Shubh Maheshwari
20161170

Report

Feature	Classifier	Accuracy	F1 Score(micro)
Raw Pixels	Soft Margin Linear SVM	30.02%	29.02%
Raw Pixels	Logistic Regression	32.8666%	31.8666%
Raw Pixels	MLP	52.07%	48.07%
Raw Pixels	Kernel SVM with RBF Kernel	40.547%	41.547%
LDA	Soft Margin Linear SVM	38.344%	35.40%
LDA	Logistic Regression	23.31%	21.3099%
LDA	MLP	32.94%	34.94%
LDA	Kernel SVM with RBF Kernel	37.41%	37.4099%
PCA	Soft Margin Linear SVM	33.2%	33.2%
PCA	Logistic Regression	35.78%	34.77%
PCA	MLP	29.33%	29.33%
PCA	Kernel SVM with RBF Kernel	44.92%	44.22%

Execution of the codes

```
python3 <dimensionality_reduction><classifier>.py
```

Summary of the codes

We got to experiment with multiple hyperparameters and also learnt about various **scikit-learn**libraries and
how to use them. The codes can take **hyperparameters**as user inputs and also show the options via **CLI**. The above code has assumed f1 score averaging to be done using **micro**.

Observations

We found very interesting observations while tweaking the hyperparameters. They are as follows-

- **Raw Data + MLP**gave **52.07%**accuracy and f1 score after changing the hyperparameters like number of hidden layer numbers, number of iterations and activation functions. In its default format. Linear Classifier would only give around 10% Accuracy but with right hidden layers and activation we can achieve 48% score.

- **Raw Data**gives high accuracy and f1 scores without any dimensionality reduction techniques but takes a lot of time.

- Solver **lbfgs**provides maximum accuracy and f1 score in **MLP**.

- Activation functions **tanh**and **relu**in **MLP**give almost the same accuracy and f1 score. Sigmoid shows vanishing gradient
- Increasing learning rate decreases accuracy and f1 score in **MLP** classifier. We were not using any gradient descent optimizaion like Adam, etc.

  .

- Reducing tolerance in **Soft Margin Linear SVM** reduces time but also accuracy and f1 score.
- Keeping loss as **hinged**in **Soft Margin Linear SVM** reduces accuracy and f1 score as compared to **squared_hinge**.

- Reducing tolerance in **Kernel SVM with RBF as kernel** reduces time but also accuracy and f1 score.
- Using **shrinking**heuristic in **Kernel SVM with RBF as kernel** increases accuracy and f1 score.
- Limiting maximum iterations in both **Kernel SVM with RBF as kernel**and **Soft Margin Linear SVM**reduces accuracy and f1 score by as much as **11%**.
- Reducing tolerance in **Logistic Regression**reduces time but also accuracy and f1 score.
- Reducing tolerance and number of components in **LDA** reduces time but also accuracy and f1 score but also reduces time taken to compute.
- Reducing number of components and keeping whiten=False in **PCA**reduces accuracy and f1 score but also reduces time taken to compute.

Problem of Overfitting

One way to quantify overfitting is as the difference between the validation accuracy and the training accuracy. So this notion of overfitting already occurs on the existing dataset. Another notion of overfitting is the gap between the test accuracy and the accuracy on the underlying data distribution. By adapting model design choices to the test set, the concern is that we implicitly fit the model to the test set. The test accuracy then loses its validity as an accurate measure of performance on truly unseen data.

To remove overfitting, we create another dataset called as CIFAR-10.1 and we measure the accuracy of CIFAR-10 classifiers by creating a new test set of truly unseen images.The data collection for CIFAR-10.1 will be designed to minimize the distribution shift relative to the original dataset.

Problems Faced

The main problem was with raw pixel data because of the amount of time spent on training and predicting was very large. This needed high computing power. Also changing and playing with the **hyperparameters**of the dimensionality reduction techniques and classifiers needed repeated execution of code.

Reasons for low accracy

1. Very less data. Due to the time required to train the model on cpu I could only train on 1000 images

2. High Numbers of parameters W = 45*(D*2) D = 32 – 3072

3. If we had used convolution networks we could have easily achieved 80-90% accuracy as shown many individuals

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html