Midterm report

Jin Tack Lim (jl4312) March 24, 2017

1. Data analysis

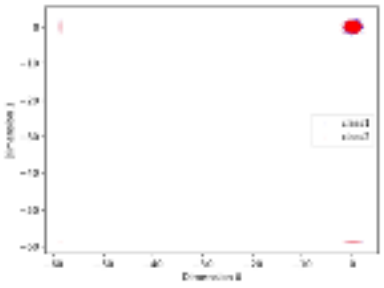


Figure 1. Pair-wise diagram

Each input data vector has either -9.332 or some value x where $|x| < 1$.

Average number of occurrence of -9.322 in a row is 10 out of 100.

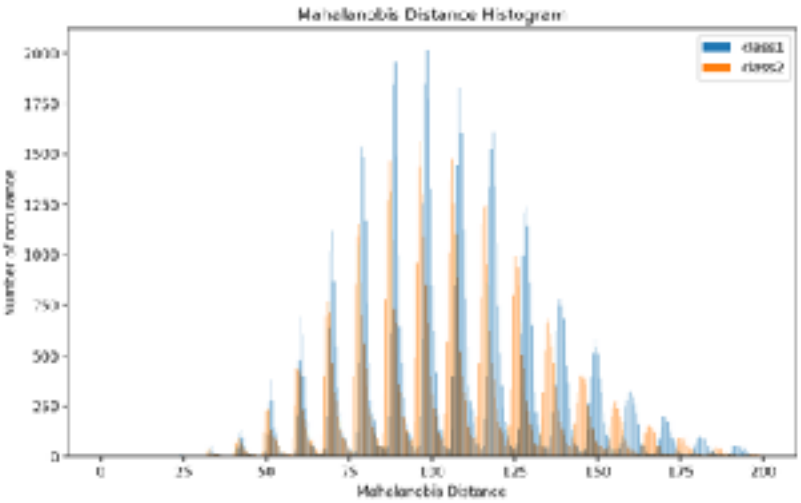
So, Figure 1 shows typical 2D plot of any selection of two dimensions. As shown in the left figure, most of points are around (0,0) and there are few points which have -9.332 either x or y or both regardless of class. The figure on the right shows points around (0,0).

Command: python3 train.py -t

Note: need to put Data_[xy].pkl to others/StudentData.

2. Mahalanobis Distance

Mahalanobis distance gives some more information. I got the Mahalanobis distance using MinCovDet[1] for each class. Figure 2 shows the histogram of Mahalanobis distance for every data points. I got much higher distances for small number of points, but those are not shown in this graph due to the page limitation.



The first take away is that the underlying distribution looks like based on Sinusoidal Basis Functions or Radial Basis Functions. I thought running SVM with rbf would give reasonable result, but it didn't.

The second takeaway, which I took more seriously, is that two different classes showed slightly different patterns. In other words, as the distance is increased, the two histograms get out of sync, so we may be able to predict a class of new data by looking at the Mahalanobis distance. So this is my methodology.

1) Get the covariance from the training data for each class.
2) Get the distance distribution using the covariance for each class. (Figure above). At this point, training is done.
3) For testing, use the covariance and compute two Mahalanobis distance for each new data point
4) Compare the probability of being each class using the computed the Mahalanobis distance and its underlying distribution for each class.

I implemented this idea, even though step 4) is a bit simplified due to time limitation. The evaluation result was no better than 0.5. Due to time constraint, I couldn't investigate further, but one thing I observed at the last moment is that the pattern of mahalanobis distance is not the same even though I use the same data. (See others/Mahalanobis-dist.png) . It also happens when using EllipticEnvelope as well.

Command for getting diagram: python3 train.py -m

Command for training a classifier: python3 train.py -e

3. Outlier

As briefly described, there are points whose distance is way larger than most of the points. (e.g. average distance is roughly around 120 but they have around 9000). About 10% of total data fall into this category. We can define them as outlier. In fact, excluding those points from the training phase resulted in better performance when using AdaBoost. Score is improved from 0.50 to 0.53. However the shortcoming of using mahalanobis distance is that it takes long time to get it.

A quicker way to handle outlier is to use RobustScaler[2]. It worked fine in a sense that using default params for RobustScaler transformation was fast and also gave slightly better performance with Adaboost, similar to using mahalanobis distance.

4. PCA & Feature selection

I used PCA to reduce dimensions, but it turned out not necessary for our dataset. The reason is, the given dataset does not have a dominant direction that the data is stretched, but all 100 directions are more or less the same. This is top three and the bottom three the result of explained_variance_ratio_.

```
[0.01064309  0.01062621  0.01058187 … 0.00943618  0.00940585  0.00936699]
```

They are all close to 0.01, not like other examples where they show dramatic difference between values[4]. Applying PCA didn't change the classifier performance as well, so I decided not to do PCA.

Command: python3 train.py -p

5. Final Classifier

I have used BernoulliNB, GaussianNB, SGDClassifier, DecisionTree, LogisticRegression, SVM with linear, rbf and poly kernel. With default parameters, none of them gave score better than 0.5. SVM with rbf gave 1.0 training performance, but the test result was also about 0.5. So it was obviously overfit.

With the outlier handling with RobustScaler, Adaboost gave the best performance, 0.57 with some more iterations than the default setup. I did the cross variation using ShuffleSplit(), and the performance was very stable.

Command: python3 train.py

5. Other things I tried

I transformed the input data to binary by mapping -9.993 to 1 and other numbers to 0. I thought it would be a good setting to use BernoulliNB or GaussianNB but didn't have any impact. I tried to run np.cov() to observe any correlation between dimensions but calling that function resulted in out-of-memory.

References

[1] MinCovDet: http://scikit-learn.org/stable/auto_examples/covariance/plot_mahalanobis_distances.html#sphx-glr-auto-examples-covariance-plot-mahalanobis-distances-py

[2] RobustScaler: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

[3] EllipticEnvelope: http://scikit-learn.org/stable/auto_examples/applications/plot_outlier_detection_housing.html#sphx-glr-auto-examples-applications-plot-outlier-detection-housing-py

[4] http://scikit-learn.org/stable/auto_examples/plot_digits_pipe.html#sphx-glr-auto-examples-plot-digits-pipe-py

[5] ShuffleSplit: http://scikit-learn.org/stable/modules/cross_validation.html

[6] EllipticEnvelope : http://scikit-learn.org/stable/modules/generated/sklearn.covariance.EllipticEnvelope.html#sklearn.covariance.EllipticEnvelope