**Project description and instruction BMLB2024**

**Introduction**

After all you’ve learned, the proof is in the pudding. Which is to say: no better way to test what you’ve learned than by applying it. So, time to get open that tub of elbow grease and get going applying what you’ve learnt. The brunt of the work will be done in [scikit-learn](https://scikit-learn.org/stable/), where it’s especially important to use [Pipelines](https://scikit-learn.org/stable/modules/compose.html#pipeline) to easily ensure that (nested) cross-validation is performed correctly. You submit your final **annotated** (i.e. understandable for the teacher, which is usually Dieter, aka. me) Jupyter notebook **to** [**d.g.g.stoker-6@umcutrecht.nl**](mailto:d.g.g.stoker-6@umcutrecht.nl) **by Wednesday the 13th of March 2024 15:00 at the latest**. **Make sure to include your names and student numbers in the e-mail, and have it CC’ed to all group members. This project counts for 25% of the grade** (see Grading below).

**Instructions**

1. Form a team of two people. If there is one person left a team of three is allowed. If you prefer to do it alone: fine by me as well, and power to you.
2. **Optional** if you want to use GitHub to work together on the Jupyter notebooks: make sure you have nbdev installed so you don’t get inconsistencies due to saved cell state. Run conda install -c fastai nbdev. You hand in one notebook per team so you can also work together on one laptop, whatever suits you.
3. Read the dataset description below.
4. Do the guided exercises provided below that.  
   **Do your work in a Jupyter notebook file. This file needs to be handed in to me on Wednesday 13-03-2024 by 15:00 at the latest.** Things I want to see in your Jupyter notebook file are underlined in the exercises below.
5. After the guided exercises, the assignment is simple: get the best performance on the test dataset. In practice, I expect that you do one thing that is not in the guided exercises: optimise hyperparameters more, use a different classifier, make different classifiers vote, etc. Here too, describe what you did and why in the Jupyter notebook you hand in to me. **As long as you attempt some extra step beyond what is asked in the guided exercises you have fulfilled this requirement**. **At the end, you test performance of a few of the classifiers trained on the held-out test data that you should leave untouched all throughout *until the very end*.**
6. ***I do not accept materials handed in after 15:05 on the 13th!***

**Grading**  
  
This project is meant for you to put into practice what you’ve learned on correct ML training procedures, now using modern ML libraries. You will only be graded on whether you correctly perform the guided exercises (correctly means: you do what is asked, you supply answers to the questions in the Jupyter notebook you use, you clearly supply performances, you follow correct cross-validation procedures, you label your plots) and perform at least one additional tweak to your classification pipeline and clearly report on what you did, why, and what the result was in the Jupyter notebook you hand in. If you do these things, you are golden. I don’t care about coding style, naming conventions, or who gets the best performance in the end. The project counts for 25% of the course grade (given that you get at least a 5 on the exam).   
  
As an indication: last time marks ranged between 7 and 10, and if you do everything in the guided exercises correctly you are already high in that range**. I will grade your notebooks.**

**Dataset description**

**TL;DR: 3520 training samples, 480 test samples. 2000 features. 4 classes. Imbalanced data.**

Small cell lung cancer is a highly malignant and deadly type of lung cancer which spells death for most of its sufferers (see [here](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6538259/)). Successive gene expression studies have identified different amounts of subtypes (where, as often, histological subtyping (looking at tissues) and genetic subtyping (clustering based on gene expression data) don’t always align). Of course, the genetic expression subtypes also don’t completely match each other. A recent synthesis converged on 4 main subtypes of this cancer, characterised by mutations in some important factors. Let us assume that each of these subtypes needs very different treatment regimes, and that current clinical tests don’t allow adequate separation between the cancer subtypes.

Our dataset is a simulated one, and our hypothetical problem description is as follows: given patient expression data, we want to make a diagnostic classifier that will tell us which cancer subtype the patient has. We can imagine that the expression data comes from a biopsy of a SCLC tumour. For our case, we have assayed all human protein-coding genes, and then pre-selected 2000 (so **2000 features**) of them to keep the problem manageable (you can imagine that many genes don’t vary at all between the subtypes and hence would have no value whatsoever).

In our dataset we have samples from each of the **4 classes**. Unfortunately, the 4 subtypes have not been sampled equally, but we don’t know whether that’s because of actual population incidence (i.e. that one subtype occurs less often than another because it requires more specific mutations, say) or because of sampling (perhaps patients with one subtype experience problems later than others, but then deteriorate so rapidly that we haven’t been able to obtain as many biopsies of those tumour subtypes). **We assume that we want to be equally effective at detecting all subtypes, so for those purposes take note that the dataset is *imbalanced*.** The simplest fix for this is that you downsample the data, removing samples randomly from the training set for certain classes so that you have equal size data per class again. Of course, this leaves information on the table, and you can do smarter things down the line, if you want.

We can imagine that the data is in the format of Reads Per Kilobase Mapped reads (RPKM) (see [here](https://www.biostars.org/p/273537/) for a short discussion on some metrics for RNASeq). It’s a metric of gene expression that has been normalised for the total number of reads and the gene length. This has no bearing on your workflow, but it is nice to know what you could *imagine* the numbers to mean. Note that there are no fractional reads and no negative reads, so range should be from 0 to infinite.

I would like to reiterate that I just generated a dataset and mapped it to an actual scenario. Don’t go looking for actual determinants of SCLC subtype and think you can get ahead: it won’t work!

**Guided exercises**Do not use the test data at all until the very end. All your work should be done with the train\_2024.csv file. Note that you should probably split off the Outcome column (i.e. don’t train on a feature matrix with the class as one of the features!). Only at the end do I ask you for performance on the test set. You can disregard the Id variable, except at the end where you will need it to calculate performance on test\_2024.csv together with answer\_key\_2024.csv

1. Do some exploratory data analysis. Don’t just look globally, also look at these metrics by class (if applicable). There might be differences, after all:   
     
   How many missing data are there?   
   Are there other strange values in the data?   
   What is the proportion of classes in your data?   
   Are there genes (features) that are extremely highly correlated with each other? If so, how many?  
   What is the maximum value in the data, and the minimum?   
   Which feature has the highest variance?   
   Finally, make a boxplot of the expression values of the 30 features with the highest variance in the dataset, ordered by this variance.   
     
   I expect a short comment on/answer to all these questions and what you found in your notebook, as well as the requested boxplot.
2. Big surprise: there’s missing data. Before you deal with that, take care of the class imbalance problem in the brute-force way described above: downsample the data so that you have equal numbers of data for each class. Continue with that training data to the following steps.  
     
   Clearly print the head and tail of the new training data (and labels) and show that it has equal amounts of each class.
3. Make a Pipeline that combines three steps:   
   -imputing missing data (removing np.nan) by replacing them with the mean value (SimpleImputer)  
   -scaling the data to have 0 mean and unit variance (StandardScaler)  
   -predicting the class using *unregularized* logistic regression (sklearn.linear\_model.LogisticRegression)  
   Now, split the training data into 20% validation and 80% train data.  
   Fit the Pipeline on the train data and test on the validation data.   
   In your notebook file:  
   -report the ROC AUC one-versus-rest and macro F1 score on the training and validation data. Read up on these here ([1](https://towardsdatascience.com/multiclass-classification-evaluation-with-roc-curves-and-roc-auc-294fd4617e3a), [2](https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f)) so you know what they are, and write what they do in your own words in the notebook.  
     
   -describe why you can’t directly impute and/or scale on all the training data and only then split into train and validation sets if you want an accurate estimate of your generalisation performance
4. Again, make a Pipeline that combines three steps:  
   -scaling each value to 0 mean and unit variance  
   -using a KNN imputer to impute missing data (removing np.nan). Set n\_neighbors to 3 and weights to ‘distance’. Make sure you understand what this does.  
   -using a logistic regression without regularisation to predict the class.  
   Train on the same 80% split, and report the performance on the validation data (F1 macro; ROC AUC OvR). Then finally train on all your training data, and predict on the test set.   
   In your notebook, write down why the order of imputation and scaling is now different compared to step 3, and what would happen if you do it the other way around.
5. This data has many dimensions. You are hence probably overfitting *like it’s 1999* (or [1699 for the Weird Al fans out there](https://www.youtube.com/watch?v=lOfZLb33uCg)). Let’s not do that. One simple way to combat overfitting a bit is by regularisation. Regularise using an L2 penalty with a C of one. Note that C is . We will also use cross-validation now, as we know that this gives a better estimate of generalisation performance and uses training data more efficiently.  
   Make a new Pipeline that:  
   - uses the SimpleImputer again (KNN is slow)  
   - uses L2-regularised logistic regression for classification  
   For training the model:  
   - use 10-fold cross-validation rather than a single split. Use sklearn.model\_selection.cross\_validate, and look at F1 macro and ROC AUC OvR averaged over the folds.   
     
   Report how much average cross-validation performance with regularisation included improves over unregularized logistic regression.
6. Do a PCA on your (normalised, imputed) training data (all of it, not within a Pipeline) and make a plot of the first 2 PCs. Colour the points by the class labels. How much variance is on the first and second component? Which single feature contributes the most to each PC?   
   I want to see this plot and the answer to these two questions in your notebook
7. For the next part, let’s use PCA in your prediction Pipeline. Insert a PCA step where you think it fits and use 100 principal components. Keep the regularisation in your logistic regression.   
   Report the average macro F1 score and ROC AUC OvR over folds now that you’ve included linear dimensionality reduction in your notebook. How many percentage points do you improve? Train a final model on *all* training data. Use this final model to predict on the test set.
8. There’s two things left to do: using nested cross-validation for hyperparameter optimalisation, and training some different classifiers. Let’s focus on nested cross-validation first. For now, do a RandomizedSearchCV over n\_components for the PCA [50, 100, 200, 500] and C for the logistic regression [0.01, 0.1, 1, 10, 100]. **Use an outer\_cv of 5, and an inner\_cv of 3.**  
   In your notebook, again report the average F1 macro and ROC AUC OvR over outer folds.  
     
   Finally, train a classifier on all the train data using the best hyperparameters (take the average of the ones used in the outer folds), and make your predictions on the test set.
9. Now, you can switch out the classifier you use. I want to introduce you to a classifier we haven’t covered in detail: a Random Forest (RF). Chances are you’ve heard of it.  
     
   **Background:**A RF is useful because (in theory, *not per se* in practice) it won’t overfit. It guards against that all by itself. Also, it doesn’t need features to be scaled ([it can even hinder estimates](https://stackoverflow.com/a/8962851), though for now we disregard that detail), and calculates its own *feature importances*, which tell you how important certain features are for its correct classifications, although the standard feature importances calculated in sklearn are biased in some ways ([1](https://datascience.stackexchange.com/questions/51976/why-is-random-forest-feature-importance-biased-towards-high-cadinality-features), [2](https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-25#Sec12), [3](https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html)). Where logistic regression still assumes an underlying additive linear model where features have independent effects, a RF can combine features in a nonlinear way (class is 1 only if x1> 20 AND x2 > 10 but < 20 AND x3>0.5).  
     
   I want you to watch (the material that you don’t know from) these 3 videos:  
   [one](https://www.youtube.com/watch?v=_L39rN6gz7Y&t=0s) ; [two](https://www.youtube.com/watch?v=J4Wdy0Wc_xQ); [three](https://www.youtube.com/watch?v=sQ870aTKqiM). Then, I want you to skim [this](https://www.quora.com/Why-is-random-forest-better-than-logistic-regression?share=1), to get an idea of the merits of RFs versus logistic regression. As optional extra information: a [benchmarking paper](https://epub.ub.uni-muenchen.de/39955/1/TR.pdf) found that RFs using default parameters are better in ~70% of datasets compared to logistic regression using default parameters, so they are good but not always king.   
     
   Write down in one or two paragraphs how a RF works, mentioning what the Gini impurity is and how it is used, and how bootstrapping per tree and random feature subsampling per split are used to automatically combat overfitting.  
     
   **Practical implementation RF:**  
   Now that you know what’s happening, use a Random Forest to make your classifications in the Pipeline. It is up to you whether you want to leave PCA in or not, both are fine. Use n\_estimators=250 and default parameters otherwise. You might want to set n\_jobs=-2 to use all but one CPU to speed up the fitting. Note that, as the video says, RFs can use their own weighted KNN Imputation internally to deal with missing data. **However, sklearn’s implementation does not do this, so you still need to SimpleImpute!** As before, when training you use cross-validation on all the training data you get, for your final model you train on all the data and *then* predict on the test set and upload your predictions. Training these RFs takes some time, you can expect at least 10-15 minutes total. If it takes too long, feel free to reduce the number of estimators of the RF.  
   Report the following:  
   The cross validation performances (AUC ROC OVR, F1 macro) of the Random Forest classifier.  
     
   How much better or worse then regularised logistic regression the RF classifier performs  
     
   The 5 features that had the highest feature importance (and the associated feature importances) in **the final RF trained on all training data**.   
     
   A plot for each of these 5 features that shows the distribution of the feature split by class in the training data (see [this](https://seaborn.pydata.org/generated/seaborn.displot.html#seaborn.displot)).
10. Finally, I want you to train a simple feedforward dense/fully-connected neural network on this data. Use ReLU activation functions, and 3 hidden layers with 30, 20, and 10 neurons, and 4 output neurons for the classification (with a softmax activation). Do like we did before: define a function to make your neural net in Keras, make a scikit-learn object out of it, and train it with cross-validation. **Don’t perform hyperparameter optimalisation for the neural network here**. That could become very time-consuming. Scale the data, but **rather than imputing nans just set them all to -1.**

**Further instructions + testing on the test set.**

From this point on, you are on your own. The goal is to get the best classification F1 macro score on the test set. What are some obvious avenues to explore? Well, we used downsampling, which leaves information on the table. You could try some functions from [imbalanced-learn](https://imbalanced-learn.org/stable/) to see whether you can do something smarter. I let you use a few classifiers, but sklearn has a lot more, and there are more out there. An oft-used one is [XGBoost](https://xgboost.readthedocs.io/en/stable/). [StatQuest has a video series on it](https://www.youtube.com/watch?v=OtD8wVaFm6E) (but feel free to use it as well if you don’t know what it’s doing). It can be used within sklearn, see [this tutorial](https://www.datacamp.com/community/tutorials/xgboost-in-python). You used PCA for dimensionality reduction, but you [have options there as well](https://scikit-learn.org/stable/unsupervised_learning.html#unsupervised-learning). You could also combine multiple individual classifiers into a [voting classifier](https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier). Finally, you can of course do better by just putting more computing power behind finding optimal hyperparameters for a given pipeline.  
  
Clearly write down (at the end of the Jupyter notebook) what step(s) you took and why to try to increase your final classifier performance. Report the average ROC AUC OvR and F1 macro score over outer cross-validation folds of your classifier.

**As as final step, run your full trained classifiers from step 3, 5, 7, 9 and your final tweaked classifier on the test data and report the ROC AUC OvR and F1 performance. NOTE: for your final classifier you always train on *all the training data you have*. Which classifier does best on the held-out test data?**

**What I expect**

I expect that the first 10 steps take you at least 1.5 days. Hence, I assume you have something like 4 hours to try something extra. **I want *at least* one extra step from you that is different from the guided exercises. If you further optimise a hyperparameter or two using nested cross-validation that is fine!** Extra time is probably better spent preparing for the exam, so do not go overboard.

**Jupyter notebooks**

You should hand in a .ipynb file with the code you used to do these exercises, and with the underlined information you are asked for within it. **Send it to** [**d.g.g.stoker-6@umcutrecht.nl**](mailto:d.g.g.stoker-6@umcutrecht.nl) **by 15:00 at the latest on the 13th of March 2024.** **Make sure to include your names and student numbers in the e-mail, and have it CC’ed to all group members.**