Take home exam
DAMI_II
Davide Lagano

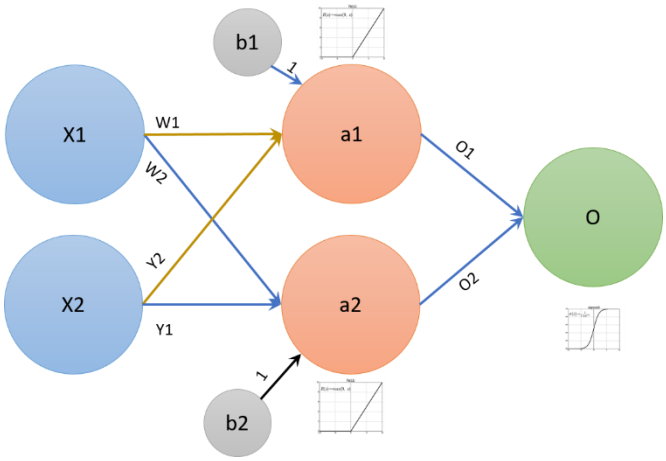
25/01/2020

1a) Variance: measures the sensitivity of an algorithm about a specific training data set. A high variance value indicates an excessive adaptation to the distribution of training data, and for this reason, it is also said there is overfitting.
Bias: is the opposite concept of the variance; it measures the inability of an algorithm to represent and therefore adapt well enough to the data distribution of a training set.
In other words, the bias gives weight to the simplifications adopted to make a target function easy to model and quick to train. However, the drawback of it consists in not being able to find the most representative function of the distribution. In these situations, the algorithm will have high bias values, causing underfitting.
The bias-variance tradeoff is an interesting dilemma. On the one hand, if the algorithm adapts too well to the data, and is therefore overfitted, it will have a low bias but a high variance. This will result in a limited ability of the algorithm to generalize the distribution learned during training.
On the other hand, if an algorithm is too simple to represent the distribution of training data, and is, therefore, underfitting, it will have a low variance and high bias values, which will result, perhaps, in better generalization performance than overfitting, but probably also in a representation of the real distribution so loose that it is not sufficiently representative.
The ability to develop an algorithm that manages to find the right balance between the bias and variance values is exactly our mission: the simplest approximation model with the best performance in terms of generalization (= with the lowest generalization error).

1b) There are different techniques to solve the dilemma between bias and variance:
Ensemble models:
• Bagging: Reduce the variance without increasing the bias. The idea is to use a set of models that decreases the variance of the average model (for the central limit theorem). The problem is that we only have one training set. The solution is to implement the bootstrap, that consists of doing random sampling on the training set to create a set of new training sets and learn about each of them.
As a result, we obtain a set of models.
• Boosting: Reduce the bias without increasing the variance. Instead of learning a very precise model, we use a set of simple models (weak learner) sequentially and combine their performances.
Each simple model does little better than a random guess; together, they can give a good performance. An example is Adaboost.
Idea: we combine the decision-making power of many models to create one (ensemble methods).
• Stacking: in bagging, we combine estimates of the same learning algorithm on different data sets generated by resampling, whereas in stacking, we combine estimates of different learning algorithms on the same data set. Stacking also helps to decrease variance and bias, but it is also efficient at preventing overfitting.

1c) How the bias/variance trade-off can be used to diagnose the performance of machine learning models:
The algorithms that has low variance, so high bias, are less complex. This algorithms train steady models that are not accurate on average. These models can be parametric or linear, e.g. Naive Bayes or Regression.
In the other side, we have algorithms that have high variance, so low bias, are much more complex. They train variable, but precise models on average. Typically, these algorithms are non-parametric or non-linear, e.g. NN or decision tree. The bias-variance trade-off reflects the complexity of an algorithm. We cannot find an algorithm that is more and less complex at the same time.

2a) We can consider a neural network as a black composed from at least three layers:
- An input layer, or the input data;
- Hidden layers: we can have one or more of them, it represents the place where the processing takes place;
- An output layer, containing the final result.



The neural network is made up of neurons, arranged in successive layers. Each neuron is typically connected to all neurons in the next layer via weighted connections. In this process, we also consider the number of connected neuron because the neurons are multiplied by them.
Every neuron adds up the weighted values of all neurons connected to it and adds a bias value. An "activation function" is applied to this result, which does nothing but mathematically transform the value before to pass it to the next layer.

In this way, the input values are propagated through the network to the output neurons.
Each node is connected to all the nodes of the next layer. In the algorithm, these connections are "weighed" by multiplicative factors, which represent the "strength" of the connection itself.
There are several activation functions:
• Sigmoid function: has an S-shaped trend. We can identify different advantage of the sigmoid function: it is differentiable; it compresses every value in a number between 0 and 1 and to be therefore very stable even for large variations in the values.
This function has a very slow convergence. Furthermore, as it is not zero-centred, the values in each learning step can only be all positive or all negative, which slows down the training process of the network. Function no longer widely used in intermediate layers, but still very valid in output for categorization tasks. Cons of the sigmoid activation function is that the exp() is computationally expensive and 'Saturated neurons kill the gradient update' (slide 12 lecture 6).
• ReLU function: very simple function to calculate and also fast to compute: flattens the response to all negative values to zero and leaves everything unchanged for values equal to or greater than zero.
This simplicity makes it a particularly attractive function in the intermediate layers, where the amount of steps and calculations is important. Calculating the derivative is very simple: for all negative values it is equal to zero, while for positive ones it is equal to 1. At the angular point in the origin, however, the derivative is undefined but is still set to zero by convention. It is not just fast to compute, but it also converges faster than the sigmoid function. If the sigmoid has problem with the saturated neurons, ReLU function does not saturate the gradient in the positive region; it is also really simple to interpret. Drawbacks of it are also that it is not zero-centred, the gradient for x = 0 is not defined.
• Leaky ReLU: Leaky ReLU, for negative values has a slope, rather than zero. Two advantages of Leaky ReLU: it solves the issue of "dying ReLU," since it has no zero-slope sections. The training is also speeding up; the "mean activation" close to 0 makes training more rapid. Leaky ReLU, compared to ReLU, is more "equilibrated" and can learn faster.
It has the same pros and cons of ReLU, but its negative weights does not die.

2b) Already spoke in '2a' about the pros and cons of these activation functions.

2c) A neural network without an activation function is simply equivalent to a regression model. Without an activation function, in this model, practically every layer (if they are 1 or 1000 does not change anything) would behave in the same way as the previous one. The purpose of neural networks is to be able to

approximate any function. In order to do this, it is necessary to introduce a nonlinearity factor, hence the activation function.

3a) Dynamic Time Warping (DTW) is an algorithm that measures the similarity between two timelines that can vary in speed. For example, similarities in walking could be detected using DTW, even if one person walked faster than the other or if there had been accelerations and decelerations during an observation. It can be used to combine a sample voice command with a command from others, even if the person speaks faster or slower than the pre-recorded sample voice. We can use DTW in different fields as audio file, temporal sequences of video and graphic data; in fact, any data that can be transformed into a linear sequence can be analyzed with Dynamic Time Warping.
In general, Dynamic Time Warping is a method that calculates an optimal match between two given sequences with some restrictions. Let us say we have two sample and test voice sequences, and we want to check whether these two sequences match or not. Here the voice sequence refers to the converted digital signal of your voice. It could be the breadth or frequency of the voice that denotes the words said.
Dynamic Time Warping allows us to align two sequences, even with different number of observations, by calculating their distance. In general, the DTW allows us to find an optimal alignment between two signals through their non-linear warp concerning the time variable.
We can conclude by saying that these are algorithms capable of comparing two sequences of temporally scanned events, finding the similarities between them regardless of the speed at which the events follow one another — finding optimal global alignment between two time series, exploiting temporal distortions between them.
Euclidean distance uses the distance between each pair of the time series and compares them using the Euclidean equation. It requires that two generic time series have the same number of observations. On the other hand, DTW allows to expand or compress the signals concerning time, redefining them in a comparable common space, in order to create the best possible alignment between two time series.
Euclidean distance does not consider the interdependence between the observed values and can match the point just sequentially, so the first point of Q must be matched with the first point of X and so on.

3b) Dynamic programming: Considering two time series Q and X, we create a matrix, and each cell in the matrix corresponds to two points of the time series (one cell correspond to one point of Q and one other of X). We want to be allowed to have a sneaky path; to find the sneaky path with the smallest distance; to find a way to shift the points in one time series to match the other. The dynamic warping computation will find it.
The diagonal (Euclidean) path does not have the smallest distance because it just follows the diagonal.
What happens in the matrix? For example, the first two points match each other, then the second point of X match with the second, third point of Q. Then the third point of X match with the fourth, fifth, sixth and so on. Every time we see the path prolonging across Q, it means that X is stretch, every time we see the path prolonging across X, it means that Q is stretch.
Computationally: we follow each cell and for each cell (i,j) we compute the difference between the two squared points and then check the points in the diagonal, below it and in the side; then propagate the point that has the smallest distance. It means that instead of moving to the next point, you can stretch one of the two time series and get a smaller cost.
Drawback: you assume that the first point of Q is mapped to the first point of X, and the last point of Q is mapped to the first point of X. We also assume there is no gap in the path, so every point needs to be mapped to a point; you cannot skip one point.
The path needs to go from left to right and from bottom to top because otherwise, it will allow points to crossover.

3c) Lower bounding is useful because it allows us to speed up similarity search by using a lower bounding function. Lower bounding function always give a value that is either the true value of the distance between the two time series or something smaller, so it will always underestimate the distance and do it faster.
This concept is applied to the NN search. The algorithm would be simple: We have a 'best', that initially is infinity, I am looking for the smallest. Then we can take the query, look at the first time series or first object

and compute the lower bound (because it is faster). If the lower bound is less then the best value we have so far, then we need to check the actual distance. If also the actual distance is better than the best so far, we have found a better answer and set this value as the best value. If when we check the lower bound, it is more then the best so far, we skip that particular object and move to the next.
Two measures for measuring the effectiveness of a lower bound are:
LB_Keogh: we take the query time series Q, and we are going to create two envelopes, the upper U and the lower L. These envelopes will differ a little bit, compared to Sakoe-Chiba Band or Itakura Parallelogram. How are the envelopes created? Is quite simple: for the upper envelope we look at the values of the time series: R point before and R point after that, and we take the max value of this range; this is the value high for the upper envelop. After this, we keep slight the window of -R, +R range, and for each point, we just come up with the max value. The procedure for the lower envelope is the same, but we take the minimum values. For Sakoe-Chiba Band and Itakura Parallelogram, the envelops are different because Itakura Parallelogram is tighter at the beginning and the end but become larger in the middle. Sakoe-Chiba Band is 'constant'.
LB_Keogh: we have the query time series, we compute the envelope. It look at the time series, and for all points are outside the envelop (above the upper envelop or below the lower envelop) it computes the distance to its closest point in the envelope. Everything is inside the envelope, it ignores them and add 0 to the cost. Pretty similar to LB_Yi, but in LB_Yi we would look at the global max and global min time series, here we look at the local max, local min across the whole time series.

---

4) Based on the dataset http://timeseriesclassification.com/Resamples.csv
The theory of this question was taken from the slides of lecture 4.

4a) Null-hypothesis H0: there is no difference between the mean of BOSS and HIVE-COTE.
Alternative hypothesis H1: the mean between BOSS and HIVE-COTE is different.

Given two matched samples, we want to test whether the difference in mean performance between these classifiers are different. We want to test whether the two samples come from the same distribution.
In order to do it, we use the t-statistic:

$$t = \frac{\bar{d}}{\frac{\sigma_d}{\sqrt{n}}}, \text{where } \sigma_d = \sqrt{\frac{\sum_{i=0}^{n}(d_i - \bar{d})^2}{n-1}}$$

- $\bar{d}$ is the difference between the mean performance of h_1 and h_2
- d_i is the difference between h_1 and h_2 for trial i.
- n is the number of trials.

In order to execute this test, we follow several steps:
n = 85.
$\bar{d}$: we need to calculate the mean performance of HIVE-COTE minus the mean performance of BOSS. We calculate a simple mean between all the 85 values of both columns, and then we calculate the difference between them. The result is -0.03569.
d_i: then we need to calculate the difference between HIVE-COTE and BOSS for every trial i, so we simply compute the difference between the first value of HIVE-COTE with the first value of BOSS, then the same for all the other values.
σ_d : in order to compute this value, we divided the process into more steps:
1) $\bar{d}$ - 1° value of d_i and we repeat this operation for every value of d_i.
2) Sum all the value of the previous step = 0,15143.
3) SQRT (result of the step 2/(n-1) = 0,042458.

At the end, we need to calculate t = $\bar{d}$ /(result of the step 3/SQRT(n)) = -7.7490.

The degree of freedom for the t-test is n - 1 = 84. Therefore, the null hypothesis can be rejected at the 0.001 significance level (the t value we obtain has to be greater than 2.37156369 for that to be possible).

We reject the null hypothesis because 7.7490 > 2.37156369.

b) For multiple datasets and multiple algorithms, we compute the non-parametric Friedman's test that is a multiple-hypothesis test.
Null-hypothesis H0: there is no significant difference in rank between BOSS, HIVE-COTE, Flat-COTE, LS, DDTW_R1_1NN and TSBF.
Alternative hypothesis H1: exist at least one pair of classifiers with significantly different performance.
In the case of alternative hypothesis H1, the test is followed by a Post-hoc test that identifies significant different pairs of classifiers.
The dataset is composed by n=85 and k=6.

```
In [10]: df1.head()
```

Out[10]:

	Unnamed: 0	BOSS	HIVE-COTE	Flat-COTE	LS	DDTW_R1_1NN	TSBF
0	Adiac	0.749412	0.815396	0.809847	0.527418	0.583043	0.726777
1	ArrowHead	0.875200	0.887657	0.876800	0.841314	0.867600	0.800857
2	Beef	0.615000	0.722667	0.764000	0.697667	0.533000	0.554333
3	BeetleFly	0.948500	0.959000	0.921000	0.861500	0.811500	0.798500
4	BirdChicken	0.984000	0.950500	0.941000	0.863500	0.877500	0.902000

As a first step, I read the dataset in Python and keep just the columns I need (there are just five rows because I visualized just the first part of the dataset).

```
df1_ranked=(df1.rank(axis=1,ascending=False))
df1_ranked
```

	BOSS	HIVE-COTE	Flat-COTE	LS	DDTW_R1_1NN	TSBF
0	3.0	1.0	2.0	6.0	5.0	4.0
1	3.0	1.0	2.0	5.0	4.0	6.0
2	4.0	2.0	1.0	3.0	6.0	5.0
3	2.0	1.0	3.0	4.0	5.0	6.0
4	1.0	2.0	3.0	6.0	5.0	4.0

Then I ranked the cells based on the values of each row (there are just five rows because I visualized just the first part of the dataset).

```
mean
```

```
BOSS           3.276471
HIVE-COTE      1.364706
Flat-COTE      2.341176
LS             4.341176
DDTW_R1_1NN    5.405882
TSBF           4.270588
dtype: float64
```

Moreover, I calculated the mean of each column in order to obtain R_j.

Then, based on the formula $$\chi_F^2 = \frac{12n}{k(k+1)} \times \left(\sum_{j=0}^{k} R_j^2 - \frac{k(k+1)^2}{4} \right)$$ I calculated X^2_F. The result is 264,37624.

Finally, based on the formula $$F_F = \frac{(n-1)\chi_F^2}{n(k-1) - \chi_F^2}$$ I calculated F_F. The result is 138,25853.

F-distribution with k - 1 and (k - 1)(n - 1) degrees of freedom, gives a critical value for alfa = 0.05 as 2.23547696.
Since 138,25853 > 2.23547696, the null-hypothesis can be rejected.

Now we are going to use the Nemenyi test because Friedman's test reveals that there is a significant difference among the algorithms.

Based on the formula $$CD = q_\alpha \sqrt{\frac{k(k+1)}{6n}}$$ I calculated CD = 2,85*0,2870=0,8179.
If the average difference between any pair of classifiers are larger than this value, we can reject the hypothesis that they have the same rank.

DDTW_R1_1NN - LS = 1.0647	> 0.8179, we reject
DDTW_R1_1NN - TSBF = 1.1353	> 0.8179, we reject
DDTW_R1_1NN - BOSS = 2.1294	> 0.8179, we reject
DDTW_R1_1NN - Flat COTE = 3.0647	> 0.8179, we reject
DDTW_R1_1NN - HIVE COTE = 4.0412	> 0.8179, we reject
LS - TSBF = 0.0706	< 0.8179, we cannot reject
LS - BOSS = 1.0647	> 0.8179, we reject
LS - Flat COTE = 2.0	> 0.8179, we reject
LS - HIVE COTE = 2.9765	> 0.8179, we reject
TSBF - BOSS = 0.9941	> 0.8179, we reject
TSBF - Flat COTE = 1.9294	> 0.8179, we reject
TSBF - HIVE COTE = 2.9059	> 0.8179, we reject
BOSS - Flat COTE = 0.9353	> 0.8179, we reject
BOSS - HIVE COTE = 1.9118	> 0.8179, we reject
Flat COTE - HIVE COTE = 0.9765	> 0.8179, we reject

References:

1b) https://books.google.it/books?id=i8hQhp1a62UC&pg=PT136&lpg=PT136&dq=Stacking+bias+variance&source=bl&ots=90pgCsczaL&sig=ACfU3U11xwV_e0Ts3L-j8qnRLxDKDMLbnw&hl=it&sa=X&ved=2ahUKEwizmoypg5XnAhVHqxoKHXP3D50Q6AEwEXoECAkQAQ#v=onepage&q=Stacking%20bias%20variance&f=false

1c) https://elitedatascience.com/bias-variance-tradeoff

2a) https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

3a, b, c) Slides: Time series -Panagiotis Papapetrou, PhD, Professor, Stockholm University, Adjunct Professor, Aalto University.

4a, b, c) Research topics in Data Science, Research methods in Data science, Isak Samsten, November 13, 2019 Lecture 4.