Practical 2 - *Know your Performance*

Chris Emmery

19 September 2017

In this practical[1] we will *again* focus (i) on exploratory dataset analysis — an important first step before engaging in any Data Mining activities. We want get to know our data: find interesting observations, anomalies in data, and think about how to best go about solving the task. After, (ii) on making predictions, ascertaining the correctness of these predictions, and trying to party improve them. However, this time you are going to focus on **classification** rather than regression, and work mostly on your own.

[1] **Important Practical Note**: If you cannot answer a task during the practicals fully, or feel unsure about your answer (even after the explanation), please ask! It is very important that you develop the correct intuitions for each of the points we discuss here. Sometimes they just don't 'click' by themselves; they require a lot of repeated practice and interpretation, and not every explanation works for everyone. We will be very happy to answer all your questions on the Forum or during class!

Refresher

Any machine learning task can be formally evaluated by using the true values for a target[2] y, and comparing them to the predicted values $\hat{y}$ by an algorithm. Previously we did this by simply looking at the error (deviation from the true values), or accuracy scores (percentage of correctly predicted instances).

[2] Which generally can be any feature of choice, but mostly there's a clear prediction task associated with a dataset / job / or research question.

As our point is generalization, training and evaluating (or testing) on all currently available data will not give us a good sense about how well the model will perform on totally new data (which is what we are interested in)[3]. Rather, the results would merely hold for reproduction; showing how well a model with perfect information would be able to reproduce exactly the same labels. The odd result of evaluating on your training data, is that intuitively uninformative features (unique for each instance, does not contain any information about the target we are trying to predict) will likely yield incredibly high results, while intuitively informative features might not[4]. As this is counter-intuitive, it is correct to deem this behaviour as wrong. Uninformative features should NOT yield high scores (unless they have some surprising characteristic, which is unlikely), and high evaluation scores, or very low errors, in general should not be trusted at face-value.

[3] Imagine for example trying to predict the weather. There's a lot of weather data available now, which you can probably predict with very little error. We don't have future forecasts available, because that's what we're actually trying to do. So, we create an 'artificial future' by splitting off some part of our dataset, and seeing how well our predictions from say 2005-December last year (our training set), work for the past months in the current year (our test set).

[4] Taking the Titanic dataset as an example, if we include `PassengerId` as a feature (and the model memorizes this for every person), when offering the training data to evaluate, the model will make new predictions based on the same data. As such, it would only have to look at the passenger ID, and it would know whether this person died or not. However, if we would have a test set, the passenger IDs for that test set would not have been seen by our model, and would therefore be totally useless to predict survival for other passengers.

Testing Models

In a realistic setting, we would want to leave some of our data *unseen* by the algorithm. For this unseen data, we provide the model with the feature vectors only (no target), and hope that it will guess the label correctly. Now, in that setting, if we obtain a high score —

then we might conclude that our algorithm is performing well, and we can start to interpret the actual contribution of features (for some models) to this score. Still, even the interpretation of the performance on the test set should be considered in the light of (i) the size and quality of your data[5], and (ii) the hyper-parameter settings of your model[6].

One of the naive ways of making sure that the model is provided with some unseen data is by simply chopping off a part of your data, and leaving it aside for **final**[7] testing. As such, you have A) a big training set, B) a small test test. The actual proportions of these sets depend on the amount of data that you have, but in general it's good to leave at least 20% for testing. These proportions are *not set in stone*, and therefore you cannot just say "always leave 20% for testing". Treat it as a starting point.

[5] e.g. high performance on little data gives no guarantees if your goal is to make predictions for much larger volumes of data.

[6] Remember these are the knobs that we can tune to potentially improve the performance of certain algorithms, like the amount of neighbours k for k-NN.

[7] That means, after we have preprocessed, selected features, selected hyperparameters for our models, and selected an algorithm of preference. Thus: after all decisions in the Data Mining workflow have been made final.

Baselines

IN THIS SETTING we have some guarantee that the model is tested fairly. However, we are still unsure if it has actually learned something. The notion of learning is a bit simplified for our purposes: in general, we want to either i) outperform state-of-the-art models doing the same task (otherwise no need for us to train our own), or, if there is not such a model, ii) beat some simple model, or a very stupid approach to prediction. The latter is known as a **baseline**. We have seen this before when using the mean of the target value in our training set when doing regression, and predicting this mean for all instances in our test set. For classification, we use a similar variant: the majority baseline. At training time, it records the distribution over labels (so the label and their associated frequencies), and at test time, it always predicts the most frequent label from the training set.

Both of these baselines perform very well under certain circumstances: if the target value is normally distributed in case of regression, and if there's one very common (dominant) class in the case of classification.

Different Sets for Different Purposes

AS WE'VE SEEN up until now, any Data Mining routine starts with data; given that in the Practicals we're using pre-made datasets, the first step is determining the goal of the dataset, informing yourself about the features, some preprocessing, inspecting the data carefully for any anomalies, and setting up a prediction task for yourself.

To know how well you are doing while running your models we can discern up to four types of splits in a given dataset:

- Training set.
- Validation (or development) set.
- Testing set.

The size of these with respect to the whole dataset usually differs, and if you are using data for an existing prediction task, usually you are already provided with a test set. In general, popular splits are (train / validation / test) `80/10/10` or `50/25/25` (given enough data).

Validation

THE PROCEDURE OF TESTING performance would then be as follows: we train our model on the training set, and evaluate using the validation set[8]. We tune the hyperparameters of our classifiers to maximize the score on our validation set (and generally between our training and validation set).

[8] Note, again, that we're leaving the test set aside for the entire process.

The intuition behind this is as follows: since we are trying to predict new data (our test set), we want to simulate this in some way with our completely unseen test set. If we would tune the hyperparameters based on the result on the test set, the optimal hyperparameter set for this piece of data is now know to us. However, if we then get actual new data, it might turn out that this wasn't the best hyperparameter setting, and we now perform very poorly. So, effectively, we would not know how the model performs on new data. If we use a validation set to determine the effect of tuning the hyper-parameters, we can determine an optimal setting, and then after have a fair, unseen evaluation round on the test data, so that we have a correct measure of how the model would perform on new data. In conclusion, important to remember: you are also part of a fair evaluation!

Part A - Evaluation Metrics

HOW DO WE MEANINGFULLY interpret the performance of our model? So far, we have looked at R^2 and RMSE for regression. For classification, we have briefly mentioned accuracy; the percentage of correctly classified instances. Let's go into a bit more detail on it here.

Classification Metrics For this example, let's assume that we have a dataset with a lot of dogs (5000), and some cats (200). We want to see if we can distinguish a cat from a dog. As such, we have two classes: `cat`, and `dog`. Cat is the minority class here, so we are actually interested to see how well we will do on classifying cats amongst a lot of dogs (because a majority baseline would get a pretty high score for dogs). As such, `cat` will be our **Positive** instance (that which we are interested in doing well at), and dog **Negative**. Consider the example in Table 1.

true label (y)	predicted label ($\hat{y}$)
dog	cat
cat	dog
dog	dog
cat	cat
dog	dog
dog	cat
dog	cat
cat	cat
cat	cat
cat	cat

Table 1: True labels and predictions.

Confusion Matrix This is generally the first thing we look at. A matrix representation of the positive instances we predicted correctly as positive (TP: true positives), the positive instances we predict incorrectly as negative (FN: false negatives), the negative instances we predict incorrectly as positives (FP: false positives), and the negative instances we predicted correctly as negatives (TN: true negatives). The notion here is that you treat it from the prediction point of view: if we predict some instance to be a negative class incorrectly (false), it's a false negative (our negative prediction was false). So false negatives should actually have been positive. False positives should have been negative. We represent it like this as in Table 2.[9]

[9] We currently only explain this in a binary prediction task (with only 2 labels, so either yes or no, dog or cat). In reality you probably have more labels, just make sure that you understand it in the binary scenario.

		predicted as	
		cat	dog
true label	cat	TP	FN
	dog	FP	TN

Table 2: Confusion matrix.

Accuracy The amount of correctly predicted examples:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \tag{1}$$

Recall Will tell you something about how many of the instances that you wanted to classify you actually classified correctly. So in our example, if we managed to classify all cats as actual cats (even if we classified some dogs as a cat) our recall is 100%. This distinguishes itself from accuracy, because accuracy says something about how well we classify ALL our classes. Say that we classified 100 cats as cat, but 100 cats as dog, our recall is 50%.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

Precision Will tell you something about how many instances you have classified as being Positive (i.e. cat here), were actually a cat. In our example, if we managed to not classify any dogs as cats (even if we only classified two instances as being a cat), our precision is 100%. Say that we classified 100 cats as cat, but also 100 dogs as cat, our precision is 50%. So:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3}$$

F_1 Measure These two metrics therefore trade off two properties of our prediction process: how many instances that we want to detect as Positive do we actually detect, and how many instances that we detect are actually Positive. In some scenarios we want to optimize precision, in some we want to optimize recall, but this is only when we can't do both (because of lack of data, unbalanced instances, poor choice of classifiers, etcetera). In general, we actually want both high recall and high precision. To sort of squeeze this into one metric, we use the 'harmonic mean' (approximately average) between precision and recall, called the F score or F_1 score:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

Task 1

- Make a confusion matrix using the cats / dogs table.
- Calculate Accuracy, Precision, Recall and F_1 score.
- How well does this classifier do on your prediction task (compared to a baseline)?
- Think of a task where optimizing precision is important, and one where recall is important.

Part B - Metrics In Practice

Up until now we've touched upon many factors within a data mining pipeline that need to be taken in consideration to accurately interpret the predictions made by our models. We discussed metrics, baselines, different data splits (train, test, val), parameter tuning, and overfitting and underfitting. These are all important variables in your entire set-up, and therefore require a thorough understanding of best practices with regards to different scenarios.

The Data

For this part, we'll work with some imaginary data. We assume that we have already thought through setting up the predictions, so the characteristics of the data matter less for now.

What we do know is: the data has 500.000 instances, with a mix of continuous and discrete features. There are two potential targets for prediction; one is continuous, and one is discrete.

Task 2

Try to solve the **mistakes** in the methods for the cases below given what you know of earlier mentioned concepts, and the description of the case.

- **Case 1** — We sampled about 20% of the total data. We apply standard Linear Regression and get an error of 0.05. We're happy with the result and report it as is.
- **Case 2** — We use all of the data, but split off 10% for testing our model. We start applying k-NN and tuning k manually until we get the highest accuracy score on the train set (k=50, accuracy=0.80). We then apply it to the test set and get a score of 0.85.
- **Case 3** — We add a baseline to Case 2 to compare. With the exact same procedure (as Case 2) we get a baseline accuracy score of 0.50 on the test set. We're happy and report it as is.
- **Case 4** — We set up the necessary splits, tune k fairly and see that both our test and train scores are now lower (0.80 and 0.70). However, we are still above baseline performance (0.50). We see that the final k value settled on was $k = 3$. However, when we manually try $k = 5$, we get 0.79 for train and 0.78 for test. We settle on $k = 5$.
- **Case 5** — We decide that some of the labels we use for classification actually have a low occurrence, and when we inspect the confusion matrix we see that they are almost always guessed incorrectly. We remove them and run the experiment again.
- **Case 6** — Suppose the data we have been using up until now are behavioural attributes of persons recorded by a security camera on an airport. They have binary labels with potential threat yes / no. We have 98% harmless travellers, and 2% potential threats. We use accuracy to evaluate our classifier against a majority baseline.

Part C - DIY Pandas & Sci-kit Learn

The dataset we'll be working with for this practical is the following:

Information	not available	← Text in the table is clickable.
Data	raw github	

Data description of an almost identical dataset can be found here. Please make sure you understand this dataset and the task **before** beginning!

IT has moved our course's repository to a central location (`/srv/data-mining`), so no need to pull the repository any more. You can access the IMDB dataset like so:

```
import pandas as pd

df = pd.DataFrame.from_csv('/srv/data-mining/data/IMDB/imdb.csv')
df
```

Task 3

Explore the dataset like we did before, and hand in your results in the same format.

Tips[10]

[10] These will probably keep updating, always check the latest version of the document!

1. The feature `quality` is not in the original dataset. Try to compare it with imdb_score.
2. Check the `gross` feature for outliers, and find them in the DataFrame. See something interesting with these instances?
3. Make sure to `pop` unusable features!
4. Find classifiers and evaluation metrics in scikit-learn's manual.
5. The dataset has NaNs encoded as `?`, which are not interpreted by Pandas. You'll have to manually replace these with:

   ```
   df['some_column_name'] = df['some_column_name'].replace('?', 0)
   ```

6. If you want to convert feature values from categorical to numeric, you can apply either of two methods. If order matters you'll have to make the mapping by hand, e.g.:

   ```
   df['quality'] = df['quality'].replace(
     {"quality" : { "very-bad" : 1, "bad" : 2, "okay" : 3,
                    "good" : 4, "very-good" : 5 }}, inplace=True)
   ```

   If it doesn't, you can use scikit-learn like so:

   ```
   from sklearn.preprocessing import LabelEncoder
   le = LabelEncoder()

   df['some_feature'] = le.fit_transform(df['some_feature'])
   ```

7. If you get any error that says 'cannot convert ... to float', and you did remove your NaNs, you might want to convert your features to numbers. You can try either of, or both:

   ```
   df['some_feature'] = df['some_feature'].astype('float')

   df['some_feature'] = df['some_feature'].convert_objects(convert_numeric=True)
   ```

8. Before making plots, carefully check your features on potential required fixes from the points above.

9. If you get an error related to `$DISPLAY`, don't forget `%matplotlib inline`.

Solutions

Task 1:

- TP = 4, FN = 1, FP = 3, TN = 2.
- A = 0.60, R = 0.80, P = 0.57, F_1 = 0.67 (rounded).
- Majority baseline will have 50% (given balanced classes), so we're doing a bit better.
- Recall is important when we don't want to miss out on any potential targets, regardless of how many errors we will make in the process, and misclassification comes at a low cost. Precision on the other hand is where misclassification does come at a high cost. Say that we're *identifying* potential plane hijackers; we do **not** want to miss any of those, so we require high recall for inspection; however, once we are determining if we should *detain* them, we don't want to falsely imprison innocent civilians, so we require high precision.

Task 1:

- **Case 1** — There are many issues with this particular example, most of which will become more apparent once we progress through all of these cases. The most important one, however, is that of the prediction set-up.

  In this case we train a regression model without any note of a validation or test set. So, this very small error that we got is a direct result of evaluating predictions conducted on previously seen data. As such, we reproduce the target values rather than predicting them; we don't learn a model that generalizes, but is specifically tailored to perfectly reproduce the values associated with the input.

  Sampling can't be regarded as a mistake per se; despite the fact that our model covers less data, there might be some restrictions that limit us to only using this much (e.g. computation time, size of the data). We do however need to realize that data reduction through sampling makes our model less robust (i.e. have the ability to generalize well), and will not give us a lot of confidence in how well it might perform on a bigger variety data. If you can, use all the data you can process.

- **Case 2** — In this case, parameter tuning of k is done by evaluating its performance on the training set. This makes as little sense here

as it did in the previous case. What we can observe as well, is that tuning resulted in a very high k value of 50, which means the model could be underfitted (which we can confirm when we see the score on the test set is higher). Ideally, we would want to validate our tuned performance on a separate set.

Moreover, and this also holds for the previous case, there is no baseline! Even if we did everything correctly, we don't know if a baseline score might have resulted in an accuracy score of .90 or something - which would've left us with the conclusion that the model learned nothing.

- **Case 3** — We score much higher than the baseline, so our model is definitely doing well. However, we still need to create that development/validation set to make sure that our tuning is correct.
- **Case 4** — For k to be tuned fairly, tuning was conducted on a validation split. The scores are lower, but that's to be expected, as we're now actually doing predictions on unseen data (both validation and test). The fact that we're scroing better than a baseline classifier (such as majority) indicates that the model is actually learning something from the data.

  The mistake here is that we go back again after having looked at the performance of the test set. We know use our newly acquired knowledge of this 'new data' to go back and tune. Through this, we now don't know how it would perform on new data.
- **Case 5** — Again, we're using information that we did not have any access to prior to conducting these experiments - just to increase our score. Just because these labels are hard to predict, doesn't mean that they are to be excluded!
- **Case 6** — Majority baseline will be a very hard baseline to beat (given that 98% is negative), which is good! However, accuracy will not reflect our score very well; it will be anything up from 98% and therefore might give us (or someone else) the illusion that we're doing way better than we actually are. Generally, we would want to focus on the correctly classified **positive** instances in this regard.

  Additionally, this is a typical case where precision and recall become important metrics. Are we going for as few mistakes as possible (high precision) or are we interested in capturing as many threats as possible even though we might have to harass innocent civilians for it (high recall)? Note that the threat here is not specified. Drug traffickers might be less important to capture than plane hijackers.