Statistical Machine Learning

Assessed Practical

P244 P272 Pxxx

Week 8, Hilary Term 2018

Contents

Abstract

In this report, we consider the problem of determining the gender of an individual in a picture. More precisely, using a set of features derived from the original images, we try to fit a machine learning model that can accurately classify individuals as male and female, and accurately represent its confidence in the classification.

The performance of the model was evaluated through a Kaggle competition, in which we submitted predictions as team "The Poisson Fishermen".

1 The data

The data available are pre-processed data from male and female pictures. Each picture is represented by a 128-numbers long vector of features and a label: 0 for male, 1 for female.

The labelled training set is composed of 15 000 observations, about half of which are of female individuals. The recorded features are all roughly centered and have standard deviations close to 0.9.

Exploratory data analysis does not show any feature particularly standing out. After a PCA, even the first principal component carries only around 3% of the total variance, which suggests variability is widespread across features. Likewise, the Spearman correlation between indiviudal features and the labels are rather low (all between -0.3 and 0.3 with most much closer to 0).

Despite this lack of interpretability of the features, the data shows good separability.A simple logistic regression gives 92% accuracy when predicting the labels. Likewise, using T-SNE (t-distributed stochastic neighbor embedding), an unsupervised method that give a 2D representation of the dataset, clearly separates males and females. This is promising for the task at hand, since it means that separation of males and females is quite feasible.

1.1 Evaluating of the model

To evaluate the performances of our models, the log loss, aka logistic loss or cross-entropy loss, is used:

$$-\log(y|\hat{y}) = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

with y the true label and $\hat{y}$ the predicted one. A particularity of this loss is that high confidence in a wrong prediction is very heavily penalized. This means that not only classification should be accurate, but predictions must also be conservative when there is some doubt on the label.

2 Building the model

2.1 Basic classifiers

Given the good separability of the data, we started by experimenting with very simple algorithms. Our first successful attempts was the Quadratic Discriminant Analysis (QDA), which both ran almost instantaneously and yielded an accuracy over 98%. K-Nearest neighbours with 10 neighbours had similar precision.

However, in both cases the log loss remained higher than even the simple logistic regression (even though its classification performance was lower). This is because the previous algorithms are not well-calibrated: the exact value that they give when predicting a probability is not representative of their true classification performance. This causes them to be over- or under-confident in their probability estimates and they are thus heavily penalized by the log loss metric. Although some methods exist to calibrate such algorithms (using an isotonic regression for instance), we chose to move on to algorithms that could directly optimize the right metric.

This is the case of the xgboost package, which can perform gradient boosting on trees using a log loss. This method showed promising results but failed to consistently bring the average loss below 0.1.

Having gone through those and a number of other standard methods, we turned to deep learning for comparison. After minimal tuning, a Multi-Layer Perceptron (MLP) with two hidden

layers gave an average log loss around 0.08. Considering the wide gap between the performance of this algorithm and the others, we chose to focus the rest of our study on neural models.

Before moving on to more complex neural network, we tried to make use of the work mentioned above to derive some useful features for our final model.

2.2 Generating new features

To try and derive interesting features from the data, one possibility is to use the predictions from lower-performing models as new covariates for the final model. We chose to use the outputs from the K-Neighbors and QDA for that purpose: since they are not linearly derived from the data, they may provide some information that would take a lot of computation for the neural network to find on its own.

We also tried to obtain condensed features with a simple autoencoder implemented with a multi-layers perceptron. It consists of three layers: the input and the output are composed of 128 nodes, while the hidden layer is composed of 32 neurons. By training the NN with identical input and output, the weights are such that the output of the hidden layer is an encoding that minimises the loss of information. Theoretically, this allows for a more informative representation of the data since the compressed representation from the hidden layer will reduce the noise of the data and keep its most informative, so that the output layer can decode this representation and reconstruct the initial data.

The third path we explored to generate new features was to use non-linear embeddings of the data in lower-dimensional space. As previously mentioned, although linear methods such as the PCA and LDA yielded disappointing separation, more advanced ones such as spectral embedding and T-SNE. We experimented with those, adding the projected coordinates as new covariates for the neural model.

All in all, the most successful addition was that of the QDA, which gave the model a noticeable improvement. Some of the other artificial features appeared to give a slight boost to the classification but since the difference was extremely small and the computation of these features quite costly, we chose to keep only the QDA.

3 The final model

3.1 Improving the neural network

To improve on the simple MLP model's performances, we worked on building a more advanced neural network using the dedicated PyTorch framework. In particular a gradient descent with momentum (Adam algorithm) significantly improved the model.

Quite quickly, it became apparent that using more than one hidden layer was counter-productive and resulted in strong overfitting. More generally, the main difficulty was to find ways to improve the fit of the model without harming its generalization. When training the model on part of the data and validating on a new set of rows, it became apparent that the performance of the model depended heavily on the split between the training and validation data. This high variability in the performance, associated with the large difference between the training and validation error, called for regularization.

L2 regularization, implemented through weight decay improved the overall performance but did not do much to reduce variability. Dropout — disabling some of the neurons during training

to force generalization — had a similar effect. We also experimented with batch normalization which did not help much.

The end result of our efforts was to bring our public leaderboard score do 0.074. In local validation, performance varied wildly, from 0.065 to almost 0.08. This is what led us to try and find a way to reduce variability further.

3.2 Preventing overfitting

The main problem of our base network was large overfitting to the training data. To prevent this, we tested multiple possible solutions.

The first, obvious method, is to adjust the optimizer and its *learning rate.* The Adam method is very common in classification networks, but we also tested a simple stochastic gradient descent. We determined the optimal learning rate by cross-validation, taking into account the number of epochs to train the network. But the learning rate is not the only relevant hyperparameter: we can also introduce other regularisation methods, such as momentum for SGD and L2 regularisation *(weight decay)* for Adam. Weight decay has given excellent results to limit overfitting, and has improved the neural network results both in overall performance and in variability.

The second important regularisation method is to add a *dropout layer* after the dense hidden layer. In our final configuration (determined by cross validation), 50% of the hidden units are randomly set to zero during training. This helps a lot to control the variability of the output.

Another method frequently used to control overfitting is *batch normalisation.* It allows to normalise each minibatch during training. As we normalise the entire dataset before training, it performs a similar operation after the hidden layer. However, neither the performance nor the variablity of the log loss was significantly impacted by this additional layer.

3.3 Ensembling trees

Then the final probability is a weighted average of the output of each model. The weights are based on the performances of this NN (measured by cross-validation). [Add weights formula **TBC**]

3.4 The road to 0.06

4 Conclusion