are normally rather slow.

You can utilise one of the existing frameworks providing HE, such as the Intel HE-Transformer, or the Microsoft SEAL Library, to implement a simple HE-based machine learning model (logistic regression, Perceptron, k-NN, Naive Bayes, or similar). This means adapting existing implementations, depending on the case exchanging computations with approximations.

You shall then compare an implementation on unencrypted data with the encrypted version. The dataset can be a small toy dataset, e.g. the IRIS dataset or even smaller.

Tool support is available, e.g.

- Intel's https://github.com/NervanaSystems/he-transformer
- Microsoft's https://github.com/Microsoft/SEAL
- A list at https://github.com/jonaschn/awesome-he
- Some others, like https://github.com/shaih/HElib or https://github.com/tfhe/tfhe

If you chose this topic, register in TUWEL which algorithm you would like to implement.

Topic 3.6: Evaluate effects of anonymisation on utility

Anonymisation is a form of data sanitisation with the goal of privacy protection, by removing information from data. Generally in a first step, before anonymisation, **identifiers** that directly reveal the identity of a person without

having further analysis of the data (such as first and last names, email address or social security number) are removed..

Quasi-identifiers do not directly identify a person (for example, birthdate/age, post code, sex, ...), but can be used in combination to restrict possibilities to such a degree that a specific identity follows logically. On the other hand, this information might hold significant information for the purpose of research. Therefore, we generalize this kind of information, which means to lower its level of granularity. As an example, one could generalize specific age, eg. 38, to a range of values, eg. 30-40, or suppress columns or entire rows.

A popular approach is e.g. k-anonymity, which ensures that at least *k* samples in the data set have the same values for the quasi identifiers, and are thus not distinguishable.

Performing data sanitisation such as anonymisation generally has an impact on the utility of the data, as information is lost.

Your task in this assignment is thus to empirically analyse the quantity of this approach. You shall utilise **two** different data anonymisation tools (and different settings therein for achieving anonymisation), on **two** datasets that have attributes that are potentially privacy relevant. You shall evaluate the impact of the anonymisation at e.g. varying levels of *k* in the anonymisation via the effectiveness results on the specific classification task you can perform with your dataset.

As your datasets, you can use e.g. the Adult (census income) data set (https://archive.ics.uci.edu/ml/datasets/Census+Income), or the larger variant utilised by the KDD challenge (https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29), or other datasets that have similar, privacy-sensitive information that needs to be anonymised.

Topic 3.7.: Large-Scale evaluation

In these exercises, you shall employ the API from an existing ML Toolkit (Python scikit-learn, R, WEKA, …) and perform a more exhaustive evaluation than in the previous exercises, and an evaluation beyond what you can obtain from manually running single experiments/parameter settings (but also more detailed than you likely get from a GridSearch implementation)

You shall persist your results (e.g. tested accuracy, precision, TP rate, …) and meta-data (name and parameters of classifiers, runtime (total, and each of training and testing), date of experiment, input data used (e.g. filename) about your experiments, in an application-independent manner; one option is a simple CSV file, another e.g. the ARFF Format, APIs for manipulating them are available in many toolkits.

As you shall do mass evaluation, you should generate tables and diagrams automatically from the result files, instead of manually compiling them.

You can re-use the data sets you used in your first exercise, as you should be familiar with them - which should help you in the evaluation. ***Mind however that you shall perform experiments with a large number of data sets, so you will need to choose additional data sets.***

Topic 3.7.1: Feature selection & evaluation

There is a wealth of feature selection algorithms, both supervised and unsupervised ones; the WEKA API gives a good overview what is available and used, such as PCA, Information Gain, .. (see http://weka.sourceforge.net/doc.dev/weka/attributeSelection/AttributeEvaluator.html for a list)

In this task, you should evaluate the effect of the different techniques, and different parameters to it. For this exercise, we will provide you with a set of data sets from the music classification domain, which (contrary to e.g. text classification) does not yet widely employ feature selection.

Develop a small framework that on a set of different data sets automatically applies different feature selection techniques. There are two different topics to choose:

a) Evaluate the number of selected features

Evaluate the effect of the desired number of features after the selection (or thresholds used in the selection technique to decide how many features are kept). Vary the number of features from very few to very high, and record and analyse the results; specifically, for each feature selection technique and each target number of features selected, record the the runtime and classification accuracies. Then analyse the chart of performance and runtime for each method on several dataset, and try to make an evaluation of a potentially optimal number (or range) of features to select. The user should be able to specify the different numbers of features, or a step-size; if none of these is provided, think of

a good strategy yourself (e.g. smaller step-size for smaller number of features, then increasing, depending on the total number of features in the dataset).

b) Compare feature selection methods

Analyse the differences in the rankings between the different methods, i.e. analyse how each individual feature is ranked by the different methods. Investigate whether there is a trend in that a certain subset of features is selected by all (or most) techniques - do they tend to select a specific subset of features, or does that vary a lot? Is there a trend in supervised and unsupervised methods?

Further, investigate whether a combination of feature selection methods is more beneficial, i.e. by taking those features that get selected by most algorithms, or by combining the top N from each algorithm, into a subset of features that you would then use for the actual training & testing.

For both topics, your program shall allow the user either to specify which feature selection techniques (and parameters) to use (via command-line, or via a configuration file), or otherwise simply use all available techniques.

Topic 3.7.2: k-nearest neighbour search optimisation

k-nearest neighbours, even though a very simple approach, yields good results in many classification tasks. One major issues is however the classification time on larger data sets, mainly stemming from the expensive (pair-wise) comparisons needed to find the nearest neighbours. In this exercise, you shall thus evaluate different optimisation strategies for this neighbour search.

A set of such strategies is for example in implemented in WEKA in combination with the IBk and LWL classifiers, and might also be available in other ML packages.

Develop an evaluation framework that on different data sets automatically evaluates the effect of applying these strategies on the runtime and performance of the algorithms, and compares the different strategies against each other, and against the "base-line" of using the classic linear (brute-force) search.

Record and analyse the following aspects: (1) how long it takes to build the search optimisation structures, (2) how the optimisation changes the time needed for the classification process, and (3) how the overall time needed (building the structure plus classifying) compares to the base line.

Try to estimate what the "break-even" point of time consumption is in regard to the size of the dataset and the dimensionality, i.e. at which combination of size and dimensionality building the search optimisation structure is equal in overall runtime than the linear search.

Further evaluate the effects on the search optimisation strategies on the accuracy of the algorithm - does that differ? Even if the accuracy stays the same, is this achieved by exactly the same data samples getting correctly predicted, or

there data samples that get classified differently by the different strategies? If so, is there a trend that the various search optimisation strategies classify the same samples differently than the linear search baseline?

Your program shall allow the user either to specify which search optimisation strategies (and parameters) to use or otherwise simply use all available strategies

Topic 3.8: Deep Learning

N.b.: if you want to work with Deep Learning, but on more advanced topics, also take a look at the task about the synthetic image data set (using generative adversarial networks) and adversarial machine learning (inversion, evasion, poisoning attacks) below.

The main goal of this exercise is to get a feeling and understanding on the importance of representation of complex media content, in this case images. You will thus get some datasets that have an image classification target.

(1) In the first step, you shall try to find a good classifier with „traditional“ feature extraction methods. Thus, pick

- One simple feature representation (such as a colour histogram, see also the example provided in TUWEL) and
- A feature extractor based on SIFT (https://en.wikipedia.org/wiki/Scale-invariant_feature_transform) and subsequent visual bag of words (e.g. https://kushalvyas.github.io/BOV.html for python), or a similar powerful approach (such as SURF, HOG, ..).

You shall evaluate them on a few algorithms (also specifically including an MLP), and parameter settings to see what performance you can achieve, to have a baseline for the subsequent steps.

(2) Then, try to use a deep convolutional neural network and see how your performance differs.

Compare not just the overall measures, but perform a detailed comparison and analysis per class (confusion matrix), to identify if the two approaches lead to different types of errors, and also try to identify other patterns.

You can base your DL implementation on the tutorial provided by colleagues at the institute, available at https://github.com/tuwien-musicir/DL_Tutorial/blob/master/Car_recognition.ipynb (you can also check the rest of the repository for interesting code; credit to Thomas Lidy (http://www.ifs.tuwien.ac.at/~lidy/)).

Also perform a detailed comparison of runtime, considering both time for training and testing, including also the feature extraction components.

The **datasets** you shall work with are

- As an easy dataset to start with, the car detection dataset (which will be utilised for the simpler „car present or not“ task, and not specifically the object localisation), from http://cogcomp.org/Data/Car, also used in the tutorial.

N.b.: As this dataset comes with (two) test sets that contains only images from one of the classes, please follow this procedure: First, combine both training and test sets into one set, and then randomly split again (either holdout, or cross-validation).

- The 30 fruits image dataset (http://www.vicos.si/Downloads/FIDS30), which is a bit harder to learn. We are providing you a slightly pre-processed version that fixes a few errors in some of the image encodings, so that you do not have to deal with that.

For the CNNs specifically

- Use an architecture of your choice – you can work with something simple like a LeNet, or a bit more advanced architectures, where maybe transfer learning is required for efficiency reasons.
- Use as well data augmentation (you can reuse the code from the tutorial), and compare it to the non-augmented results