Project 2 Report

Jingyang Guo, Yifan Wang, Anthony Colas, FNU Ronald Wilson

**1.Results**

*1.1 Accuracies*

|  |  |  |
| --- | --- | --- |
| RVM | SVM | GPR |
| 93% | 95.22% | 96.22% |

　\*best results using 5-fold cross-validation

*1.2 Overall confusion matrices*

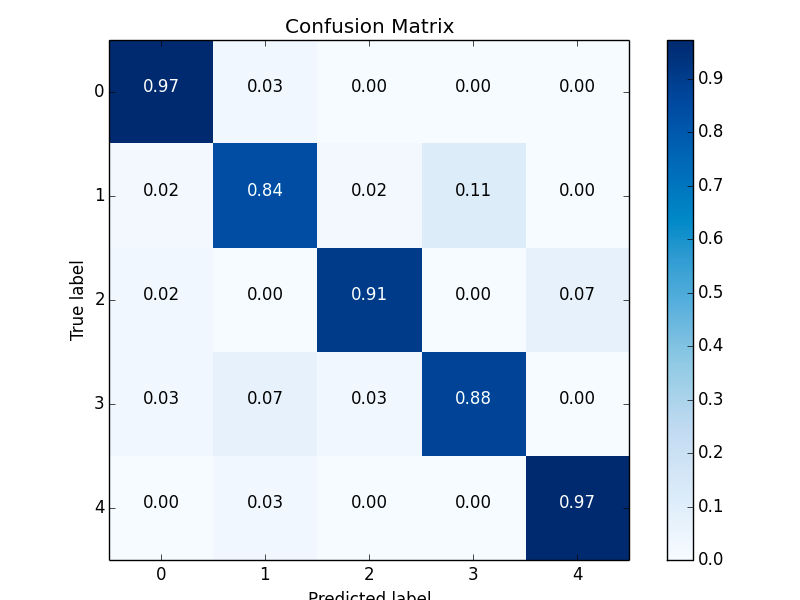


Figure 1: RVM Confusion Matrix

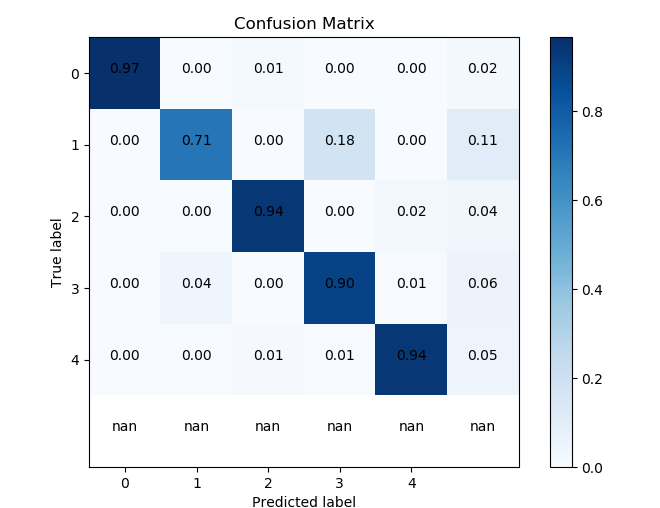


Figure 2: SVM Confusion Matrix

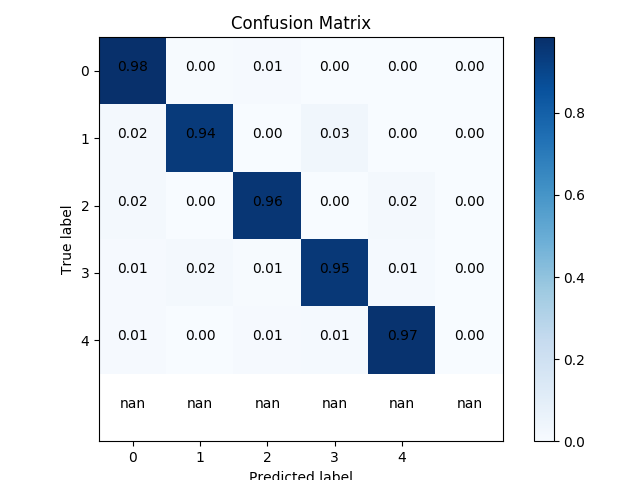
**

Figure 3: GPR Confusion Matrix

*1.3 Number of vectors*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
| RVM | 800 | 800 | 800 | 800 | 800 |
| SVM | 1029 | 1039 | 1017 | 1020 | 1025 |

**2. Observations**

*2.1 Relevance Vector Machines (RVM)*

RVM calculates Hessian requiring O(N^3) computation ,thereby, slowing down the training phase. Therefore, the amount of data was randomly downsampled from 25,000 samples to 2,000 samples. RVMs provide probabilistic predictions and can be easily extended to multiple-classes but their accuracy may be affected by downsampling. We are also training using a reduced number of iterations compared to SVM and GPR. Hence, the degradation in performance.

*2.2 Support Vector Machines (SVM)*

There are 2 factor affecting performance. If gamma is large, there will be more support vectors so that it will be easy to overfit. The second is the number of training iterations. In our experiments, SVM converges fast. Around 100 to 300 iterations are enough. If we iterate too many times, there will be overfitting. Performance of SVM will be excellent without overfitting. Accuracy of each validation fold is over 94% and quite close to each other, which means not only does SVM have good accuracy but also great stability.

*2.3 Gaussian Process Regression (GPR)*

We chose scikit-learn library as our toolkit to implement this algorithm. What confused us at first is that the Gaussian Process Classifier in sklearn does not support returning probabilities when using one\_vs\_one (i.e. all-pairs) method, so we had to train a classifier using one\_vs\_rest method for each pair of classes in our function “MyGPC”, and work out the one\_vs\_one (all-pairs) results manually according to the method mentioned in project 2 instruction. Secondly, about the probability that one doesn’t belong to any of those classes, intuitively, the closer those probabilities are, the more likely one belongs to a new class. So we believe it should be related with the deviation of the probabilities of these classes. To introduce a reasonable new probability value, let standard deviation of the probabilities that one belongs to each of current classes, denote it with std\_dev, then the new probability should be p = exp(-std\_dev). The reason why we choose this function is that it satisfies such fact: when std\_dev = 0, p = 1, making the probability that this sample comes from an unknown class pretty high; when std\_dev -> ∞, p = 0. This is a reasonable property. However std\_dev tends to be tiny, making p somehow too high, so we multiplied it by #Classes \* 10, because intuitively the more known classes, the less likely a sample comes from an unknown class.