Project description and instruction BMLB2023

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. For that, we host an InClass competition on Kaggle. Kaggle is a huge online machine learning platform that has thousands of (cleaned) datasets from many domains and regularly hosts machine learning competitions. Competitors share their code and approach, can discuss why they did what they did, and learn from each other. There are also tutorials and many plug-and-play examples of ML pipelines. A good place to know for the aspiring ML practitioner!

You can sign up to the competition here. The brunt of the work will be done in scikit-learn, where it's especially important to use Pipelines to easily ensure that (nested) cross-validation is performed correctly. **Note that the presentation has been scrapped to save you time: you just hand in a Jupyter notebook with your work and submit results on Kaggle. I will shortly discuss the assignment at the start of the question hour on Thursday 16th.**

Instructions

1. Make a Kaggle account.
2. Sign up to the competition using this link.
3. Form a Team of two or three people on the competition page. You should probably navigate to the Teams tab of the competition for this.
4. Read the dataset description below.
5. 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 15-03-2023 between 15:05 and 17:00.** Things I want to see in your Jupyter notebook file are underlined in the exercises below.
6. For some steps in the guided exercises, you need to submit your predictions for the test dataset. You will be tested on the macro F1 of your classifier on Kaggle. You can use this score function from sklearn to get macro F1 values in your pipelines. Read up on this metric here. To submit on Kaggle, you simply upload a .csv with 2 columns: one called "Id" with the IDs of the samples, and one called "Outcome" with your predictions (0,1,2, or 3). Note: to save a DataFrame as csv without an index column, use .to_csv("yourNameHere.csv", index=False)
7. After the guided exercises, the assignment is simple: get the best performance on the test dataset. The minimum I expect is 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 in the Jupyter notebook you hand in to me.
8. **The Kaggle competition closes at 15:05 on 15-03-2023.** So submit your final result before that.

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 30% 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. **You do not need to give a presentation. Instead, I will shortly discuss the assignment at the start of the question hour on Thursday 16th, and grade your notebooks.**

If for some reason the Kaggle competition does not work, submission to Kaggle is dropped as a requirement, but I still want the Jupyter notebook with your work.

If the guided exercises turn out to be too much, I will update requirements. This will be communicated if and when it turns out to be necessary.

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). 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 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

1. Do some exploratory data analysis. Don't just look globally, also look at (some of) these metrics by class. 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?
   How many genes (features) have entirely equal measured expression values as other genes (i.e. are there duplicate columns in the data)?
   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, 2) 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.

-it should again use 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.
Submit a .csv with your predictions on the test set to Kaggle.

5. This data has many dimensions. You are hence probably overfitting *like it's 1999* (or 1699 for the Weird Al fans out there). 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 $\frac{1}{\lambda}$.
   Change the imputation step back to a SimpleImputer (KNN takes a long time, as you may have noticed). Switch from using one split into 80% train and 20% validation to using 10-fold cross-validation. 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.
   Submit a .csv with your predictions on the test set to Kaggle.

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 10, and an inner_cv of 5.**
   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.
   Submit a .csv with your predictions on the test set to Kaggle

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. 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, 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, 2, 3). 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 ; two; three. Then, I want you to skim this, to get an idea of the merits of RFs versus logistic regression. As optional extra information: a benchmarking paper 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.

   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. 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.
   I want you to report the 5 features that had the highest feature importance (and the associated feature importances) in your notebook, along with a plot for each that shows the distribution of the feature split by class in the training data (see this). I also want you to report the same scoring metrics as before. Does the RF improve over logistic regression or not?
   Submit a .csv with your predictions made with a Random Forest

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. A neural network doesn't like nan values, and is also prone to overfitting, so keep the normalisation, SimpleImputer, and PCA step in your pipeline for now.
    Submit a .csv with your predictions made with a Dense neural network

Further instructions

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 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. StatQuest has a video series on it (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. You used PCA for dimensionality reduction, but you have options there as well. You could also combine multiple individual classifiers into a 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.

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 Kaggle submission from you that is different from the guided exercises. If you further optimise a hyperparameter or two using nested cross-validation that is fine. It would be nice if your tweaked classifier does better** (but if it doesn't: that's life)**!**

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 between 15:05 and 17:00 on the 15th of March 2023.**

Extra information

You can do a maximum of 20 submissions per day on Kaggle. This should be plenty. **The competition closes on the 15th of March at 15:05**, so submit your last result before that. It's fine if you've already stopped working before that moment and spend some time revising what we've learned in the course.