Sigve Skaugvoll, Eirik Baug
MIT, AI
Deadline: 13.04.18

Assignment 4 - TDT4173

1 Theory

1.1) What is the core idea of deep learning? How does it differ from shallow learning? Support your arguments with relevant examples.
Deep learning neural networks allows computational composed of multiple processing layers to learn data representation with multiple levels of abstraction. Deep learning methods are algorithms that allow for automatically discovering the representation needed for detection or classification. They do this by having multiple levels of representation which in turn is obtained from composing simple non-linear modules that each transform the representation at one level, all the way from the raw input to the final output data. Each level transforms the input data into a representation at a higher and more abstract level. By combining enough such representation, very complex functions can be satisfactorily approximated. The key idea here is that higher levels of representation can amplify the aspects of the input data that we need for effective classification and suppress irrelevant variations in the features, and the feature layers are not designed manually by humans, but rather learned by the algorithms general-purpose learning procedure.

Deep learning is often used for tasks such as image- and speach classification. E.g the image is represented as an array of pixel values, and every layer attempt to extract different features from the image. The learned features of the first layer could typically represent the presence or absence of edges at particular orientations and locations in the image. The second layer could then focus on detecting motifs by spotting different arrangements of the edges and the third layer could assemble these motifs into larger combinations that correspond to parts of familiar objects. In this way different parts of the image are detected an analyzed in the different layers allowing for one or several classifications to be predicted in the final output value.

Shallow learning which is how conventional machine learning techniques attempt to approximate functions suffer from the drawback that they are limited in their ability to process natural, raw data. This means that constructing complex systems like pattern recognition systems requires a considerable amount of domain expertise from the engineers so that the system could suitably transform the raw input data into a representation the system could use to detect and classify patterns.

An example of shallow learning is for instance a case-based reasoning system for fixing cars. Programming something like this the engineer will need a considerable amount of domain knowledge. If the headlight of the car is broken the fix is likely specific to the type of car and it’s necessary to know what a headlight is and what its intended use is, making such a system hard to train using only raw input data. This does however, make the decisions made by the system much more understandable to humans as a deep learning system based on raw input data will likely have a difficult time explaining why a decision was made.

Sigve Skaugvoll, Eirik Baug
MIT, AI
Deadline: 13.04.18

1.2) Chart out comparisons between machine learning techniques: k-NN, decision tree, SVM and deep learning. Discuss, individually, about each of the techniques mentioned above when and why one should NOT preferred over the other.

Traits	**k-NN**	**Decision tree**	**SVM**	**Deep learning**
Easy to implement	Yes	Yes	Yes	No
Need a lot of domain knowledge	Yes	Yes	Yes	No
Unstable learning	No	Yes	No, but can be couple of first rounds.	Yes
Unsupervised learning	Yes	No	Yes	Yes
Gives reason to predictions | classifications	No	Yes	No	No
Can be parameterless	No	Yes	No	No

K-NN

k-NN is an easy to use clustering algorithm. It does, however have the drawback of needing to know exactly how many clusters or different classifications are needed as this is something that is set before running the actual algorithm. This algorithm is not preferable when working with very complex problems where deep learning is a much stronger choice as it works better with raw natural data where we know nothing about the amount of clusters needed.

k-NN is also required to have some sort of distance function as it calculates the nearest neighbours. In instances where such a function is very hard or impossible to make this algorithm should be avoided. In classification problems with lots of different variable types, where a decision tree is strong, k-NN will find it hard to classify correctly. k-NN also doesn't handle soft boundaries which is when cases appear on both sides of a boundary.

It should also be noted that k-NN as well as SVM and deep learning algorithms generally does not give reasons as to why predictions and classifications were made in a certain way. For complex classification problems only a calculated output score is given. If you want a

system with proper explanations of the classifications something like a decision trees are preferred.

Decision tree
Decision trees are among the easiest method to read and understand for humans. But its complexity is time and cost consuming. A large tree with many branches requires a lot of resources. This is because it searches through a lot of hypotheses. Decision trees are not very good at regression and continuous values and one should then consider another method if possible. It also does not see relationship between features: If there are many relationships and abstract information one should perhaps choose deep learning algorithms and methods - one such example could be image classification.

As noted decision tree classification can become complex. As the problem space increases the calculations and memory needed increase exponentially along with the depth of the tree. In addition, a decision tree is usually built recursively so the recursion limit of the system will limit the depth of the tree. Generally, decision trees does not scale very well and should be avoided for very complex classification problems with a very high number of classes.

SVM
SMV attempts to maximize the distance between the “separating line” or “decision boundary” to its margins. What this means is it attempts to find the maximum distance between our decision boundary between all cases/classes as much as possible as it tries to find an optimal separating hyperplane which categorizes new examples. SVM has drawbacks such as hyper-parameter tuning. This is the key to getting good performance from SVMs which gives certain requirements to the “applier”. He or she needs to know the problem, there is a lot of trial and error to find the correct hyper-parameters. Decisions tree on the other hand does not need hyper-parameter training. Thus SVM should be avoided when there is little information and knowledge about the problem, because improper tuning can be deadly for the SVM. if one tries to run SVM with a linear kernel for a non-linear problem and wrong parameters would be cost and time consuming and not give good results. Thus one should use algorithms that require less hyper-parameter setting or have self-learning parameters and a non-linear kernel if one is insecure about the data and there is no way of checking. One example is that the problem can be solved with higher accuracy using i.e Random forest within minutes and a improper SVM can run for days without good results or converging. So choosing a less complex method such as decision trees can sometimes be a better choice.

Deep learning
Deep learning has the drawback of probably being the most complex of the mentioned methods. It has the property of being very useful at complex feature learning, but at the cost of unnecessary complexity and computation for some problems. The computational complexity increases exponentially with the number of layers and nodes in a fully connected neural network unless certain optimization techniques are used, so this methods should generally be avoided for simple classification problems. This is also true for when the data set is small, because it is unlikely to outperform other techniques when this is the case.

Lastly, it can be very difficult to determine the correct parameters for a deep learning algorithm as this is a field in itself and a lot of guesswork is involved.

Generally, deep learning excels at really complex problems, but for simpler problems with little data it is not very efficient and the same accuracy can be found using less complex methods such as k-NN for simple linear classification problems.

1.3) Discuss when should we use ensemble methods in context of machine learning? Explain briefly any 3 types of ensemble machine learning methods.

When we should use ensemble methods

Using assemble methods one can reduce random errors because its more accurate to use multiple methods and their decisions than random guessing, this means that random errors cancel each other out, and correct decisions are reinforced. One would perhaps benefit from using assemble methods when the training data is too small and many hypotheses satisfy it, many local minimas, which is hard to get out of, and it can possibly find a better representation of the problem, than a individual method could. Also, ensemble methods can and usually are better at suited to handle missing values in datasets.

Bagging method (bootstrap aggregating)

Bootstrapping is useful when one needs a reliable average and knowing the error of the average and it's impossible to get multiple samples from the training set. One then creates a new sample from the one sample chosen from the training set and calculates some values that are used for all the quantities of interest. One should avoid using bagging when the data set is small and the original sample is not a good approximation of the population. Outliers can add variability in our estimates, that's why it's not good to create samples from outliers and very small datasets. Bagging is not suitable when there exist dependencies in the structure, because bagging is based on the assumption of independence. The algorithm combines different predictors by either voting, averaging, etc. The predictors are trained using a bagging sample.
Bagging is usually used for improving the stability, reduce the variance and increase accuracy.

Boosting method:

The boosting method is a method which train classifiers in a sequence, where the next classifier focus on those cases which were incorrectly classified by the previous classifier. Then all the classifiers vote on the final prediction.
Each data case is weighted and wrong classifications gives high focus to the case, to let the next algorithm find and focus on these. Each boosting round learns a new classifier on the weighted dataset, and all these classifiers are combined into one. Boosting method is usually good for reducing bias and also variance in supervised learning.

Decision tree and random forest

For introduction to decision trees see task 1.2 §Decision Tree.
Random forest can be used for both classification and regression. Random forest, combines multiple decision trees, thus given the name forest. The general idea is the more trees, the

more robust and higher accuracy. Random forest grows multiple trees, to classify a new object each tree in the forest gives a classification or votes for a classification to give the new object. Then the random forest method chooses to give the object that most voted classification. This allows to handle missing values and still maintain accuracy. Another advantage is that there is less probability to overfit the model, when we build and depend on many different trees. It also gives advantages over simply using one tree, because it can support larger datasets and higher dimensions / more attributes. Random forest algorithm or method selects a set out of N training cases, at random, but with replacement. After that it select a set of the m <M features / attributes, where M is the total number of features. It does this at each node, and the m (set of features) are selected at random. Each node selects the same number of attributes for the selected feature set. Each three is grown to its largest and there is no pruning. Then a new object is predicted by each tree in the forest, and then each tree votes for what they predicted. the prediction with highest votes, wins and thus the new object gets the winning prediction.

Sigve Skaugvoll, Eirik Baug
MIT, AI
Deadline: 13.04.18

1 Programming

K Nearest Neighbor

- In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k*_nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.
  - **USES** : weighting scheme (inverse distance weighted average) consists in giving each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor

- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

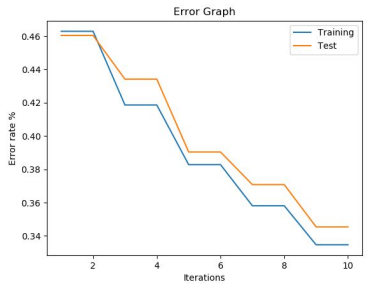
Print Format: Nearest neighbor, (dist, (feature1,..,fn, y))

```
CLASSIFICATION STARTS HERE

nn (0.0, (6.3, 2.7, 4.9, 1.8, 2.0))
nn (0.17320508075688762, (6.2, 2.8, 4.8, 1.8, 2.0))
nn (0.24494897427831774, (6.3, 2.5, 5.0, 1.9, 2.0))
nn (0.3605551275463989, (6.1, 3.0, 4.9, 1.8, 2.0))
nn (0.36055512754639907, (6.3, 2.5, 4.9, 1.5, 1.0))
nn (0.3741657386773937, (6.3, 2.8, 5.1, 1.5, 2.0))
nn (0.41231056256176557, (6.0, 2.7, 5.1, 1.6, 1.0))
nn (0.4242640687119281, (6.4, 2.7, 5.3, 1.9, 2.0))
nn (0.4358898943540672, (6.0, 3.0, 4.8, 1.8, 2.0))
nn (0.4795831523312724, (6.5, 2.8, 4.6, 1.5, 1.0))
The sample: (6.3, 2.7, 4.9, 1.8) is predicted class: 2.0
The actual class is: 2.0 and predicted is: 2.0
Thus the prediction is True

REGRESSION STARTS HERE

nn (0.0, (6.3, 2.7, 4.9, 1.8))
nn (0.17320508075688762, (6.2, 2.8, 4.8, 1.8))
nn (0.20000000000000018, (6.3, 2.5, 4.9, 1.5))
nn (0.2236067977499782, (6.3, 2.8, 5.1, 1.5))
nn (0.22360679774997896, (6.3, 2.5, 5.0, 1.9))
nn (0.3000000000000001, (6.1, 2.8, 4.7, 1.2))
nn (0.3464101615137755, (6.1, 2.9, 4.7, 1.4))
nn (0.3605551275463984, (6.0, 2.7, 5.1, 1.6))
nn (0.3605551275463989, (6.1, 3.0, 4.9, 1.8))
nn (0.3741657386773947, (6.5, 2.8, 4.6, 1.5))
The sample: (6.3, 2.7, 4.9) is predicted class: 1.9
```

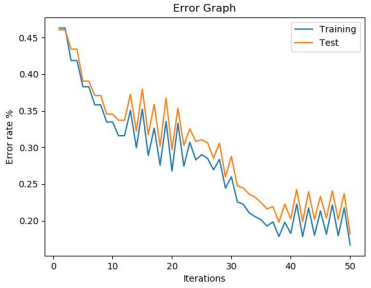
Sigve Skaugvoll, Eirik Baug
MIT, AI
Deadline: 13.04.18

AdaBoost

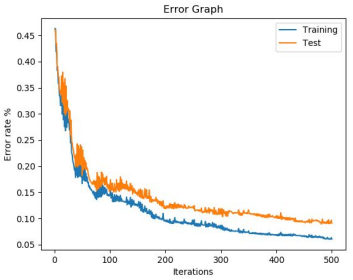
— = Test error,
— = Training error



ACC: 65.46% -- ERR: 34.54%



ACC: 81.79% -- ERR: 18.21%



ACC: 90.92% -- ERR: 9.08%

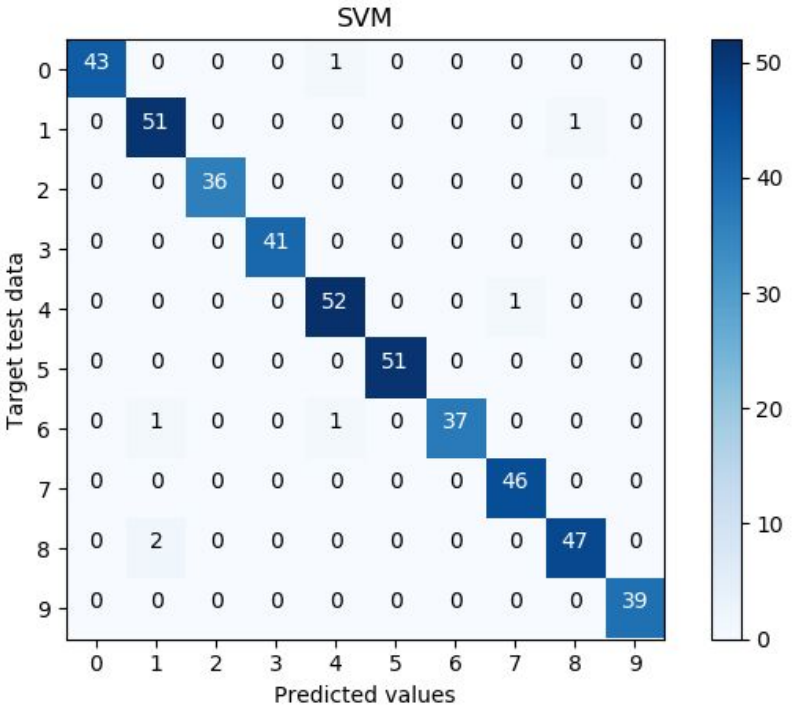
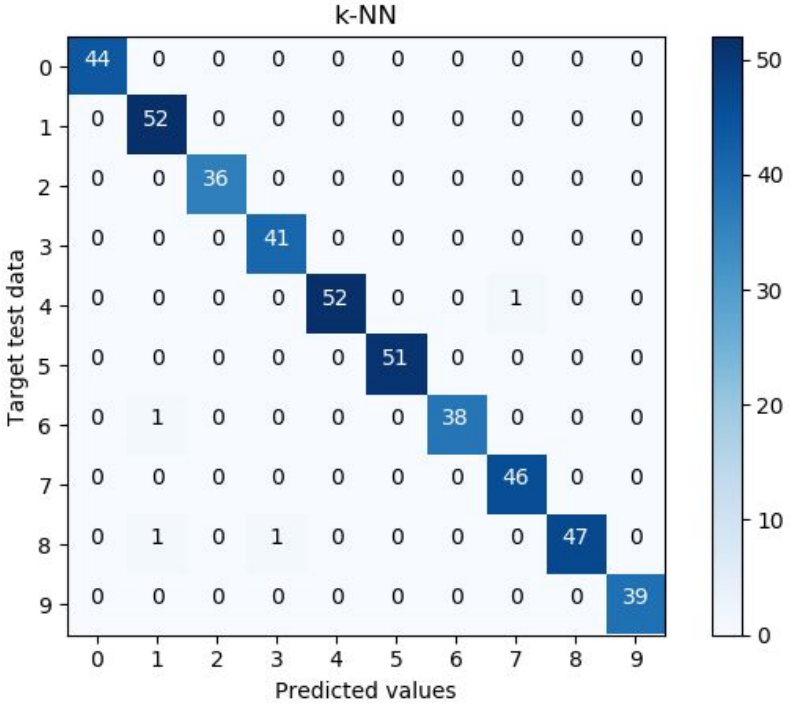
We run adaboost for 10, 50 and 500 iterations with `DecisionTreeClassifier(max_depth=1)`, thus it combines 10, 50, and 500 weak learners which should give a reasonably strong classifier, which corresponds with the results above.

```
#1 -- Accuracy: 0.5370833333333334
#2 -- Accuracy: 0.5039583333333333
#3 -- Accuracy: 0.5465625
#4 -- Accuracy: 0.5039583333333333
#5 -- Accuracy: 0.5364583333333334
#6 -- Accuracy: 0.5039583333333333
#7 -- Accuracy: 0.5339583333333333
#8 -- Accuracy: 0.5039583333333333
#9 -- Accuracy: 0.5404166666666667
#10 -- Accuracy: 0.5039583333333333
```

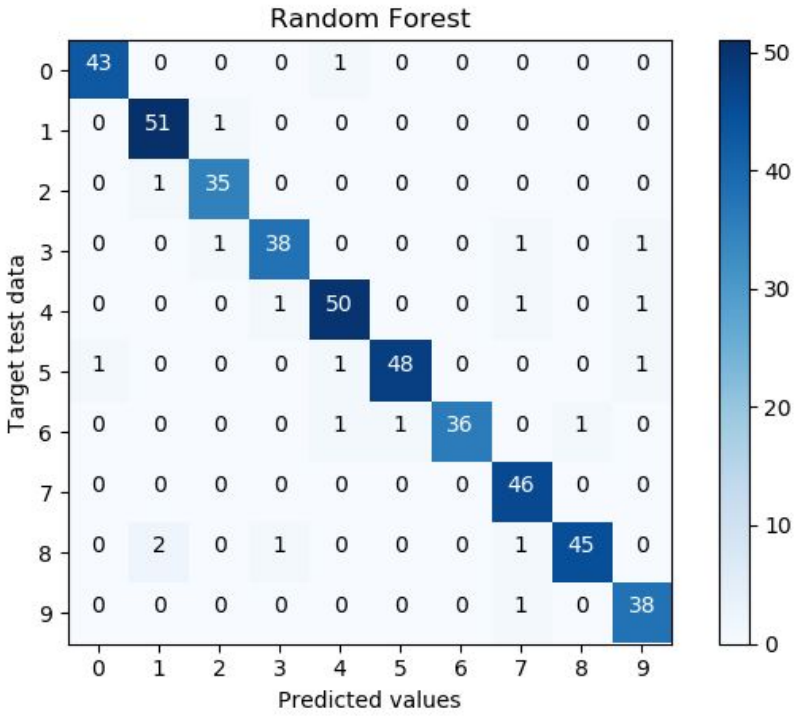
When looking at the accuracy of the 10 weak classifiers individually, we can see that each classifier has approximately 50% accuracy, but combined they have a accuracy of ~65%. When we run with 50 and 500 iterations - thus 50 and 500 weak classifiers, each classifier still has only around 50% accuracy, but the total accuracy increases to ~82% and ~91%. Thus we see that combining many weak classifiers and letting them vote significantly increases the accuracy. Thus adaboost becomes a very good and adverse method, because one can combine different classifiers such as k-NN and DecisionTrees, one can also train different parameterized version of different classifiers and combine them.

Sigve Skaugvoll, Eirik Baug
MIT, AI
Deadline: 13.04.18

2.3) k-NN vs SVM vs Random Forest on sklearn digit data set

	Confusion matrices:
SVM	
k-NN	

Sigve Skaugvoll, Eirik Baug
MIT, AI
Deadline: 13.04.18

Random Forest	

	Classifier signatures:
SVM	`SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)`
k-NN	`KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')`
Random Forest	`RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)`

The plotting code for the preceding confusion matrices is taken from
http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

Sigve Skaugvoll, Eirik Baug
MIT, AI
Deadline: 13.04.18

Basically, the numbers in the diagonal of the matrices are the number of correctly classified values of that label (e.g. target test data of 9 is predicted to be 9, 38 times during the random forest algorithm run.

In this task I simply chose a random_state variable for all test runs of 101 to get the same dataset split every time (the number itself was randomly chosen). The test data constitutes of 25% of the total dataset.

Not a lot of exciting data was found, except that all the classifiers, as indicated in the matrices, classify the test data really well as long as the parameters are correct. The classifier functions from the Sklearn library used in this task worked very well for all the classifiers with the default function values except for the SVM library which has a default kernel function of RBF. This gave very poor predictions, but changing to a linear kernel functions seemed to work very well.

Overall the classifiers are more or less equally effective on this problem with very minor variations in the predictions.