Handout on basic machine learning with solutions

The following exercises should help fix some topics presented during the lessons, without being in front of a computer and possibly together with some colleague and some coffee. With one or more coffee symbols are indicated the exercises that may require some more time/thought.
Please write to aris.marcolongo@gmail.com if you find errors in the exercises or for discussion.

1 Classification metrics

1. Precision and recall in binary classification.
In binary classification it is custom to distinguish a positive (1) and negative (0) classes. Precision and recall metrics are often defined in this setting with respect to the positive class, especially when this is the minority class. You are given the following confusion matrix:

$$C = \begin{pmatrix} 26 & 16 \\ 33 & 25 \end{pmatrix}$$

Compute in this binary setting the precision and recall, where the rows/columns report results for the two classes [0,1], in this order. Try not to use the formulas given in the class, but to remember the meaning of precision and recall and 'rederive' those formulas.

SOLUTION EX 1.
Precision answer to the question: "among all predictions done for of positive samples, how many were correct?". In this case:

$$\text{precision} = \frac{25}{16+25} \sim 0.61$$

Recall answer the questions: "among all positive samples in my dataset, how many was I able to classify as positive?". In this case:

$$\text{recall} = \frac{25}{33+25} \sim 0.43$$

2. Aggregated metrics in multiclass classification (☕).
In a multiclass framework you are given the following confusion matrix:

$$C = \begin{pmatrix} 11 & 11 & 9 \\ 9 & 9 & 11 \\ 14 & 10 & 16 \end{pmatrix}$$

where the rows/columns report results for the three classes [0,1,2], in this order.

- Compute precision and recall for each class. Report the mean of precision and recall over all classes. This way of combining individual metrics to return a final one is called ”macro”-averaging.
- Evaluate also the overall precision of the model as the total percentage of correct predictions. This way of aggregating metrics is called ”micro”-averaging.

BONUS: you may want to use numpy manipulations to perform explicitely the task. Can you find in sklearn what are functions doing the heavy work for you? Can you check your results using sklearn function?

SOLUTION EX 2.

- Precision for a single class answer the question: ”among all predictions done of this class, how many were correct?”. In numpy terms this amounts to the calculation:

  $$\text{precision}[i] = \frac{\text{C[i,i]}}{\text{np.sum(C[:,i])}} \tag{1}$$

  Recall for a single class answer the question: ”among all samples in my dataset belonging to that, how many was I able to classify as belonging to that class?”. In numpy terms this amount to the calculation:

  $$\text{recall}[i] = \frac{\text{C[i,i]}}{\text{np.sum(C[i,:])}} \tag{2}$$

  The macro-averaged precision and recall are than np.mean[precision] and np.mean[recall]. This last operation is called *macro* averaging because we first perform a mean at each entity level and than macro-average the results. Nevertheless, the precision and recall per-entity offer an important view of the results already.

- The overall precision answer the questions: ”among all samples in my dataset, how many were classified correctly?”. In numpy terms this amount to the calculation:

  $$\text{precision} = \frac{\text{np.trace(C)}}{\text{np.sum(C)}} \tag{3}$$

  It is called also a *micro* averaged precision because it performs a mean over all samples individually but may be problematic when one class is much more represented than the others because it will dominate the micro averaging. Therefore in these cases the per-class precision and recall or the macro averaging is an alternative.

  BONUS: relevant functions are:
  sklearn.metrics.confusion_matrix,
  sklearn.metrics.recall_score,
  sklearn.metrics.precision_score
  and sklearn.metrics.precision_recall_fscore_support metrics

2 Regression and classification models

3. Linear regression.

Given a single input continuous feature x and an output y, select which models can be fit via linear regression:

1. $y = ax + b$
2. $y = ax$
3. $y = ax^2 + bx$
4. $y = ax^b + cx$
5. $y = ae^{-x} + b$
6. $y = ae^{-bx} + c$
7. $y = ae^{-bx} + ce^{-dx}$

SOLUTION EX 3.

The functions 1, 2, 3, 5 can be fitted by linear regression because they are linear in the parameters. For example in case 3 one can define two features $x_1 = x, x_2 = x^2$, and use multilinear regression to determine a and b. Cases 4, 6 and 7 are not fittable by linear regression. E.g. in case 6 one cannot use e^{-bx} as a feature because b is unknown.

4. Linear regression continued.

Suppose you still want to use a modified scheme, still based on linear regression, to fit the functions that you selected as problematic in the previous exercise. How could you do that?

SOLUTION EX 4.

Fix e.g. case 6. One could fix b and determine the loss as a function of b via linear regression, because the dependence on a and c is linear. Choosing the b that minimizes the loss would determine the best possible b. Alternatively one can define:

$$L(a, b, c) = \sum_i (ae^{-bx_i} + c - y_i)^2$$

and use non-linear optimization techniques to find a, b and c simultaneously.

5. Splits in classification trees (☕).

You are given a training data set of 4 points as in Fig. 1 where black points belong to one class and white to another. Note that you have just a single scalar feature. Between the two suggested splits can you indicate which one could lead to a positive information gain? Choose the entropy or Gini criterion and evaluate numerically (at least in formulas) the information gain of the most promising split.

SOLUTION EX 5.

We use the notation $E[p_{\text{black}}, p_{\text{white}}]$ to denote the entropy (or the gini coefficient) of a distribution. Before splitting we have $p_{\text{start,white}} = p_{\text{start,black}} = 0.5$

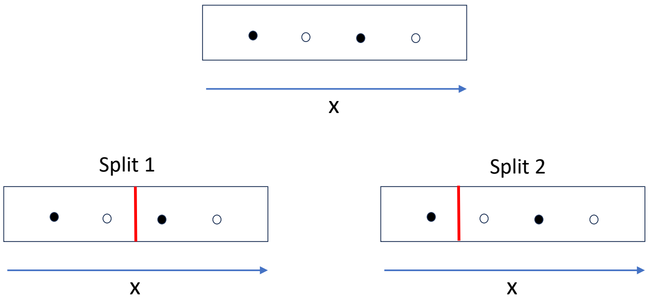

Figure 1: Figure for Ex. 5

and therefore we have an entropy $E([0,5,0.5] \sim 0.69$ (if measured in bits). After getting the information due to the split the entropy changes to:

$$\frac{N_{\text{left}}}{N} E([p_{\text{left,black}}, p_{\text{left,white}}]) + \frac{N_{\text{right}}}{N} E([p_{\text{right,black}}, p_{\text{right,white}}])$$

For split 1 since the number of points going to the left and to the right is the same, and since the popoluation probabilities do not change, the information gain is null even without making any calculation.
For split 2 instead:

$$\text{IG} = E([1/2, 1/2]) - 1/4E([1,0]) - 3/4E([1/3, 2/3])$$

, which is not equal to zero and evaluates to ~ 0.21 bits.

6. Splits in regression trees (☕).

You are given a training data set as in Fig. 2 with a single continuous input feature x.

1. Suppose you have a tree without any split. What would be the tree prediction if you want to minimize the SSR (sum of squares of the residuals)? HINT: a tree without splits is associated to a constant prediction c. Show that the optimal c is given by the mean of the y_i by analytically minimizing the SSR. Note that you have to perform some derivatives here. This justifies the choice, presented during the lesson, of the mean as best value for each leaf of a regression tree.

2. Now suppose you build a tree of depth 1 (called also a *decision stump*). Draw in Fig. 2 a generic decision boundary for the decision stump and the associated prediction values in the two splitting regions. Can you draw approximately the location of the optimal decision boundary? HINT: the splitting criterion for regression chooses the boundary such that the resulting SSR is as small as possible.

3. Now suppose to find an optimal tree of depth 2. How would the decision boundaries look like?

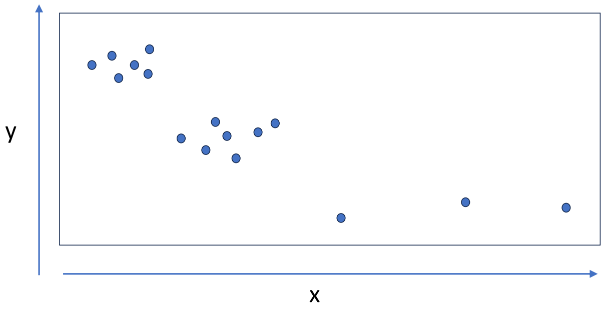

Figure 2: Figure for Ex. 6

SOLUTION EX 6.

1. If we consider just the samples in a leaf of a regression tree and a trial constant c, we can write the sum of squares of the residuals as:

$$L(c) = \sum_i (y_i - c)^2$$

   Evaluating $\frac{\partial L}{\partial c} = 0$ we have the condition:

$$\frac{\partial L}{\partial c} = 2\sum_i (y_i - c) = 0$$

   from which we get:

$$c = \frac{1}{N}\sum_i y_i$$

   Thus, the choice of the mean as a predicted value for the leaf is justified.

2. see picture 3

3. see picture 3

7. Gini criterion (☕).

Motivate the Gini criterion. In particular consider one set S of N points with associated classes $c_i, i = 1..N$. Take the sample proportions p_c for each class as defined in the lesson. Now propose the following labelling scheme that uses just the sample proportions:

1. choose a point at random from S

2. assign that point to class c with probability p_c

Show that the probability of labelling in a wrong way the the point with this procedure is exactly $G(S)$. The Gini index can be therefore interpreted as giving a measure of how well the sample proportions describe the points inside the

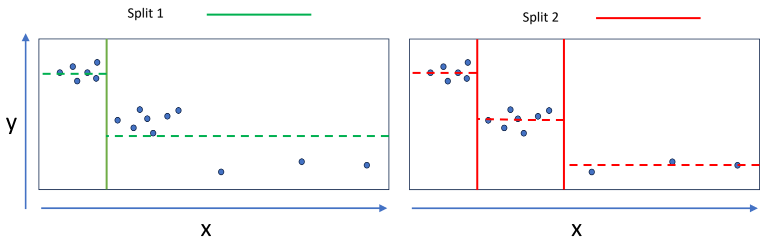

Figure 3: Figure for Ex. 6 with a solution to points 2 and 3. Dashed lines indicate the predicted value (which is the mean of the y values in the corresponding region), whereas continuous lines indicate boundaries in the one dimensional x-feature space.

corresponding leaf.

SOLUTION EX 7.

Let's fix an element from step 1, whose correct class is c with probability p_c. If we label this element according to sample proportions, the probability of making a mistake is $1 - p_c$ (since the correct class is c). Therefore the probability of making a mistake with the combinations of steps 1 and 2 is:

$$\sum_c p_c(1 - p_c) = \sum_c p_c - \sum_c p_c^2 = 1 - \sum_c p_c^2.$$

We thefore conclude that the probability of making a mistake by labelling an element according to the sample proportions is indeed equal to the Gini coefficient of the sample $G(S)$.

8. Bias-Variance decomposition (☕ ☕).
Suppose that you want to fit a deterministic mapping $f^{ex}(x), x \in X, f^{ex} : X \to \mathbf{R}$. A parametric fitted model will provide some predictions $Y^{model}(x) \in \mathbf{R}, x \in X$. These can be considered stochastic (random), because fitting with a different training set could have provided (hopefully slightly) different results. As discussed in the class, this is called the *variance* of the model.
The error at x can be defined as:

$$Err(x) \equiv E[(Y^{model}(x) - f^{ex}(x))^2] \tag{4}$$

, where the symbol $E[]$ is used to indicate the expectation value. We also define the function $h(x) \equiv E[Y^{model}(x)]$.

- Prove that:

$$Err(x) = E[(Y^{model}(x) - h(x))^2] + (h(x) - f^{ex}(x))^2 \tag{5}$$

, where the first term is the variance contribution to the error and the second is the bias.

- Interpret the archery figure presented in the lessons according to the formula above.

SOLUTION EX 8.

- Perform the add-substract trick:

$$\begin{aligned} Err(x) &= E[(Y^{model}(x) - h(x) + h(x) - f^{ex}(x))^2] \\ &= E[(Y^{model}(x) - h(x))^2] + E[(h(x) - f^{ex}(x))^2] \\ &+ 2E[(Y^{model}(x) - h(x))(h(x) - f^{ex}(x))] \end{aligned}$$

  Now the term

$$E[(Y^{model}(x) - h(x))(h(x) - f^{ex}(x))]$$

  is null as can be shown by the following chain of manipulations:

$$\begin{aligned} &E[(Y^{model}(x) - h(x))(h(x) - f^{ex}(x))] = \\ &\text{(take out of expectation non random prefactors)} \\ &= (h(x) - f^{ex}(x))E[(Y^{model}(x) - h(x))] = \\ &\text{(linearity of expectation)} \\ &= (h(x) - f^{ex}(x))(E[(Y^{model}(x)] - E[h(x)]] \\ &\text{(expectation of non random function)} \\ &= (h(x) - f^{ex}(x))(E[(Y^{model}(x)] - h(x)] \\ &\text{(definition of h(x))} \\ &= (h(x) - f^{ex}(x))(h(x) - h(x)) = 0 \end{aligned}$$

  , thus showing the decomposition.

- the first term contribution

$$E[(Y^{model}(x) - h(x))^2]$$

  describes a spread of the model predictions around its mean value $h(x)$. Therefore in our archery example shows how much the points are spread around a common mean. This is the variance contribution. The second term says

$$E[(h(x) - f^{ex}(x))^2]$$

  says how much this mean mean $h(x)$ is far from the exact value $f^{ex}(x)$. It is therefore the bias component of the model error.

9. Change of data representation (☕☕).
Point 1. Consider a fixed training set $(\boldsymbol{X}_1, y_1), .., (\boldsymbol{X}_N, y_N)$. A normalization or standardization amounts to the replacement:

$$X_{i,a} \to \tilde{X}_{i,a} = m_a X_{i,a} + n_a, \tag{6}$$

where the index a indicates the corresponding feature and m, n are constants. Prove that fitting a linear model using the standardized/normalized variables will provide the same predictions on the test set. Hint: show that you can obtain the same predictions from the two models after changing the definition of the parameters. Do you expect the same to hold with regularization?

Point 2. Consider instead now the more generic transformation:

$$X_{i,a} \to \tilde{X}_{i,a} = g_a(X_{i,a}), \tag{7}$$

where the $g_a(x)$ are monotonic real functions. Show that two decision stamps (trees with depth 1) based on the same splitting criterion will provide equivalent models after the data transformation. Argue that the same conclusion is true for any decision tree and a random forest model. Hint: the criterion for choosing a cutoff to split a node along feature a is based only on the values of y_i and the ordering of $\{X_{i,a}\}_{i=1..N}$.

SOLUTION EX 9.

Point 1. Suppose we have the model fitted with the normalized variables

$$Y = \sum_a \alpha_a \tilde{X}_a + \beta$$

Then if we replace the normalization formulas we get:

$$Y = \sum_a (m_a \alpha_a) X_a + (\sum_a \alpha_a n_a) + \beta = \sum_a \alpha'_a X_a + \beta'$$

This shows that can have exactly the same predictions in terms of the unnormalized variables with a different choice of the parameters. Therefore fitting the model with normalized or unnormalized variabels will lead to the same predictions.

This concolusion is not true if weight decay/regularization is present because variable normalization will force a different value of the optimal coefficients, that will be regularized differently.

Point 2. Suppose that we are splitting a decision tree along the a-coordinate. Suppose that with the original, non transformed variables, the optimal splitting was selected, according to the CART criterion, at $x_a = x_{opt,a}$. We claim that the optimal split in terms of the transformed variables will be at $\tilde{x}_{opt,a} = g_a(x_{opt,a})$. The monotonicity condition indeed says that if $x < x_a$, than $\tilde{x} = g_a(x) < g_a(x_a) = \tilde{x}_{opt,a}$. Therefore the left and right subsets will be equal considering transformed and non transformed variables with these two cutoffs. Further, since the information gain depends only on how the dataset is split into a left and right subsets but not on the precise value of the independent (x or $\tilde{x}$) variable, then the split is optimal in both cases.

Further, the predictions on the left and right node will be equal as well because, similar to the information gain, these depends just on which samples fall in in the right or left node and on the values of the dependent (y) variable, but not on the precise value of the independent (x) variable. This argument shows that tree models are invariant to these variable transformations and therefore require less feature engineering than the linear models. These reasonings apply to a single split, a decision tree and random forests as well.

10. Linear regression with categorical variables (☕☕☕).
When an input feature is categorical and with a natural ordering, like {bad, good, super} we have two possibilities to use the variable as input of a linear model. We want to get some intuition considering the case of a single input categorical variables x with n-levels, and a scalar output y.
Point 1. One possibility is to map the categorical input to a scalar one, e.g. $\{\text{bad} \to 0\}$, $\{\text{good} \to 1\}$ and $\{\text{super} \to 2\}$.

- Can you write down how the most generic linear model would look like in this setting? In particular how many parameters would the model have?

Point 2. Another possibility is to use a one-hot encoding. This would encode the categorical variable x into a one-hot vector $(x_1, ..., x_n)$ with length equal to the number of categories and $x_i \in \{0, 1\}$. This vector would define n variables out of the single categorical one.

- Show that $x_1, ..x_n$ are not independent. As a consequence the minimization problem solved by linear regression is not well posed. One trick to circumvent this issue is to drop the last variable from the input features.
- Can you write down how the most generic linear model would look like in this setting? In particular how many parameters would the model have?
- **HARDER**: Do you know in advance what would be the result of performing linear regression in this setting? Hint: the model should predict for each category the mean $\frac{1}{N_c}\sum_{i|x_i=c} y_i$, that is the mean over all predictions for that category. Try to write down the residual some of squares for this model and differentiate w.r.t. to the parameters you found in the previous question.

BONUS: inspect what happens when you combine a categorical feature variable with a continuous one, especially in the one-hot encoding setting? How would you interpret the coefficients?

SOLUTION EX 10.
We call $K = 3$ the number of classes in the problem. We define the function associating to a training sample i its class $C(i) \in \{1, .., K\}$.
Point 1. The most generic linear model would predict y with the model:

$$y = \alpha x + \beta$$

, where $x \in \{0, 1, 2\}$, according to the class. More explicitely:

$$y^{pred}(i) = \begin{cases} \beta & \text{, if } C(i) = \{\text{bad}\} \\ \alpha + \beta & \text{, if } C(i) = \{\text{good}\} \\ 2\alpha + \beta & \text{, if } C(i) = \{\text{super}\} \end{cases}$$

, and the number of parameters is 2 independently of the value of the number of classes K.
Point 2.

- The new one-hot encoded variables will not be independent, since $x_1 + x_2 + .. + x_K = 1$. The design matrix would be of dimensions $N_{\text{samples}} \times K$

but the columns (the encoded features of the model) would be linearly dependent. Therefore usually the last column is dropped, leading to a design matrix of dimension $N_{\text{samples}} \times (K-1)$.

- After dropping the last column the linear model is

$$y = \sum_{c=1}^{K-1} \alpha_c x_c + \beta$$

, with K parameters, where x_c are the one-hot encoded features. The model has therefore a number of parameters increasing with the number of classes.

- **HARDER:** We can find analytically how the parameters and the predictions minimizing the loss function would look like when one-hot encoding the categorical feature. If we call:

$$y^{pred}(i) \equiv \sum_{c=1}^{K-1} \alpha_c x_{i,c} + \beta$$

, the prediction for the i-sample, we note that this implies the following predictions:

$$y^{pred}(i) = \begin{cases} \alpha_c + \beta, \text{if } C(i) = c \in \{1,..,K-1\} \\ \beta, \text{if } C(i) = K \end{cases}$$

, in particular it makes sense to make this formula explicit for our example:

$$y^{pred}(i) = \begin{cases} \alpha_{\text{bad}} + \beta & , \text{if } C(i) = \{\text{bad}\} \\ \alpha_{\text{good}} + \beta & , \text{if } C(i) = \{\text{good}\} \\ \beta & , \text{if } C(i) = \{\text{super}\} \end{cases}$$

.

The loss function is:

$$L = \sum_c \sum_{i|C(i)=c} (y^{pred}(i) - y^i)^2$$

, or

$$L = \sum_{c=1}^{K-1} \sum_{i|C(i)=c} (\beta + \alpha_c - y^i)^2 + \sum_{i|C(i)=K} (\beta - y^i)^2$$

Minimizing this function w.r.t. to $\alpha_{\overline{c}} \in \{1,..,K-1\}$ leads to:

$$\frac{\partial L}{\partial \alpha_{\overline{c}}} = 2 \sum_{i|C(i)=\overline{c}} (\beta + \alpha_{\overline{c}} - y^i) = 0$$

The latter equation implies for all classes $\overline{c}$ different from the last one that:

$$\alpha_{\overline{c}} + \beta = \frac{1}{N_{\overline{c}}} \sum_{i|C(i)=\overline{c}} y^i.$$

By comparison with the previous formulas this implies that the predicted value for a sample of class $\overline{c} \neq K$ is the mean of the dependent variable over all samples of that class. The derivative with respect to β is:

$$\frac{\partial L}{\partial \beta} = 2 \sum_{c=1}^{K-1} \sum_{i|C(i)=c} (\beta + \alpha_c - y^i) + \\ + 2 \sum_{i|C(i)=K} (\beta - y^i).$$

The first term vanishes according to the previous condition. Similar to the other classes the predicted value for samples of the K class is also the mean of the dependent variable over all samples with $C(i)$ equal to K:

$$\beta = \frac{1}{N_K} \sum_{i|C(i)=K} y^i$$

.

BONUS: Combining a on-hot encoded cathegorical variable with a continuous one z would lead to the model:

$$y = \sum_{c=1}^{K-1} \alpha_c x_c + \gamma z + \beta$$

So the prediction would depend both on the class and the value of z. We can spell it out in the following way:

$$y^{pred}(i) = \begin{cases} \alpha_c + \beta + \gamma z, \text{if } C(i) = c \in \{1, .., K-1\} \\ \beta + \gamma z, \text{if } C(i) = K \end{cases}$$

, which can be interpreted as a common linear prediction γz, but with an intercept which is class dependent. Minimization of the parameters can be performed analytically as before.

3 Train-test split

11. Possible mistakes using wrong test-set distributions (☕☕☕).

Sometimes to avoid information leakage one is forced to use a test set with a different distribution than the training set. This exercise builds some intuitions on how metrics like precision and recall will be affected.

Consider a binary classification framework. As a setup we consider a starting population X, with X_p and X_n the subsets of positive and negative examples. We consider a fixed classifier C (also called *discriminator*) assigning to each eelement of the population a label and define $P(C = p|E = p)$ as the conditional probability that if an element is positive, the classifier will be also positive. Similarly one can define $P(C = n|E = p)$,$P(C = p|E = n)$ and $P(C = n|E = n)$.

Let a test set population be built with the following process:

1. Choose a random number R with a $Ber(\alpha)$ distribution, i.e. 0 with probability α and 1 otherwise.
2. If $R = 0$ choose an element from X_n (according to the corresponding conditional distribution). If $R = 1$ choose an element from X_p.

Let the resulting modified, unbalanced, population be denoted by X_α.
Answer the following questions:

- In the test set population X_α, what is the probability for an element to be negative/positive?

- Write for large N a confusion matrix for the test set X_α in terms of the four conditional probabilities.

- Use the previous results to derive precision and recall as a function of α.

SOLUTION EX 11.

- The procedure leads to a test set X_α where the probability of being a negative/positive element are $\alpha/1-\alpha$. α therefore is a parameter modelling an imbalance in the test dataset.
- The confusion matrix reads in this setting:

$$C = \begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix} \sim N \begin{pmatrix} \alpha P(p|p) & \alpha P(n|p) \\ (1-\alpha)P(p|n) & (1-\alpha)P(n|n) \end{pmatrix}$$

- From the previous matrix precision and recalled are computed as:

$$\begin{aligned} \text{Precision}(\alpha) &= \frac{\alpha P(p|n)}{(1-\alpha)P(p|n) + \alpha P(p|p)} \\ \text{Recall}(\alpha) &= P(p|p) \end{aligned} \tag{8}$$

If we fix the training set and the model, the four probabilities $P(p|n), P(n|p), P(n|n)$ and $P(p|n)$ are fixed. Precision measures therefore a quantity that depends on the model itself, but also on the imbalance of the data over which we test the model. Recall instead is not dependent of the class imbalance parameter α.