Project-II by Group Sydney

Diego Antognini & Jason Racine
EPFL
diego.antognini@epfl.ch, jason.racine@epfl.ch

Abstract

This report provides a summary of the project two of the PCML class. The project consists of doing binary and multi-class classification on the same set of images. For both methods, we have observed that the given CNN features were more accurate than the HOG one. In binary classification, all methods were approximately equivalent except the Adaboosting and the SVM one. For the multi-class classification, neural networks and SVM were the best methods.

1 Data Description

The train-data for binary classification consists of $N = 6000$ images. As input we have 2 representations of the image : *the histogram of oriented gradients* (HOG) $\mathbf{X}_{hog}$ and the *overFEAT ImageNet CNN features* (CNN) $\mathbf{X}_{cnn}$. Our input $\mathbf{X}$ is the concatenation of $\mathbf{X}_{hog}$ with $\mathbf{X}_{cnn}$.

Each input sample is a vector $\mathbf{x}_n$ with dimension $D = 42273$ (5408 for the HOG and 36865 for the CNN) and is the concatenation of $\mathbf{x}_n^{hog}$ with $\mathbf{x}_n^{cnn}$ The output ($\mathbf{y}$) represents the classification of the images. For the binary classification, the label 1 represents a car, a horse or a plane and the label 2 anything else. For the multi-class classification, the label 1 represents a car, 22 a horse, 3 a plane and 4 anything else.

We also have test-data of size $N = 11453$ without their corresponding output. Our goal is to produce predictions for those data, as well as an approximation of the test-error using *Balanced Error Rate* (BER).

1.1 Histogram of Oriented Gradients

Histogram of oriented gradients is a feature used in computer vision in order to detect objects. To compute it, we first need to normalize the image, compute the gradients (of each pixel) for the different color channel and take those with the largest norm. Then we decompose the image in bins of size $w \times h$. For each of those bins, we compute an histogram of the orientation of the gradients using theirs angles and weighted by theirs magnitudes. The histogram has n bins from 0 to 180 degrees. We compute this histogram using 4 different normalizations.

In our case, the dimensions of this feature is $13 \times 13 \times 32 = 5408$, where the first 13 is the number of bins in the height, the second 13 the number of bins in the width. Finally 32 is 4×8 where the 4 is the different normalizations for the histogram and 8 the number of bins (each bin has a size of 22.5).

1.2 OverFEAT ImageNet CNN features

Those features are extracted from a convolutional network-based image features extractor OverFeat. They were trained on the ImageNet dataset (tens of millions images). The size is the output of the fifth layer of the neural network which is $1024 \times 6 \times 6 = 36864$. In our features, we also have an extra dimension which can be ruled out.

2 Data visualization and analysis

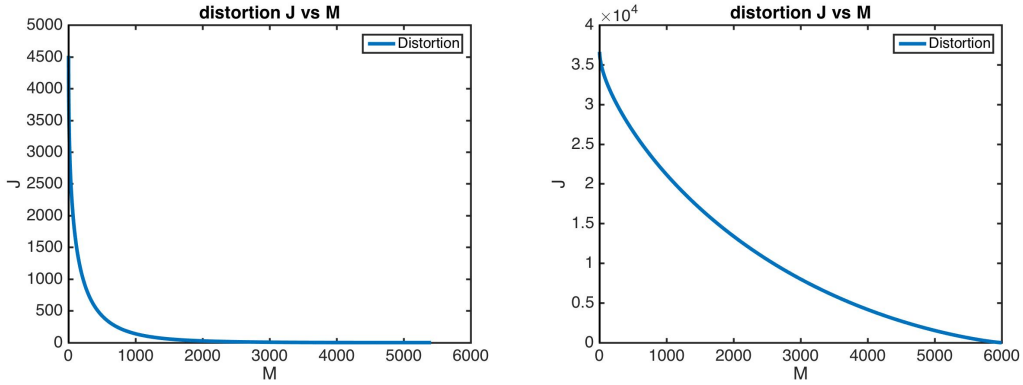
We first have plot the labels distribution for the binary and multi-class cases. For the binary case, we have 60.3% for the label 1 and 39.8% for the second one. We have observed that there is a small imbalance between the classes. For the multi-classes case, we obtain 16.07%, 19.37%, 24.87% and 39.70% for the labels 1, 2, 3 and 4. This time, the imbalance is more important, especially with the label 4 and we do have to take care about it during the training.

We also wanted to analyze the distribution of our features, and have observed that for the features HOG, each of them has a has similar distribution. The values are essentially in the range $[0, 0.2]$. We are also interested about the correlation between the input and output variables. We have observed that the correlation is in the range $[-0.24; 0.30]$, and so can conclude that features seem not to have a direct correlation between them. However, a combination of features can have a high correlation with the output but we couldn't find such combination. Finally, $\mathbf{X}_{hog}$ is rank-deficient with a rank of 3467 instead of 5408.

For the features CNN, first we can see that it is a sparse matrix and most of the values for each data is 0 (around 95.5%). This means that for each data, a small subset is able to describe the picture. Moreover, we can observe that we are faced to the problem of $D > N$ and the solution we have adopted is PCA which will describe during the next sections. The values are in mainly in the range $[0, 64.87]$. We also wanted to see the correlation of those features with the output and have found that they are in the range $[-0.24; 0.18]$, which is similar to the HOG features. Finally, $\mathbf{X}_{cnn}$ is rank-deficient with a rank of 5997 instead of 36865.

3 Techniques used

This section is a description of the techniques we have used in the project. It describes mainly how it works, why we thought it could be useful in this project.



(a) Plot of the distortion J to M where $M = D$. The distortion decreases quickly.

(b) Plot of the distortion J to M where $M = N$ because there are too many features and so, the distortion doesn't go to 0 and decreases slowly.

Figure 1:

3.1 Principal component analysis

As we have seen in the section "Data visualization and analysis", we are faced to the $D > N$ problem mainly with the CNN features and also the HOG features depending on how we se the k for the k-fold cross validation. One way to solve this problem is to use PCA. Basically, PCA works as follow : we compute the mean and the covariance matrix $S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T$ of our data and then compute the M eigenvectors of S corresponding to the M largest eigenvalues. M is the number of features we have to take into account in order to minimize the reconstruction. Finally, the approximation of a data sample is $\tilde{x_n} = \sum_{i=1}^{M} x_{ni}u_i$. However, we couldn't compute

PCA using this method, because computing the full eigenvector decomposition a matrix $D \times D$ runs in $O(D^3)$ and D is quite large. We have used an alternative proposed in [REF] which allows us to compute the same eigenvectors in $O(N^3)$, which is roughly at least 230 times faster. The tricks is to express $S = \frac{1}{N} XX^T$ and finally using $\frac{1}{N} XX^T v_i = \lambda_i v_i$, where $v_i = Xu_i$, we obtain an eigenvector equation for the $N \times N$ matrix S.

The last parameter to set, is the parameter M which defines the M largest eigenvectors to take into consideration for our feature selection.We introduce a distortion measure $J = \frac{1}{N} \sum_{n=1}^{N} ||x_n - \tilde{x_n}||^2$ which represents the mean squared distance between the original data sample and our approximation. Without going into the steps, the solution to the minimization of J is $J = \sum_{i=M+1}^{D} \lambda_i$.

The Figure 1(a) shows the plot of J for the HOG features and the CNN features. Figure 1(b) shows the distortion of the CNN features, we have tried several values for M and have conclude that $M_{CNN} = 150$ was a good value allowing us to compute faster results without loosing too much accuracy. The corresponding distortion might be very high compared to the HOG features, but the range of the values are not the same, CNN has values going larger than 0.2. We tried some bigger values but we didn't get major improvements in terms of error. For the HOG features, the distortion measure seems more nice, we did the same experiments and finally set M_{HOG} = 500.

3.2 Neural Networks

This is the multi-layer perceptron (MLP) we have seen in class. The neural networks used in this project comes from the given DeepLearning matlab toolbox [REF]. It uses batch sizes in order to use less memory (take the first n data samples and works on them, then the next n etc). We have let the default parameter (100). The number of epochs is simply the number of time where all the training data do a forward and a backward pass. We have put this value to 20, in order to have a little more accuracy without loosing a lot of time, it was a good compromise because larger epochs don't mean especially better accuracy. The number of input is M and output is either 2 or 4, depending of the type of classification (binary or multi-class). For the number of hidden layer, we have let 1 because we didn't see improvement by adding some more. However, the number of neurons in the hidden layer has been tested with k-fold cross validation and we have found 1000 for the binary classification and 700 for the multi-class case. We also have to set up the learning rate : the optimal value we have found is 3 for the binary case and 2 for the multi-class one. By default, the activation function used is the *tanh* one. We didn't see so much different using the *sigmoid*. Finally, the algorithm uses stochastic gradient descent using a momentum of 0.5, which might help to converge faster.

Neural networks are suitable for problems which might be difficult to describe in a mathematically manner and is robust to the noise, which is very useful in our case. However, we have to be careful with the weights initialization because neural networks are very sensitive to them, and especially with overfitting and computing the generalization error, this task is harder with neural networks.

3.3 Support Vector Machine

The normal support vector machine is a technique used for binary classification. The goal is to divide the space using an hyperplan in a way to maximize the margin, which represents the distance between closest members of the two classes. It is an interesting method because it has a regularization parameter which help to avoid overfitting. Another advantages is we can easily change the kernel function in order to try to find better boundaries. However, what is difficult is to find good parameters (C, γ). γ is the kernel scale parameter which is related to how the data are scattered. We might have many support vectors and overfit if we set the C too large and on the contrary if it is small, we might have not enough support vector and underfit. However, the best results we can obtain with support vector machine need to have a small imbalanced between the positive and negative examples. As seen in the section "Data visualization and analysis", there classes are not very well balanced as much for the binary classification as the multi-class classification. We have use the same algorithm we have seen in the lab SMO.

We have used the method *fitcecoc* in matlab. We have used k-fold cross valiation to find C and γ and have set the margin $C = 1$, and $\gamma = 1$. Only the linear kernel function was available in this mode, and so, gaussian kernel might give better results. Even if the method is linear, SVM only classify

directly the dataset in two classes, and so it is not a problem with the multi-class case because we always classify data in one of the two classes for the current classifier (see *OVA* below).

To adapt support vector machine to a multi-class classifier, we have used *One Versus All* method, which divides the problem in 4 binary classifications. We train the data with all the classifiers and at the end, each classifier says if the data is positive to its class. If there are more than one classifier with the data sample considered as positive, we use the highest confidence to associate the class. Another possible method would have been to use *One Versus One* which needs 6 classifiers in our case, and is less sensitive to the imbalance but needs more computational resource, but it didn't give better results.

3.4 Bootstrap Aggregation (Bagging)

As we have seen in the lecture, decision trees are good because the training and the prediction is quite fast, even with many features or large dataset but have a high variance and might overfit. The idea of decision trees, is to have a several questions where to answer yes/no and to move in the tree. At the end, the leaves give the corresponding class. To construct this list of questions, we divide recursively the space in a way to minimize the cost of impurity (depending of the threshold to find and the current feature). The depth of the tree is a parameter.

The idea of bootstrap aggregation is to average several decision trees. It will help to reduce the variance and avoid overfitting. It generates m new training sets of the same size, by sampling uniformly and with replacement the original training set. Then we have m decision trees which will grows independently with their own generated training sets. Finally, to predict the class it takes the average over all the trees. In order to find those parameters m and the maximum depth of the tree, we have also used k-fold cross validation and the best values are $m = 200$ and maximum depth of 512 for the binary classification and for the multi-class case $m = 500$ and maximum depth also of 512. Having a high maximum depth might lead to overfit because we specialize to much the features split, but if it is too small, it might underfit because the features are not well split in order to identify classes. For the parameter m, it is better to have a higher number in order to have a large number of trees which can give a better accuracy of the prediction.

We have used two different methods in matlab, one using *fitensemble* with the parameter 'Bag' and another one which is the *TreeBagger*.

3.5 Random Forests

The idea of random forests is similar than with bagging in a sense that we train several trees and take the average predictions. Like bagging, each trees have trained on a subset of the dataset. However, there is another aspect in random forest that bagging doesn't have: for each split, instead of searching among the overall features, we search only among a random subset of features. The size of this subset is a parameter m as well as the maximum depth tree and the number of trees, like in bagging. Like bagging, random forests are useful because they help reducing the variance and the overfiting. But the more interesting ideas is to select only a subset of the features in the split process. Moreover, it is efficient against imbalanced dataset.

As in bagging trees, the number of trees and the maximum depth tree behaves in the same manner. The last parameter m has an important impact because it is too small, the subset of features might not describe correctly the data, and if it is too high, this subset might be to high and might not represent well the data. We have to find a good compromise in order to have a certain number of features to describe the data. Still with k fold cross validation, we have found for the multi-class case a maximum depth of 1024, 64 trees and a $m = 20$. For m, an heuristic could be the square root of the number of features, but it was less efficient than 20 in our case. For the binary case, those parameters are 1024, 256 and 50.

The given Piotr matlab toolboox [REF] contains a method *forestTrain* which takes the defined parameters and *forestApply* for the prediction.

3.6 Adaboosting

Adaboosting is another way to average trees. If we observe more deeply the bagging approach, some trees might have a very high variance and some a very slow. Averaging all those trees decrease the variance and have a small bias. Boosting is the opposite : decrease the bias when averaging and have a small variance. In order to achieve this, each data sample has an assigned non-negative weight. After each iteration, if the data sample is misclassified, its weight is increased for the next iteration. So misclassified data samples will have more importance. The parameters are the same as in the bagging and behave in the same manner. The Piotr matlab toolboox [REF] contains a method *adaBoostTrain* and *adaBoostApply* for the prediction. We have only used it for the binary classification, because the implementation is not adapted for more classes. The parameters we obtain with k fold cross validation are 1024 for the number of trees and 4 for the maximum depth.

4 Evaluation methods

In order to evaluate our different models, we have split our data in two sets of 80% and 20% for the training and the test sets. We choose this percentage in order to have still enough data for the training set and also enough to estimate correctly our models on the testing set. We learn only on the training set and then use the test set in order to estimate the error using the *Balanced Error Rate*. This error compute the average percentage of misclassification per class. It allows us to avoid the case of the $90\%/10\%$ and having a good error. We have also used k-fold cross validation to tune the parameters up of our different models. We set $k = 10$ because we have a lot of data, splitting them into 10 folds seems reasonable (4320 data samples for the training set and 480 for the testing set). We repeated the experiment 30 times with different seeds in order to split the data in a unique random way for each trial and so try to have an unbiased estimation error.

5 Model comparison

This section describes our models for the binary classification as well as the multi-class classification. We have essentially put our effort on the CNN features. The reason is because with only the HOG features, we couldn't get an error lower than 20% for binary and multi-class classification. With the CNN features, we didn't have this problem. We have tried to take the whole matrix, only a subset of features with $M = 500$, different methods but nothing worked. Contrary to the CNN features, we think that the reason is because those features are too noisy : they came directly form the images. Some of them are rotated, inversed and so, the gradient is also transformed, which might be confusing for the learning. The CNN features are already computed from a convolutional neural networks which gives the features it has seen on the image with a specific weight. Because each sample has approximately 5% of the overall features, they are fewer and are more specific than the HOG features.

5.1 Binary classification

The Figure 2 represents the BER error for each models. The first model used simply returns the label 1 to all inputs, which will allow us to compare improvements with other models. It gives us an error of 0.5, which is obvious because all the input are in the same class and the other has none of them. The second model uses neural networks with normalized data because we got better results and because it can avoid being stuck in a local minima and might speed the training. We have used it It clearly outperforms the baseline. We can see that it has a bit of variance but has a low bias. Increasing the number of hidden layer might accentuate the variance and diminish the bias.

Our third and fourth model uses bootstrap aggregation with different implementations. They have similar performance than neural networks. In the first one, we can see that the variance is lower but there are some outliers. The other one has a bigger variance but is more robust to outliers. Let's remind than bagging averages trees and some trees might have big and small variance, which reduce variance in average. In general, it reduces the variance and increase slightly the bias. Compared to the neural network, the variance is better but the bias surely is worst.

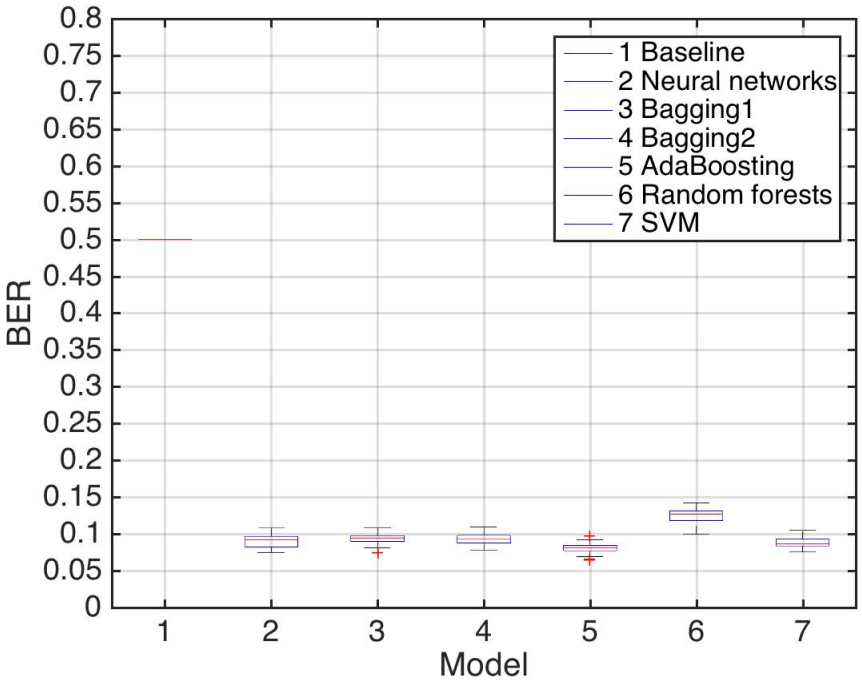


Figure 2: Boxplot of our models for the binary classification, using the BER. Each model is represented with a box where we can see their mean and their standard deviation.

The next model (5) is the boosting method. As in bagger, it reduces the variance because it takes the average and the bias should be lower, which is the contrary compared to the bagger. We can also see that there are some outliers, but they are still better than the other seen methods. We think that the main reason is the variance and seems to be higher than the other models. In general, boosting should be better than bagger.

Random forests is used for the sixth model. Surprisingly, we obtain worse results than the other methods. The k fold cross validation gave us better results when we were looking for the parameters. A possible cause of this effect is that it has overfit during the tuning process and so, gives worse performance in the real case. The parameter concerned should be the maximum depth which should be too high for our data because it specializes too much the features during the split. In theory, it should have better performance than the bagging and have less variances because we split only on a subset of features, however, the bias should be higher. Beside this, maybe the bias is too high and in this case, increasing the subset of feature might help to fix this.

Our last model used the SVM. Compared to the other methods, we can observe that it is quite stable and have a low variance. If the parameter C is set optimally, we should have a good variance and a good bias. However, if it is too high, the variance should be small but the bias could be high. In the other side, if C is small, the bias should be lower but we have fewer support vectors which leads to a higher variance. However, let's remind that we use the same as in the multi-class classifier, using a linear kernel due to the implementation of matlab. If we could have changed the kernel into a non-linear one, the decision boundary would be different and might surely reduce the error. However, we would be faced to γ which might change the variance and the bias.

Our final model will be the Adaboosting one, because we considered it has the smallest error, even if there are some outliers, those are still lower than others methods in general.

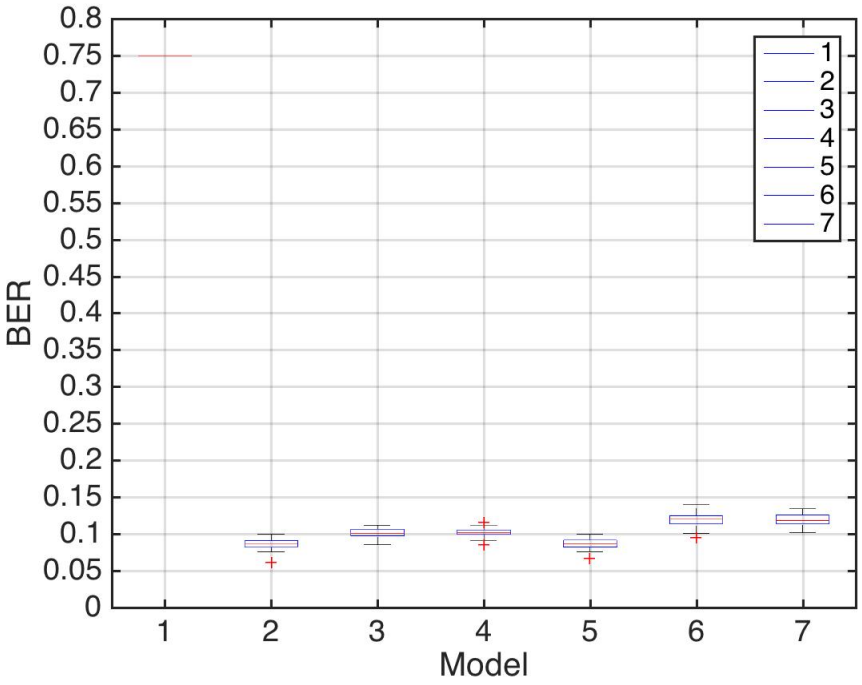


Figure 3: Boxplot of our models for the multi-class classification, using the BER. Each model is represented with a box where we can see their mean and their standard deviation.

5.2 Multi-class classification

6 Feature transformation

7 Conclusion

8 References

@miscDymolaWeb, title = "xxx Website", author = "xxx", howpublished = "`http://www.xxx.com`", year = "cited July 2008"

@miscIMM2012-06284, author = "R. B. Palm", title = "Prediction as a candidate for learning deep hierarchical models of data", year = "2012",

@miscPMT, author = Piotr Dollár, title = Piotr's Computer Vision Matlab Toolbox (PMT), howpublished = `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`