Project description and instruction

Introduction

After all you've learned, the proof should be 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. You can also search through a slew of data science cheat sheets on Kaggle here. It is highly recommended to look through these. Be sure to share good ones with your fellow students!

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.
6. When one of the steps is completed, you need to submit your predictions for the test dataset. You will be tested on the AUC ROC of your classifier. You can use this score function from sklearn to get ROC AUC values in your pipelines.
7. After the guided exercises, the assignment is simple: get the best performance on the test dataset. You are free to do whatever you like. Try different classifiers, train a neural network, combine different classifiers through voting, use unsupervised learning to find some structure and investigate that further: your imagination (and time, and programming prowess, to be fair) is the limit. The goal is to get the best predictive performance on the test dataset.
8. **The competition closes at 12:15 on Wednesday 22-12-2021.** So submit your final result at 12:00. You can use the remainder of the time to work on the presentation, or think of questions to ask for the question hour, or study for the test.

To submit on Kaggle, you simply upload a .csv with 2 columns: one called "Id" with the IDs of the samples, 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)

Dataset description

TL;DR: 4200 training samples, 800 test samples. 10.000 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 10.000 (so **10.000 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. Start out by pretending you're a bumbling fool who just found out about this new-fangled thing called Masjeen Lurning. Just split your data into a train and validation set (**use random_state=24** to make things comparable between groups), train a logistic regression without regularisation, and immediately predict with this model on the test set. To submit your predictions, make sure to submit a file with the columns "Id", "Outcome", where "Id" is the test sample's ID (you get it in the data), and "Outcome" is your class prediction (0, 1, 2, or 3).
   Submit a .csv with your predictions on the test set to Kaggle. Voila, a terrible baseline performance!

2. Do some exploratory data analysis. How many missing data is 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 others? What is the maximum value in the data, and the minimum? And which feature has the highest variance? Don't just look globally, also look at (some of) these metrics by class. There might be differences, after all.

   Nothing to submit for this step, but I expect a short comment on these questions and what you found in your presentation.

3. You've probably found that there's missing data. First, take care of the class imbalance problem in the brute-force way described above: just randomly downsample the data so that you have equal numbers of each class. For your next step, make a pipeline that does three things:
   -it should scale each value to 0 mean and unit variance
   -it should use a KNN imputer to impute missing data (remove nan)
   -it should again use a logistic regression without regularisation to predict the class.
   Again, train on a validation set, and predict on the test set. I don't ask you to use a pipeline for fun and games: imputing on values in the validation or, worse, the test set, is wrong!
   Submit a .csv with your predictions on the test set to Kaggle.

4. This data has many dimensions. You are hence probably overfitting like crazy. Let's not do that. First, by yourself, do a PCA on your (normalised, imputed) training data and make a plot of the first 2 PCs. 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 those two questions in your presentation.
   For the next part, let's use PCA in your prediction pipeline. Insert a PCA step where you think it fits and use 200 principal components. Then, switch out your train-test split with actual cross-validation, and training a final model on *all* training data. You then use this final model to predict on the test set. Still use unregularized logistic regression.

Submit a .csv with your predictions on the test set to Kaggle.

5. There's two things still 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 RandomSearchCV over n_components for the PCA [10, 50, 100, 200, 400, 800] and n_neighbours for the KNNImputation [5, 10, 20, 40, 80, 150]. See what the best hyperparameters are. Finally, train a classifier on all the train data using these best hyperparameters, and make predictions on the test set.
Submit a .csv with your predictions on the test set to Kaggle

6. Now, you can switch out the classifier you use. Indeed, down the line you can combine them. For now, I want to introduce you to two classifiers we haven't covered in detail, an SVM and a Random Forest. Let's start with the Random Forest.
A Random Forest 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 (though there's no harm), and calculates its own *feature importances*, which tell you how important certain features are for its correct classifications.
I want you to watch (the material that you don't know from) these 3 videos:
one ; two; three
Now that you know what's happening, use a Random Forest rather than an unregularized logistic regression to make your classifications. You may use default parameters. Note that, as the video says, RFs use their own weighted KNN Imputation internally to deal with missing data. 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.
I want you to report the 5 features that had the highest feature importance (and the feature importances) in your presentation, along with plots that show the distribution of these features per class in the training data (see this).
Submit a .csv with your predictions made with a Random Forest

7. There is also the Support Vector Machine or SVM, so-called because it saves the datapoints that define the decision boundary in the end. This algorithm reigned supreme before deep (convolutional) learning got going.
I want you to watch (the material you don't yet know from) these 3 videos:
one; two; three
Now that you know what's happening, use an SVM rather than a Random Forest or unregularized logistic regression. Set which kernel to use as a hyperparameter (choosing from the polynomial or radial basis kernel). Train using nested cross-validation on the train data, train a final classifier with the best hyperparameters, and then predict on the test data.
Submit a .csv with your predictions made with an SVM

8. Finally, I want you to train a simple feedforward dense 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, KNNImputer, and PCA step in your pipeline.
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 ROC AUC 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.

What I expect

I expect that the first 8 steps take you at least until Tuesday afternoon 13:15, but perhaps more like 15:00-16:00 depending on how well you got through the morning practicals on Monday. Hence, I assume you have something like 4 or 5 hours to try something extra. **I want *at least* one extra submission from each Team that is different from the guided exercises. It would be nice if it does better!**

Presentations

I want you to give a short presentation (online) to the rest of the group. Shoot for **7.5 minutes of presentation time, with 2 minutes for questions**. I require you to talk shortly about the underlined sections above, that is, some characteristics like missing data numbers etc, a PCA plot in 2 dimensions and the highest contributing original dimensions to each, and the top 5 feature importances from your Random Forest and a plot of the distribution of those feature's values for each class. After that, talk about what you did to make your final classifier, and show your average generalisation score (i.e. average ROC AUC score from your outer cross-validation). **I want to hear about at least one problem that you ran into and how you resolved it** (this can be a 'boring' problem, namely that you did something wrong from the start and didn't realise it, or a coding error that you spent *way* too long fixing). You can probably get quite far with 5-10 slides. I will cut you off after 10 minutes.

Code

You **must submit the code you used to build your final classifier to me.** It should either be a Python file (.py) or an Jupyter Notebook (.ipynb). I will check it for big oversights and consistency with your results.

Extra information

Remember, to get ROC AUC scores, check this. **You can do a maximum of 20 submissions per day on Kaggle.** This should be plenty. **The competition closes at 12:15 on Wednesday the 22nd**, so submit your last result at 12:00. It's fine if you've already stopped working before that and spend some time revising what we've learned in the course.