Home Assignment 5

Yu Wang (ndp689)

January 11, 2022

Contents

1 VC dimension (38 points) [Mathias]

1. From the definition of $d_{VC}(\mathcal{H})$ we know that:

$$d_{VC}(\mathcal{H}) = \max\{n | m_{\mathcal{H}}(n) = 2^n\}$$

And, we have $m_{\mathcal{H}}(n) \leq min\{M, 2^n\}$ (from Home Assignment 4) .
Situation 1: If $M \leq 2^n$, $d_{VC}(\mathcal{H}) \leq log_2 M$.
Situation 2: If $M > 2^n$, $d_{VC}(\mathcal{H}) \leq n$, where $n < log_2 M$.
Then, combining the two situations, we have:

$$d_{VC}(\mathcal{H}) \leq log_2 M$$

2. From Point 1, we know $d_{VC}(\mathcal{H}) \leq log_2 M$. So, when $|\mathcal{H}| = M = 2$, we have $d_{VC}(\mathcal{H}) \leq 1$.
Besides, if there is one point, $\mathcal{H}$ can shatter it. Since $|\mathcal{H}| = 2$, we can get 2 results for the point. Then, $d_{VC}(\mathcal{H}) \geq 1$.
So, $d_{VC}(\mathcal{H}) = 1$

3. A bound on the loss that is less than 1 is non-trivial, so:

$$\begin{aligned}
\hat{L}(h,S) + \sqrt{\frac{8\ln\left(2\left((2n)^{d_{VC}}+1\right)/\delta\right)}{n}} < 1 \iff \\
\sqrt{\frac{8\ln\left(2\left((2n)^{d_{VC}}+1\right)/\delta\right)}{n}} < 1 \\
8\ln\left(\frac{2\left((2n)^{d_{VC}}+1\right)}{\delta}\right) < n \\
\ln\left(\frac{2\left((2n)^{d_{VC}}+1\right)}{\delta}\right) < \frac{n}{8}
\end{aligned}$$

Then, we get the relation between $d_{VC}(\mathcal{H})$ and n:

$$(2n)^{d_{VC}} < \frac{e^{\frac{n}{8}}\delta}{2} - 1$$

4. The generalization bound we obtain with Theorem 3.16 is from Theorem 3.8:

$$P\left(\exists h \in \mathcal{H} : L(h) \geq \hat{L}(h,S) + \sqrt{\frac{8\ln\frac{2m_{\mathcal{H}}(2n)}{\delta}}{n}}\right) \leq \delta \qquad (1)$$

and Theorem:

$$m_{\mathcal{H}}(n) \leq \sum_{i=0}^{d_{VC}(\mathcal{H})} \binom{n}{i} \leq n^{d_{VC}(\mathcal{H})} + 1 \tag{2}$$

For proving equation (1), we use Hoeffding bound and union bound. And for proving the generalization bound in Theorem 3.2, we also use Hoeffding bound and union bound.

Using equation (2), we get a looser generalization bound in Theorem 3.16, compared with equation (1). Then, the generalization bound in Theorem 3.2 is tighter.

5.

6. In Figure 1, there are 3 points (3 red points) and 8 positive circles$\in \mathcal{H}_+$ colored 8 different colors, which means $\mathcal{H}_+$ can scatter 3 points.
   Then, $d_{VC}(\mathcal{H}_+) \geq 3$

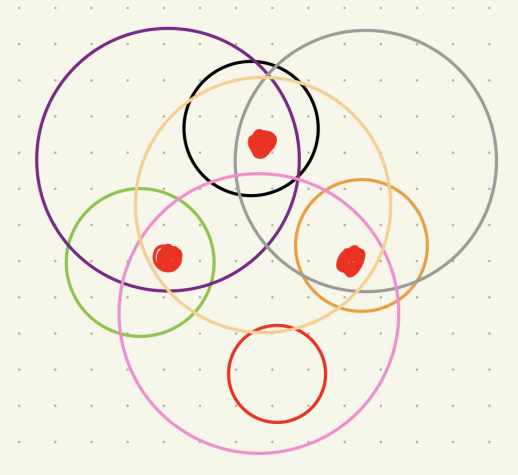

Figure 1:

7. From Point 6, we can easily know that $d_{VC}(\mathcal{H}_-) \geq 3$. Since $d_{VC}(\mathcal{H}) = d_{VC}(\mathcal{H}_+) \cup d_{VC}(\mathcal{H}_-)$. Then, $m_{\mathcal{H}}(n) = m_{\mathcal{H}_+}(n) + m_{\mathcal{H}_-}(n) \geq 2^3 + 2^3 = 2^4$. So, $d_{VC}(\mathcal{H}) \geq 4$.

8. optional

9. For the hypothesis space $\mathcal{H}_d$ of binary decision trees of depth d, we can get $|\mathcal{H}_d| = 2^{2^d}$. Thus, 2^d points can be shattered by $\mathcal{H}_d$. So,

$d_{VC}(\mathcal{H}) \geq 2^d$, and from Point 1 we know $d_{VC}(\mathcal{H}) \leq log_2(|\mathcal{H}|) = 2^d$. Then, $d_{VC}(\mathcal{H}) = 2^d$

10. For the hypothesis space $\mathcal{H}$ of binary decision trees of unlimited depth, we know that $m_{\mathcal{H}}(n) = 2^n$ for all n. So, $d_{VC}(\mathcal{H}) = \infty$

11.

12.

2 Random Forests (38 points)

2.1 Analyzing Satellite Data

1. The validation accuracy is: 74.86%

   Code snippets for building the model and computing validation accuracy:

```
# train the random forest model
rf = RandomForestClassifier(n_estimators=10, bootstrap=True, criterion='gini',max_depth=None)
rf.fit(train_features,train_label)

validation_accuracy = rf.score(validation_features, validation_label)
print("the validation accuracy is {}%".format(validation_accuracy * 100))
```

Figure 2: Code snippets

2. Code snippets for prediction and visualization:

```
prediction = rf.predict(area_data)
prediction = np.reshape(prediction, (3000,3000))

plt.imshow(prediction)
```

Figure 3: Code snippets

   Prediction result and the corresponding area see Figure 4 and 5 (the next page)

3. In the worst case, the maximum depth of a decision tree is n.

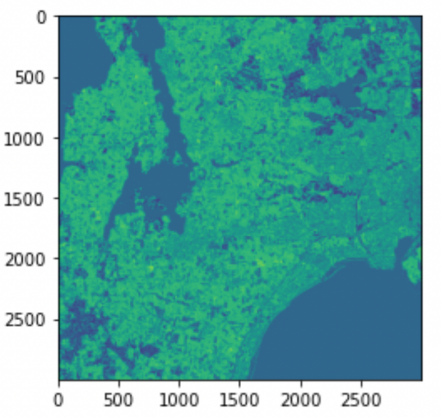

Figure 4: prediction

Figure 5: the corresponding area

2.2 Normalization

- Is nearest neighbor classification affected by this type of normalization?

  Yes!

  Example: For the nearest neighbor classification problem, if there are two features for input data: $f1 \in [100, 5000]$, and $f2 \in [10, 20]$. Then, $f1$ can contribute more weight when compute the distances between training data and test data. If we use such normalization, $f1$ and $f2$ may have the same level contribution when compute the distances. So, it will be affected.

- Is random forest classification affected by this normalization?

  No. Because the predictions of the random forest classification are based on impurity measure (misclassification error or gini index) rather than the distance. So, it will not be affected.

3 Principal Component Analysis

3.1 PCA and preprocessing

3.1.1 by the book

(b) If one dimension (say x1) is inflated disproportionately, x1 will be the main direction, and the 'natural axis' will be close to x1.

We should not perform input normalization before doing PCA. Because

the variance of every dimension will be 1 after using scaling, and then, there is no use doing PCA.

(c) If we do input whitening, every direction can be the 'natural axes'. Since every direction will be on an equal footing after whitening.

3.1.2 In the center

Assume a $d \times d$ matrix $\mathbf{A}$, consisting of $d \times d$ 1s.

Let $\mathbf{S'}$ be the matrix centering from the matrix $\mathbf{S}$. Let $\bar{\mathbf{s}}_\mathbf{i}$ and $\bar{\mathbf{a}}_\mathbf{i}$ be the column vector of $\mathbf{S'}$ and $\mathbf{A}$, respectively.

Then, for any $i \in [1, d]$ we have:

$$\frac{\bar{\mathbf{s}}_\mathbf{i}^\mathbf{T} \bar{\mathbf{a}}_\mathbf{i}}{d} = \mathbf{0} \iff \bar{\mathbf{s}}_\mathbf{i}^\mathbf{T} \bar{\mathbf{a}}_\mathbf{i} = \mathbf{0}$$

Thus,

$$\mathbf{S'^T A} = \mathbf{0} \tag{3}$$

And we know that if the product of two matrix is the null matrix, which means $\mathbf{M}_{e \times f} \mathbf{N}_{f \times g} = \mathbf{0}$, we have $\text{Rank}(\mathbf{M}) + \text{Rank}(\mathbf{N}) \leq f$.

From equation (3), we have $\text{Rank}(\mathbf{S'^T}) + \text{Rank}(\mathbf{A}) = \text{Rank}(\mathbf{S'}) + \text{Rank}(\mathbf{A})$ $\leq d$. And we have $\text{Rank}(\mathbf{A}) = 1$. So, $\text{Rank}(\mathbf{S'}) \leq d - 1$.

Then, if we center the matrix, then the rank of the resulting matrix is at most d1

3.2 PCA in practice

The code of this part I added to the notebook:

Do the PCA:

```
pca = PCA()
pca.fit(X)
```

✓ 0.3s

```
PCA()
```

Find out if 10 components are enough to explain 80% of the variance:

```
explained_variance_per_component = pca.explained_variance_ / np.sum(pca.explained_variance_)
quotient_10_components = np.sum(explained_variance_per_component[:10])
print('10 components can explain {}% of the variance'.format(quotient_10_components * 100))
```

✓ 0.2s

```
10 components can explain 73.82267688459532% of the variance
```

Plot the eigenspectrum:

```
components = np.arange(1, 65, 1)
plt.plot(components, explained_variance_per_component, label="explained variance per component")
plt.xlabel("number of eigenvector")
plt.ylabel("explained variance")
plt.title("Eigenspectrum")
plt.savefig('Eigenspectrum.png')
plt.show()
```

Figure 6: Code for the Explained variance

Plot the explained variance:

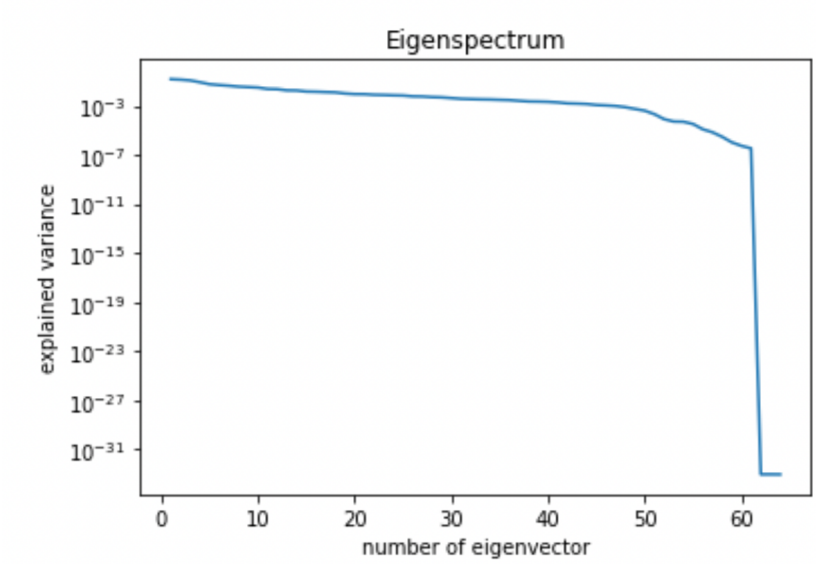

Figure 7: Eigenspectrum

Question: Are 10 principal components enough to explain80% of the variance?

No.

Simply summing the explained variances of the first ten eigenvalues, we can know that 10 components can only explain about 73.82% of the variance.

3.3 Eigendigits

Plot the first five "eigendigits":

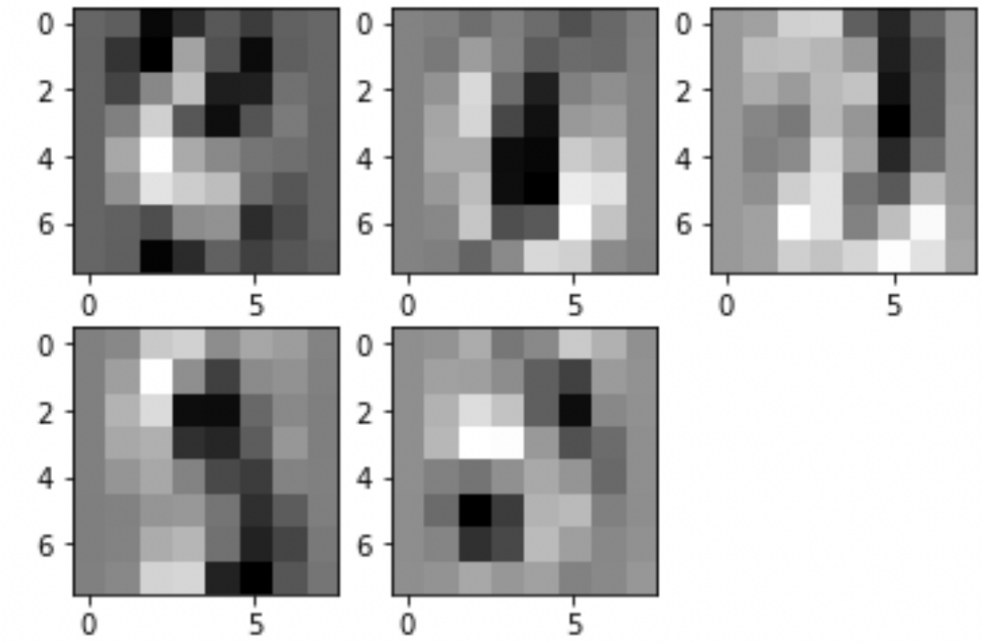

Figure 8: The first five "eigendigits"

4 Variable Stars

4.1 Data understanding and preprocessing

Frequency of classes in original:

class	frequency
0.0	0.088197
22.0	0.106355
17.0	0.033722
11.0	0.027237
24.0	0.041505
16.0	0.029831
23.0	0.088197
4.0	0.022049
10.0	0.011673
8.0	0.035019
21.0	0.009079
3.0	0.124514
6.0	0.077821
14.0	0.027237
9.0	0.075227
5.0	0.064851
13.0	0.011673
7.0	0.012970
1.0	0.028534
2.0	0.001297
15.0	0.033722
12.0	0.024643
19.0	0.012970
18.0	0.007782
20.0	0.003891

Figure 9: Frequency of classes in original

1. Report which classes and how many training and test examples remain.
   Remain classes: 0.0, 3.0, 22.0, 23.0
   314 training examples remain and 335 test examples remain.
   Code snippets for the data removal:

```
# count_class: a dict (mapping from a class to the number of data points belonging to the class)
for key in count_class:
    if count_class[key] < 65:
        # index of classes will be deleted
        index_of_train_class = (train_label == key)
        # delete corresponding examples
        train_label = np.delete(train_label, index_of_train_class)
        train_inputs = np.delete(train_inputs, index_of_train_class, axis=0)

        # the same operations for test data
        index_of_test_class = (test_label == key)
        test_label = np.delete(test_label, index_of_test_class)
        test_inputs = np.delete(test_inputs, index_of_test_class, axis=0)

print("remain classes: " + str(set(train_label)))
print("{} training examples remain".format(train_inputs.shape[0]))
print("{} test examples remain".format(test_inputs.shape[0]))
```

Figure 10: Code snippets for the data removal

2. Code snippets for the normalization:

```
# compute the mean and standard deviation of the training features
mean_of_training = np.mean(train_inputs, axis=0)
standard_of_training = np.std(train_inputs, axis=0)
# to normalize training data
norm_train_inputs = (train_inputs - mean_of_training) / standard_of_training
```

Figure 11: Code snippets for the normalization

4.2 Principal component analysis

Scatter plot of the data projected on the first two principal components:

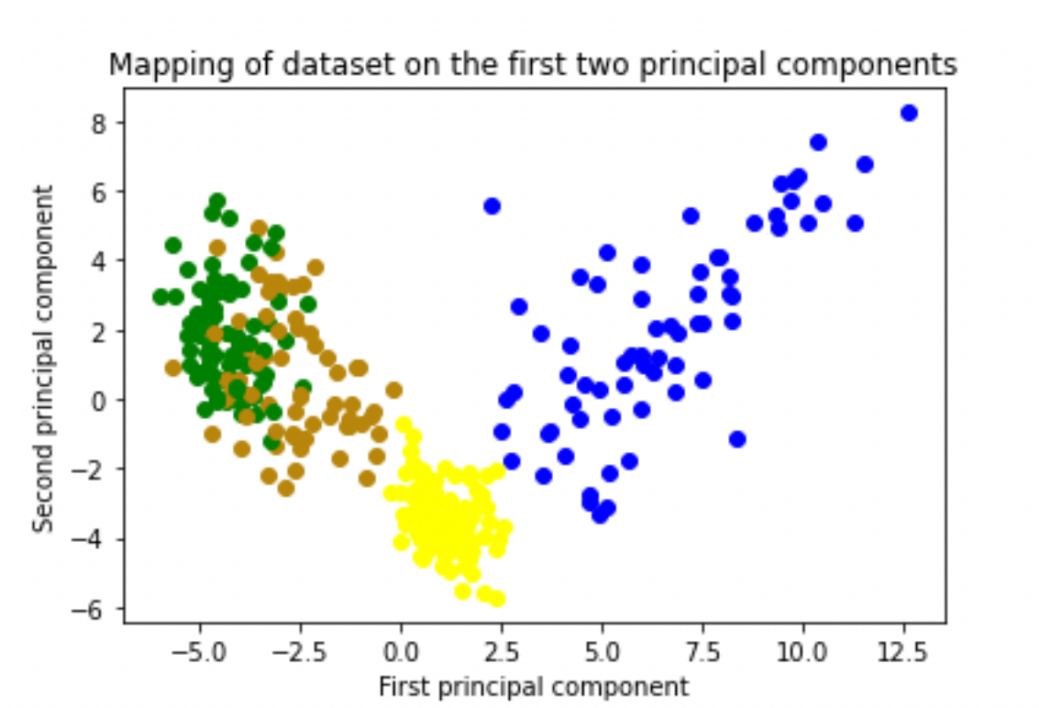

Figure 12: The data projected on the first two principal components

4.3 Clustering

Description of software used: I use the 'sklearn.cluster' library to get the KMeans method, use the 'sklearn.decomposition' library to get the PCA method, and use the 'matplotlib.pyplot' library to plot. Code snippets for using k-means and projecting the cluster centers:

```
kmeans = KMeans(n_clusters = 4, init = 'k-means++')
kmeans.fit(norm_train_inputs)
# project the cluster centers to the first two principal components(pca_1_2)
center = np.dot(kmeans.cluster_centers_, pca_1_2.T)
```

Figure 13: Code snippets for using k-means and projecting the cluster centers

Plot with cluster centers and data points:

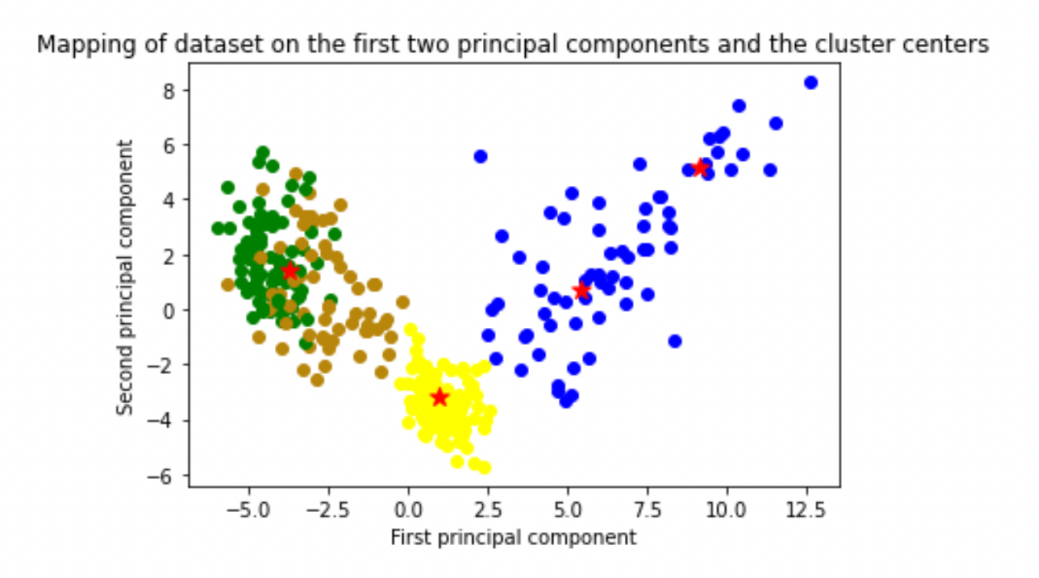

Figure 14: The data projected on the first two principal components and the cluster centers

Short discussion of results: The red stars in the Figure are the four cluster centers. The result is not entirely correct. Because the cluster center should be the center of data points of the same class, which means on the left side (consisting of green points and brown points), there should be two cluster centers, and on the right side (consisting of blue points), there should be one cluster center.