Project-II by Group Barcelona

Eric Francis Bezzam
EPFL
eric.bezzam@epfl.ch

Manish Thani Awtaney
EPFL
manish.thaniawtaney@epfl.ch

Abstract

This report summarizes our observations and results for Project-II of the EPFL course entitled "Pattern Recognition and Machine Learning." The work was done over the course of one month from November 24, 2015 to December 21, 2015. We were given a large dataset of images that can be classified into four groups: airplanes, cars, horses, and other. Our objective was to build a system that uses *Histogram of Oriented Gradient* (HOG) and *OverFEAT ImageNet Convolutional Neural Network* (CNN) features to differentiate between these four classes. We investigated several models for this task of classification: $k-$Nearest Neighbor ($k-$NN), Support Vector Machine, Random Forests, and Neural Networks.

1 Data Description

We are given $N_{tr} = 6000$ images for training. Each image $\mathbf{x_n}$ has 42273 features: 36865 CNN values and 5408 HOG values. Each training image has a corresponding label, y_n, indicating which of four classes it belongs to: 1 for 'airplane', 2 for 'car', 3 for 'horse', and 4 for 'other'.

For testing, we are given $N_{te} = 11453$ test examples. We do not have access to the images but rather the 36865 CNN values and 5408 HOG values. However, we are not provided with the labels y_n indicating which class these test examples belong to.

2 Objective

Our objective is to produce predictions for the test examples and to approximate the test-error using the Balanced Error Rate (BER):

$$BER = \frac{1}{C}\sum_{c=1}^{C}\left[\frac{1}{N_c}\sum_{n=1}^{N}(y_n - c)(y_n \neq \hat{y}_n)\right], \tag{1}$$

where C is the number of classes, N_c is the number of examples in class c, y_n is the ground truth for sample n, and $\hat{y}_n$ is the corresponding prediction.

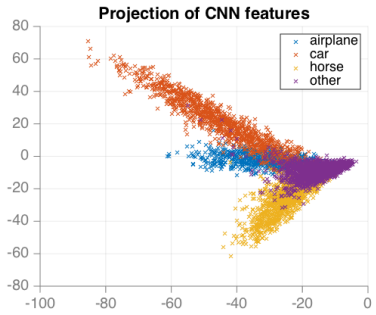
We perform two types of predictions:

1. *Binary*: we predict whether an image contains an airplane, car, or horse (positive case) or if it is part of the 'other' class (negative case), i.e. $y \in \{0, 1\}$.
2. *Multi-class*: we predict whether an image belongs to the 'airplane', 'car', 'horse', or 'other' class, i.e. $y \in \{1, 2, 3, 4\}$.

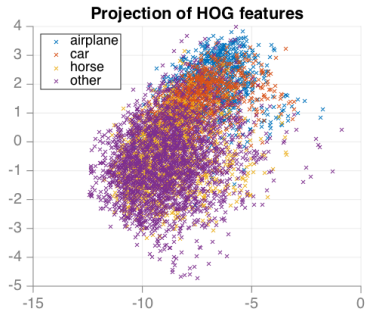
3 Exploratory Data Analysis

As each input vector for an image contains a very large amount of features (namely 42273), we used Principal Component Analysis to reduce the dimensionality to see if significant patterns can be

observed in the orthogonal components with the most variance. We plotted the two most prominent principal components for the CNN and HOG features. The plots can be seen in Figure 1(a) and Figure 1(b).



(a) Two most significant principal components of the CNN features.



(b) Two most significant principal components of the HOG features.

Figure 1: Principle Component Analysis of CNN and HOG features

As can be seen from Figure 1(a) and Figure 1(b), the CNN features will certainly play a more significant role in differentiating between the four classes as four clusters corresponding to the four classes can be seen from just the two most significant principal components. In the case of the HOG features, four clear clusters corresponding to the four classes can not be observed.

4 Feature processing

Although we observed from PCA that the CNN features may be more useful than the HOG features for distinguishing between the four classes, we will use both CNN and HOG features for our classification task. However, it is necessary to project this feature vector of length 42273 to smaller dimension as to avoid over-fitting and to reduce computational cost. For each model, we have the number of principal components as a hyper-parameter that will be optimized using $K-$ fold cross-validation.

The built-in MATLAB function `svds(X,N)` is used to compute the matrix V of size $42273 \times N$, where N is the desired reduced dimensionality. The output of the function consists of the N largest singular values and associated singular vectors of the matrix X [6]. However, we are only interested in the singular vectors that form the columns of V as this can be used to map X to a dimensionality of N as such: $X_{reduced} = XV$.

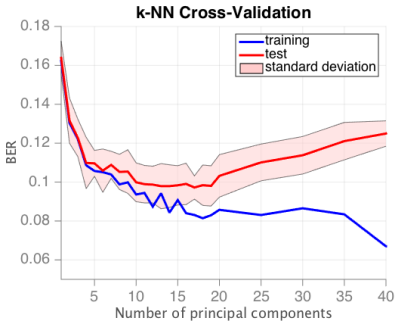
5 Applied Methods and Analysis

5.1 Binary Prediction

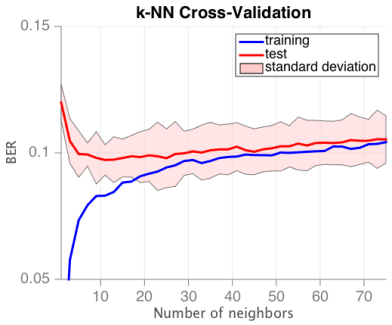
5.1.1 k-Nearest Neighbor

We used k-Nearest Neighbor as a base model to get a general idea of the dataset and an upper bound on the error we should expect. With the built-in MATLAB function function `fitcknn`, a grid search was performed to optimize two parameters of the model: D (the number of principal components) and k (the number of neighbors used to make a decision). The value of k was varied from 1 to 75 with increments of 2 and D was varied as such: from 1 to 20 with increments of 1, from 25 to 50 with increments of 5, and from 60 to 100 with increment of 10. K-fold cross-validation with $K = 5$ was used to find the optimal values for D and k and to get an estimate of the test-error.

Using 5-fold cross-validation, we determined the optimal number of principal components and number of neighbors for k-NN to be $D = 17$ and $k = 11$ respectively. Figure 2(a) and Figure 2(b) show the cross-validation results when adjusting these hyper-parameters. For $D = 17$ and $k = 11$, the test-error was estimated to be: $9.72 \pm 0.6\%$.



(a) Taking the number of neighbors which gave the minimum test error for a corresponding number of principal components.



(b) Taking the number of principal components which gave the minimum test error for a corresponding number of neighbors.

Figure 2: Cross-Validation for k-NN

From Figure 2(a), we can see that $k-$NN as a model for binary prediction has high bias for a small number of principal component (less complex model). As the number of principal components is increased (more complex model), both the training and test error decrease until $D = 17$. After this point, the test error starts to increase. However, the test error is relatively high even when we try increasing the complexity of $k-$NN.

A similar observation on the high bias of $k-$NN can be made from Figure 2(b). In this plot, model complexity increases to the left as we decrease the number of neighbors used to make a decision for a given image. For k less than 11, the test error starts to increase and even *shoot-up* for $k < 5$ as the model over-fits the training data.

According to Andrew Ng [7], a typical property of high bias is a small gap between training and test error and a high training error. This can be observed from our results in Figure 2(a) and Figure 2(b). High bias would imply that we have too simple of a model, which could be fixed by increasing the complexity by trying a larger set of features [7]. However, using a larger number of features with $k-$NN did not improve the test error as shown by Figure 2(a). Therefore, we must use a more complex model than $k-$NN for binary classification. In the next sub-sections, we investigate a Support Vector Machine (SVM) and a Neural Network for to accomplish this task.

5.1.2 Support Vector Machine

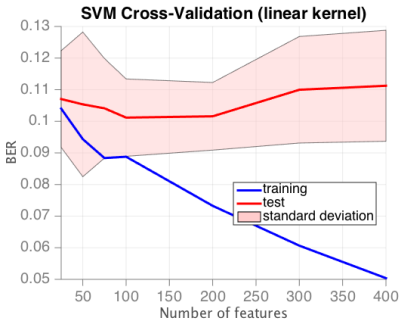
As mentioned in the previous subsection, we will use an SVM for binary classification and prediction as it allows for more complex decision boundaries. As well as linear classification, SVMs can efficiently perform non-linear classification through the use of *kernels*. We will investigate the linear, polynomial (of degree 2), and rbf (radial basis function) kernels with the built-in MATLAB function `fitcsvm`.

There are two hyper-parameters that we will tune using grid search and $5-$fold cross validation: the number of principal components D and the box constraint C. The box constraint is a parameter that controls the maximum penalty imposed on margin-violating observations and can aid in preventing overfitting [5]. Thus, it is a regularization term.

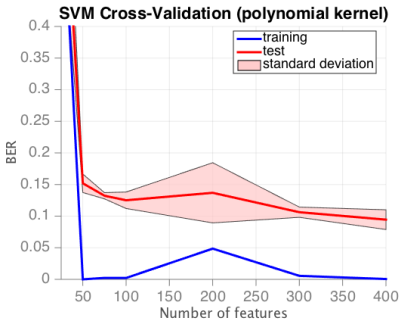
For each kernel, we will use $5-$fold cross validation and vary D from 25 to 100 with steps of 25 and C from 0.5 to 1.5 with steps of 0.5. Below are the results for each kernel. The optimal C and D are chosen such that they minimize the average test-error of $5-$fold cross validation.

Kernel	Optimal D	Optimal C	BER
Linear	100	1.5	$10.12 \pm 1.22\%$
Polynomial	400	0.5	9.44 ± 1.56
RBF	–	–	100%

Table 1: Errors for SVM



(a) For linear kernel.



(b) For polynomial kernel of degree 2.

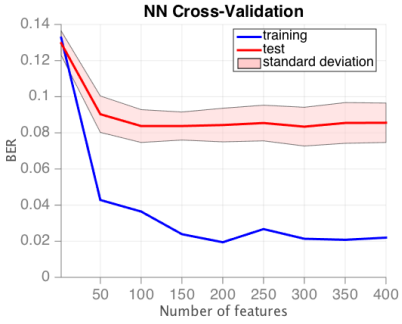
Figure 3: Cross-Validation for SVM

5.1.3 Neural Network

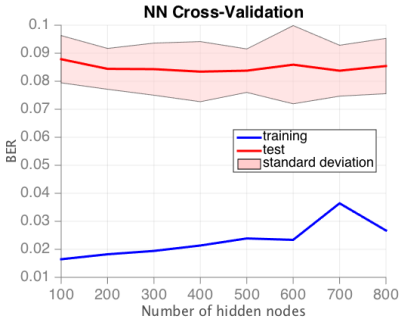
A Neural Network was also investigated for binary prediction in order to compensate for the high bias mentioned in Section 5.1.1. Neural Networks allow very complex decision boundaries. However, they are extremely difficult to tune since they have many hyper-parameters. To reduce computational complexity, we left a couple hyper-parameters fixed: the number of hidden layers to 1 and the learning rate to 5. The work of Cybenko [2] and Hornik et. al [3] in 1989 helped form the *universality theorem*, which generally states that arbitrary decision regions can be well approximated by single hidden layer neural network, provided that there are sufficiently many hidden units available. Therefore, the hyper-parameters that we will adjust are the number of hidden units H in the single hidden layer, the number of principal component features D, the number of epochs E, and the batch size B for mini-batch gradient descent. The MATLAB toolbox *DeepLearnToolbox* is used to train and apply a neural network [8].

Rather than performing an exhaustive grid search for these four parameters, we fix two and perform a grid search over the other two. This is a sub-optimal method for tuning the hyper-parameters but the computational time is much less. With $E = 30$ and $B = 50$, D was varied from 100 to 400 with steps of 50 and H was varied from 200 to 800 with step of 100. Using, $5-$fold cross-validation, the values for D and H which yielded the minimum mean test-error were $D = 300$ and $H = 400$. Consequently, with $D = 300$ and $H = 400$, E was varied from 20 to 40 with steps of 5 and B was varied from 10 to 90 with steps of 20. Using, $5-$fold cross-validation, the values for E and B which yielded the minimum mean test-error were $E = 35$ and $B = 30$.

With $D = 300$, $H = 400$, $E = 35$, and $B = 30$, the test-error is approximated to be $8.37 \pm 0.79\%$.



(a) Taking the number of hidden nodes which gave the minimum test error for a corresponding number of principal components.



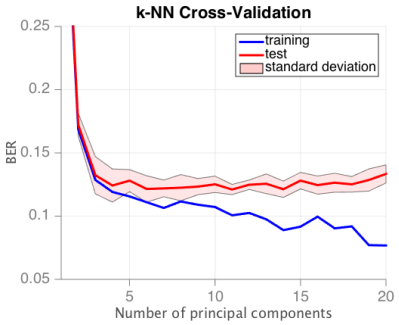
(b) Taking the number of principal components which gave the minimum test error for a corresponding number of hidden nodes.

Figure 4: Cross-Validation for Neural Network

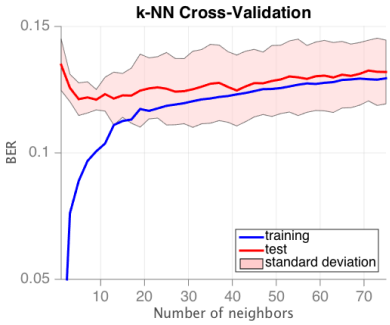
5.2 Multi-Class Prediction

5.2.1 k-Nearest Neighbor

We used k-Nearest Neighbor as a base model to get a general idea of the dataset and an upper bound on the error we should expect. With the built-in MATLAB function function `fitcknn`, a grid search was performed to optimize two parameters of the model: D (the number of principal components) and k (the number of neighbors used to make a decision). The value of D was varied from 1 to 20 and k from 1 to 75 with increments of 2. K-fold cross-validation with $K = 5$ was used to find the optimal values for D and k and to get an estimate of the test-error.



(a) Taking the number of neighbors which gave the minimum test error for a corresponding number of principal components.



(b) Taking the number of principal components which gave the minimum test error for a corresponding number of neighbors.

Figure 5: Cross-Validation for k-NN

Using 5-fold cross-validation, we determined the optimal number of principal components and number of neighbors for k-NN to be $D = 11$ and $k = 5$ respectively. Figure 5(a) and Figure 5(b) show the cross-validation results when adjusting these hyper-parameters. For $D = 11$ and $k = 5$, the test-error was estimated to be: $12.11 \pm 0.4\%$.

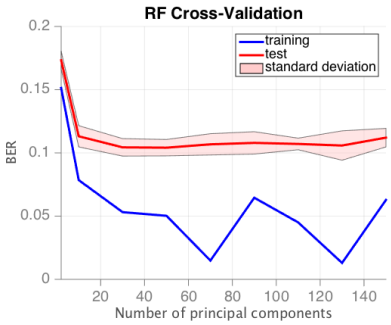
Like the performance of $k-$NN for the binary prediction task in Section 5.1.1, this model exhibits high bias. This can be concluded from Figure 5(b) as there is a small gap between the training and test error and the training error is higher than what is desired. Therefore, our solution to this is the same as in the binary prediction task: we will try more complex models. In the following sub-sections, we investigate Random Forests and a Neural Network in order to achieve better performance for multi-class prediction.

5.2.2 Random Forests

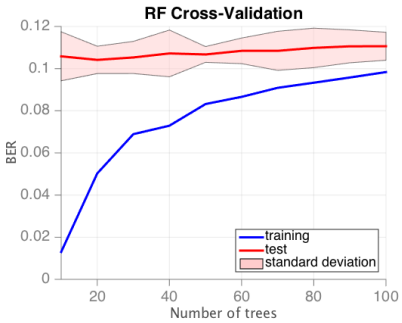
Random Forests are very flexible models that allow for complex decision boundaries while combating over-fitting through averaging. For our Random Forests, we have four parameters to consider: the number principal component features D, the maximum tree depth M, the number of trees N, and the number of features that will be sampled F. According to [1], m_{try} is typically set to $\sqrt{D}$ so we will use this rule of thumb. Also, we fixed M to 256. Higher values of M did not improve the test-error.

Therefore, in order to tune the parameters D and N, we performed a grid search: D was varied from 10 to 150 with increments of 20 and N was varied from 10 to 100 with increments of 10. We also tried values of N up until 500 but the test-error was steadily increasing after 100. K-fold cross-validation with $K = 5$ was used to find the optimal values for D and N and to get an estimate of the test-error.

Using 5-fold cross-validation, we determined the optimal number of principal components and number of trees to be $D = 50$ and $N = 20$ respectively. Figure 6(a) and Figure 6(b) show the cross-validation results for adjusting these hyper-parameters. For $D = 50$ and $N = 20$, the test-error was estimated to be: $10.41 \pm 0.65\%$.



(a) Taking the number of trees which gave the minimum test error for a corresponding number of principal components.



(b) Taking the number of principal components which gave the minimum test error for a corresponding number of trees.

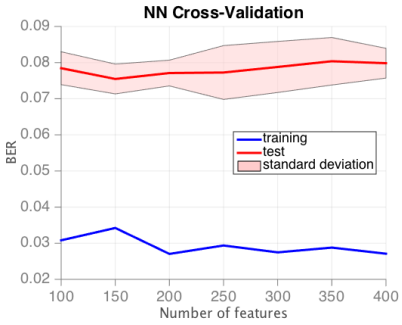
Figure 6: Cross-Validation for Random Forests

5.2.3 Neural Network

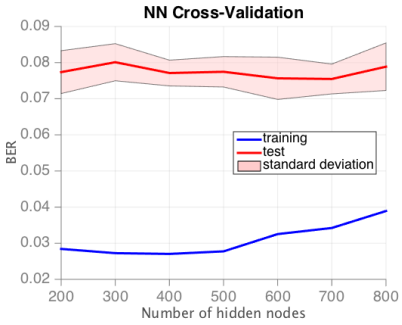
A Neural Network was also investigated for multi-class prediction in order to compensate for the high bias mentioned in Section 5.2.1. As for binary prediction, we left a couple hyper-parameters fixed to reduce the computational cost in tuning the model. The number of hidden layers is fixed to 1 and the learning rate is set to 5. The hyper-parameters that we will adjust are the number of hidden units H in the single hidden layer, the number of principal component features D, the number of epochs E, and the batch size B for mini-batch gradient descent.

Just as in the binary prediction task, we fine-tune two hyper-parameters at a time. With $E = 30$ and $B = 50$, D was varied from 100 to 400 with steps of 50 and H was varied from 200 to 800 with step of 100. Using, $5-$fold cross-validation, the values for D and H which yielded the minimum mean test-error were $D = 150$ and $H = 700$. Consequently, with $D = 150$ and $H = 700$, E was varied from 20 to 40 with steps of 5 and B was varied from 10 to 90 with steps of 20. Using, $5-$fold cross-validation, the values for E and B which yielded the minimum mean test-error were $E = 35$ and $B = 50$.

With $D = 150$, $H = 700$, $E = 35$, and $B = 50$, the test-error is approximated to be $7.71 \pm 0.56\%$.



(a) Taking the number of hidden nodes size which gave the minimum test error for a corresponding number of principal components.



(b) Taking the number of principal components which gave the minimum test error for a corresponding number of hidden nodes.

Figure 7: Cross-Validation for Neural Network

The above cross-validation plots seems to imply that our model exhibits high variance as there is a large gap between our training and test error for increasing complexity and we would like performance somewhere in between.

6 Implementation Details

Our code is separated into a subfolder for each model. Within a particular method's subfolder, there is a file called 'test_[method].m' where the parameters can be adjusted and the model can be run.

In order to make computation fast, we pre-computed the test and training data for 5-fold cross-validation in order to avoid repeatedly computing PCA. This implementation is found in the PCA subfolder. This should be done before running our code.

7 Conclusion

In summary, we first tackled this task of image classification with PCA. It was necessary to reduce the dimensionality of the input data in order to reduce the computational complexity and to prevent over-fitting. Consequently, for binary prediction we investigated several models such as $k-$Nearest Neighbor, Support Vector Machines with different kernels, and a single hidden layer Neural Network. For multi-class prediction, we investigated $k-$Nearest Neighbor, Random Forests, and a single hidden layer Neural Network. We found that the single hidden layer Neural Network performed best for both binary and multi-class prediction.

For binary prediction, we used a sub-optimal grid search to tune our parameters to the following values: number of principal components $D = 300$, number of hidden units in the single hidden layer $H = 400$, number of epochs $E = 35$, and the batch size $B = 30$. With these parameters, we predict the test-error to be approximately $8.37 \pm 0.79\%$.

For multi-class prediction, we used also a sub-optimal grid search to tune our parameter to the following values: number of principal components $D = 150$, number of hidden units in the single hidden layer $H = 700$, number of epochs $E = 35$, and the batch size $B = 50$. With these parameters, we predict the test-error to be approximately $7.71 \pm 0.56\%$.

The cross-validation plots for both binary and multi-class prediction show that our models have high variance as there is a large gap between the training and test error. Future work to remedy this issue could be to gather more training examples. According to Andrew Ng [7], this is a typical solution to fixing high variance. It would be also interesting to experiment with multiple hidden layers. Moreover, SVMs with kernels of higher polynomial degrees may offer a better alternative to neural networks for binary classification.

Acknowledgments

We would like to thank Emti and the teacher assistants for providing useful advice. The office hours and lab sessions were very helpful. The course notes and lecture slides from [4] provided the necessary guidance for this project.

References

[1] Carlos Becker. Random forests. http://icapeople.epfl.ch/mekhan/pcml15/random_forests.pdf.

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[3] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[4] Mohammad Emtiyaz Khan. Pattern classification and machine learning lecture notes http://icapeople.epfl.ch/mekhan/pcml15.html.

[5] MathWorks. fitcsvm http://ch.mathworks.com/help/stats/fitcsvm.html.

[6] MathWorks. svds http://ch.mathworks.com/help/matlab/ref/svds.html?refresh=true.

[7] Andrew Ng. Advice for applying machine learning. http://cs229.stanford.edu/materials/ML-advice.pdf.

[8] R. B. Palm. Prediction as a candidate for learning deep hierarchical models of data.