Mathematical Foundations of Machine Learning. Homework 4

Teresa Morales

November 11th, 2019

1. Projection matrix. Let V be a basis for the plane in R^3 given by $\{x \in R^3$: $x_1 - x_2 - 2x_3 = 0\}$.

   (a) Find the projection matrix P onto V:
   First, we find an orthonormal basis for the plane:

$$V = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{3} \\ 0 & 1/\sqrt{3} \end{bmatrix}$$

$$P = VV^T$$

$$P = \begin{bmatrix} 5/6 & 1/6 & 1/3 \\ 1/6 & 5/6 & -1/3 \\ 1/3 & -1/3 & 1/3 \end{bmatrix}$$

   (b) What is the rank of P?
   The rank of P is the same as the rank of V, that is to say 2.
   Columns of P are linearly dependent. In particular, we can express one of the columns as a linear combination of the other two:
   x1 = 2x3 + x1

   (c) Calculate the distance of a vector $x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ to V using projection matrix.

$$D = \| x - Px \|_2$$
$$= \| (I - P)x \|_2$$
$$(I - P)x = \begin{bmatrix} 1/6 & -1/6 & -1/3 \\ -1/6 & 1/6 & 1/3 \\ -1/3 & 1/3 & 2/3 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1/3 \\ 1/3 \\ 2/3 \end{bmatrix}$$
$$D = \sqrt{1/9 + 1/9 + 4/9} = \sqrt{2/3}$$

2. The SVD. Imagine there are three points in R^2, where $x_1 = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$, $x_2 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$, $x_3 = \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix}$

   Let data matrix be $X = = \begin{bmatrix} x_1, x_2, x_3 \end{bmatrix}$

   (a) Find an orthonormal basis for the span of the three points. The span of the three points is the whole bidimensional space. This means that a possible orthonormal basis could be the following:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

   (b) For any vector v in R^2, we can regress all points onto v, i.e., project all data points onto the subspace V spanned by v.

      i. What is the projection matrix P onto V ?
      v is uni-dimensional, which means that the projection matrix is the following:

$$\frac{vv^T}{v^Tv}$$

      ii. What is the squared distance of each data point from subspace V ?

$$\| x_i - \frac{vv^T}{v^Tv} x_i \|_2^2$$

$$= x_i^T x_i - x_i^T \frac{vv^T}{v^Tv} x_i = x_i^T x_i - \frac{v^T(x_i x_i^T)v}{v^Tv}$$

$$x_1^T x_1 = 9$$

$$x_1 x_1^T = \begin{bmatrix} 9 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\| x_1 - \frac{vv^T}{v^Tv} x_1 \|_2^2 = \frac{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} 9 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}}{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}} = \frac{9 - 9v_1^2}{v_1^2 + v_2^2}$$

$$x_2^T x_2 = 9 + 16 = 25$$

$$x_2 x_2^T = \begin{bmatrix} 9 & 12 \\ 12 & 16 \end{bmatrix}$$

$$\| x_2 - \frac{vv^T}{v^T v} x_2 \|_2^2 = \frac{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} 9 & 12 \\ 12 & 16 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}}{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}} = \frac{25 - 9v_1^2 + 24v_1v_2 + 16v_2^2}{v_1^2 + v_2^2}$$

$$x_3^T x_3 = 2$$

$$x_3 x_3^T = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\| x_3 - \frac{vv^T}{v^T v} x_3 \|_2^2 = \frac{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}}{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}} = \frac{2 - 2v_2^2}{v_1^2 + v_2^2}$$

iii. Now find v with $\| v \|_2 = 1$ that minimizes sum of the squared distance for all data points. Is v unique? Is P unique?
We want to find $v = argmin \sum_{i=1}^{3} x_i^T x_i - x_i^T \frac{vv^T}{v^T v} x_i$ which is equivalent to $v = argmax \sum_{i=1}^{3} x_i^T \frac{vv^T}{v^T v} x_i = \frac{v^T \sum_{i=1}^{3} (x_i x^T) v}{v^T v}$

$$max \frac{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} 18 & 12 \\ 12 & 18 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}}{\begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}}$$

$$= \frac{18v_1^2 + 24v_1v_2 + 18v_2^2}{v_1^2 + v_2^2}$$

We set the gradient equal to 0

$$\frac{(36v_1 + 24v_2)(v_1^2 + v_2^2) - 2v_1(18v_1^2 + 24v_1v_2 + 18v_2^2}{(v_1^2 + v_2^2)^2} = 0$$

$$\frac{(36v_2 + 24v_1)(v_1^2 + v_2^2) - 2v_2(18v_1^2 + 24v_1v_2 + 18v_2^2}{(v_1^2 + v_2^2)^2} = 0$$

And add the restriction $v_1^2 + v_2^2 = 1$
Simplifying we obtain the following:
$v_2(1 - 2v_1^2) = 0$, $v_1(1 - 2v_2^2) = 0$ and $v_1 = v_2$
There are two possible solutions for v: $v = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ and $v = \begin{bmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$

However, these two vectors represent the same subspace (one is the negative of the other).
P is unique, and is equal to:

$$P = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$$

(c) Suppose now we are doing full SVD of X, i.e., $X = U\Sigma V^T$. Please provide possible matrices U and Σ? Hint: use the results of part (b).
By definition, we know that $u_1 = v = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$
Since we are working in the bidimensional space, we know that $u_2 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ since it has to be orthogonal to u_1 and normal.
Given u_1 we can also calculate σ_1:

$$\sigma_1^2 = max \frac{u_1^T \sum_{i=1}^{3} (x_i x^T) u_1}{u_1^T u_1} = 30$$

$$\sigma_1 = \sqrt{30}$$

Similarly,

$$\sigma_2^2 = max \frac{u_2^T \sum_{i=1}^{3} (x_i x^T) u_2}{u_2^T u_2} = 6$$

$$\sigma_2 = \sqrt{6}$$

All in all, we have that

$$U = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sqrt{30} & 0 & 0 \\ 0 & \sqrt{6} & 0 \end{bmatrix}$$

3. The informative SVD. Let $X \in R^{np}$ with $rank(X) = k < minn, p$. Each row represents one training sample, and each column represents one feature. Let $y \in R^n$ be the response vector for each sample. The full SVD of X is given by $X = U\Sigma V^T$, where

$$U = \begin{bmatrix} u_1, ..., u_k, ..., u_n \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma_1 & . & . & 0 & 0 & 0 \\ . & \sigma_2 & . & 0 & 0 & 0 \\ . & . & \sigma_k & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} v_1^T \\ \\ . \\ . \\ . \\ v_k^T \\ . \\ . \\ . \\ v_p^T \end{bmatrix}$$

(a) Decompose X to be a sum of rank one matrices. One way to decompose X as a sum of rank one matrices would be to use the outer product representation of the SVD:

$$X = \sum_{i=1}^{n} \sigma_i u_i v_i^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

since $\sigma_i = 0$ for $i > k$ where σ_i are scalars, u_i are columns of U and v_i^T are rows of V.

(b) Think of each row of X as a point in R^p. Based on the full SVD, find a basis for the best 1d subspace (line through origin) fit to these n data points.

The best 1d-subspace for these n data points would be the first column of V, that is to say v_1.

(c) What are the SVDs of X^T, XX^T, and X^TX?

$$X^T = V\Sigma^T U^T$$
$$XX^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$$
$$X^TX = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T\Sigma V^T$$

(d) Now we focus on least square problem, and try to find weight vector w by minimizing $\| yXw \|_2^2$.

i. Find a basis for all weight vectors that satisfy $Xw = 0$. (If $Xw = 0$, then for any $\hat{w}$ such that $y = X\hat{w}$,we have $y = X(\hat{w} + w)$)
We know that the first k columns of V (rows of V^T) form a basis for the rows of X and that colums v_{k+1} to columns v_p are orthogonal to the former and span the null(X).
This means that a possible basis for all weight vectors that satisfy $Xw = 0$ is:

$$\{v_{k+1}, ...v_p\}$$

ii. Prove the least square problem is equivalent to minimizing $\| \tilde{y} - \Sigma\tilde{w} \|_2^2$, for $\tilde{y} = U^T y$ and $\tilde{w} = V^T w$, and find the optimal $\tilde{w}$ with the smallest norm. First we plug in the values:

$$\| U^T y - \Sigma V^T w \|_2^2$$

$$(U^T y - \Sigma V^T w)^T (U^T y - \Sigma V^T w)$$

$$(y^T U - w^T V \Sigma^T)^T (U^T y - \Sigma V^T w)$$

$$y^T U U^T y - y^T U \Sigma V^T w - w^T V \Sigma^T U^T y + w^T V \Sigma^T \Sigma V^T w$$

$$y^T y - y^T X w - w^T X^T y + w^T X^T X w$$

$$(y - Xw)^T (y - Xw)$$

$$\| y - Xw \|_2^2$$

which is equivalent to the least square problem.
In order to find the $\tilde{w}$ with the smallest norm we can use the expressions above as follows:

$$\tilde{w} = argmin(\| \tilde{y} - \Sigma\tilde{w} \|_2^2)$$

$$= argmin(y^T U U^T y - y^T U \Sigma V^T w - w^T V \Sigma^T U^T y + w^T V \Sigma^T \Sigma V^T w)$$

We set the gradient equal to 0:

$$\nabla f = -y^T U \Sigma V^T - V \Sigma^T U^T y + 2V \Sigma^T \Sigma V w$$

$$V \Sigma^T \Sigma V w = V \Sigma^T U^T y$$

$$\Sigma^T \Sigma V w = V^T V \Sigma^T U^T y$$

$$V w = (\Sigma^T \Sigma)^{-1} \Sigma^T U^T y$$

$$w* = V \Sigma^\dagger U^T y$$

This is the least square solution, which we know has the smallest norm when there are infinitely many solutions.
Since both problems are equivalent, we know that

$$\tilde{w}* = V^T w* = V^T V \Sigma^\dagger U^T y = \Sigma^\dagger U^T y$$

will also have the smallest norm.

iii. Find the optimal solution to the original least squares problem with the smallest norm.
From the above reasoning we also know that:

$$w* = V\Sigma^{\dagger}U^{T}y$$

is the optimal solution to the least squares problem with the smallest norm.

4. Recall the iris flower classification problem from Homework 3. Packages used in this problem are:

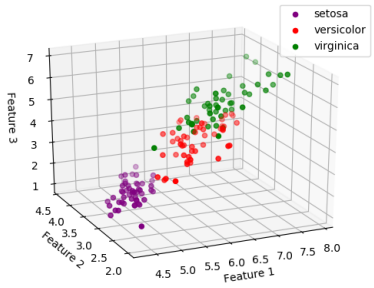
```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import math
import random
import pandas as pd
import scipy.io as sio
import matplotlib.pyplot as plt
```

(a) Use a 3d scatter plot to visualize the measurements in part (d) from HW3. Can you find a 2-dimensional subspace that the data approximately lie in? You can do this by rotating the plot and looking for plane that approximately contains the data points.
The code to create the graph is the following:

```
file_path = '/Users/teresamorales/Documents/Harris/MFML/
    Homework3/fisheriris.mat'
fisher_data = sio.loadmat(file_path)
Species = fisher_data['species']
X = fisher_data['meas']

X_sel = X[:, 0:3]

labels = ('setosa', 'versicolor', 'virginica')
colors = ('purple', 'red', 'green')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_sel[0:50, 0], X_sel[0:50, 1], X_sel[0:50, 2],
     c=colors[0], label=labels[0])
ax.scatter(X_sel[50:100, 0], X_sel[50:100, 1], X_sel
    [50:100, 2], c=colors[1], label=labels[1])
ax.scatter(X_sel[100:150, 0], X_sel[100:150, 1], X_sel
    [100:150, 2], c=colors[2], label=labels[2])
ax.set_xlabel('Feature 1', fontsize=10)
ax.set_ylabel('Feature 2', fontsize=10)
ax.set_zlabel('Feature 3', fontsize=10)
ax.legend()
plt.show()
```

Figure one shows a three dimensional representation of observations, given three of their measurements (one per axis) and their classification through colors:

Figure 1:



By inspection, we define a plane going through the origin (a subspace) that fits these points, whose basis can be defined by two vectors:

$$B = \{ \begin{bmatrix} 5 & 5 \\ 3 & 0 \\ 0 & 3 \end{bmatrix} \}$$

(b) Use this subspace to find a 2-dimensional classification rule. What is the average test error in this case?

In order to find a 2 dimensional classification rule, we first project every point in our best 2 dimensional subspace fit to the data.

```
def project(X, P):
    P_t = np.transpose(P)
    Proj = np.matmul(P, np.matmul(np.linalg.inv(
                    np.matmul(P_t, P)), P_t))
    X_p = np.matmul(Proj, X)
    return X_p


F_p = np.array([project(X[i, 0:3], P) for i in range(150)
    ])
```

Since observations now lie on a plane, the three features are linearly dependent and we can use only 2 features. Our features matrix will therefore be:

```
Fp_2d = F_p[:, 0:2]
```

We now repeat the process we followed in homework 3 to calculate error rates for different training sizes:

```
def create_dummies(name_species, n):
    y = np.array([1 if Species[i, 0] == name_species else
    -1 for i in range(n)])
```

```
    return y


def calc_weights(X, y):
    Xt = np.transpose(X)
    XtX = np.matmul(Xt, X)
    Inverse_XtX = np.linalg.inv(XtX)
    w = np.matmul(np.matmul(Inverse_XtX, Xt), y)
    return w


def divide_randomly(sample_size, training_size, seed):
    sample_size_index = list(range(sample_size))
    random.Random(seed).shuffle(sample_size_index)
    return [sample_size_index[0:training_size],
    sample_size_index[training_size:]]


def sign(num):
    return -1 if num < 0 else 1


def calc_error_rate(splited_index, features_matrix,
    labels_vector):
    X_reserved = features_matrix[splited_index[1]]
    y_reserved = labels_vector[splited_index[1]]
    X_training = features_matrix[splited_index[0]]
    y_training = labels_vector[splited_index[0]]
    w_training = calc_weights(X_training, y_training)
    y_hat = np.matmul(X_reserved, w_training)
    y_tilda = [sign(i) for i in y_hat]
    return np.sum([y_tilda[i] != y_reserved[i] for i in
    range(len(splited_index[1]))])/len(splited_index[1])


#Create dummies for each category

y_setosa = create_dummies('setosa', 150)
y_versicolor = create_dummies('versicolor', 150)
y_virginica = create_dummies('virginica', 150)

#Define training samples and repetitions:

n_repetitions = 50
training_size = [120, 115, 110, 105, 100, 95, 90, 85, 80,
    75, 70, 65, 60, 55, 50, 45, 40, 35, 30, 25, 20, 15,
    10, 5, 3]

# Error rates with 2 dimensions:
error_rates_setosa_2d = {i: [] for i in training_size}
error_rates_versicolor_2d = {i: [] for i in training_size}
error_rates_virginica_2d = {i: [] for i in training_size}

for size in training_size:
    for i in range(n_repetitions):
        index = divide_randomly(150, size, seed=i)
        error_rates_setosa_2d[size].append(calc_error_rate
```

```
    (index, Fp_2d, y_setosa))
55          error_rates_versicolor_2d[size].append(
    calc_error_rate(index, Fp_2d, y_versicolor))
56          error_rates_virginica_2d[size].append(
    calc_error_rate(index, Fp_2d, y_virginica))
57
58 mean_error_setosa_2d = {i: np.mean(error_rates_setosa_2d[i
    ]) for i in training_size}
59 mean_error_versicolor_2d = {i: np.mean(
    error_rates_versicolor_2d[i]) for i in training_size}
60 mean_error_virginica_2d = {i: np.mean(
    error_rates_virginica_2d[i]) for i in training_size}
```

In order to compare results with the ones we obtained in homework 3, we replicate this analysis with trhee dimensions and create a graph to compare results:

```
1  # Error rates with 3 dimensions:
2
3  error_rates_setosa_sel = {i: [] for i in training_size}
4  error_rates_versicolor_sel = {i: [] for i in training_size
       }
5  error_rates_virginica_sel = {i: [] for i in training_size}
6
7  for size in training_size:
8      for i in range(n_repetitions):
9          index = divide_randomly(150, size, seed=i)
10         error_rates_setosa_sel[size].append(
       calc_error_rate(index, X_sel, y_setosa))
11         error_rates_versicolor_sel[size].append(
       calc_error_rate(index, X_sel, y_versicolor))
12         error_rates_virginica_sel[size].append(
       calc_error_rate(index, X_sel, y_virginica))
13
14 mean_error_setosa_sel = {i: np.mean(error_rates_setosa_sel
       [i]) for i in training_size}
15 mean_error_versicolor_sel = {i: np.mean(
       error_rates_versicolor_sel[i]) for i in training_size}
16 mean_error_virginica_sel = {i: np.mean(
       error_rates_virginica_sel[i]) for i in training_size}
17
18 # Making a graph
19
20 setosa = {'training_size': list(mean_error_setosa_sel.keys
       ()),
21           'mean_error_setosa_sel': list(
       mean_error_setosa_sel.values())}
22 flours_df = pd.DataFrame(data=setosa)
23 flours_df['mean_error_versicolor_sel'] = list(
       mean_error_versicolor_sel.values())
24 flours_df['mean_error_virginica_sel'] = list(
       mean_error_virginica_sel.values())
25 flours_df['mean_error_setosa_2d'] = list(
       mean_error_setosa_2d.values())
26 flours_df['mean_error_versicolor_2d'] = list(
       mean_error_versicolor_2d.values())
```

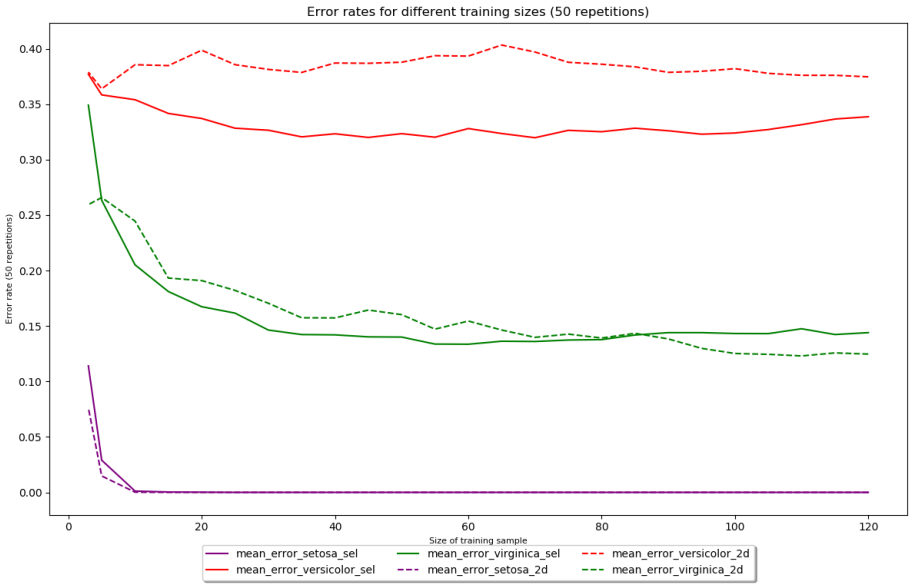
```
flours_df['mean_error_virginica_2d'] = list(
    mean_error_virginica_2d.values())

fig, ax = plt.subplots(figsize=(12, 6))
plt.plot('training_size', 'mean_error_setosa_sel', data=
    flours_df, color='purple')
plt.plot('training_size', 'mean_error_versicolor_sel',
    data=flours_df, color='red')
plt.plot('training_size', 'mean_error_virginica_sel', data
    =flours_df, color='green')
plt.plot('training_size', 'mean_error_setosa_2d', data=
    flours_df, color='purple', linestyle='--')
plt.plot('training_size', 'mean_error_versicolor_2d', data
    =flours_df, color='red', linestyle='--')
plt.plot('training_size', 'mean_error_virginica_2d', data=
    flours_df, color='green', linestyle='--')

plt.show()
```

As we can see in the graphs and summary tables displayed below, our error rate when we use a 2 dimensional projection of the points seems to be higher for some of the models/training sizes and lower for others. Both for Setosa and Virginica classifications, loosing part of the information through these projections does not seem to have a huge impact in terms of prediction errors. For predicting whether a flower is Versicolor or not, the impact is higher (it is worth saying that the error for this model was already the highest in the first place).

For a training size of 120, error rates stay at 0 for Setosa, go from 0.14 to 0.12 for Virginica and from 0.33 to 0.37 for Versicolor.

```
pd.set_option('display.max_columns', None)
flours_df.loc[[0, 4, 12, 18, 24], ['training_size',
...                                    '
    mean_error_setosa_sel',
...                                    '
    mean_error_setosa_2d']]

training_size  mean_error_setosa_sel  mean_error_setosa_2d
   120               0.000000               0.00000
   100               0.000000               0.00000
    60               0.000000               0.00000
    30               0.000000               0.00000
     3               0.114014               0.07619
flours_df.loc[[0, 4, 12, 18, 24], ['training_size',
...                                    '
    mean_error_virginica_sel',
...                                    '
    mean_error_virginica_2d']]
training_size  mean_error_virginica_sel
    mean_error_virginica_2d
   120                  0.144000                 0.124667
   100                  0.143200                 0.125200
    60                  0.133556                 0.154444
```

Figure 2:




```
    30                  0.146333                  0.170500
     3                  0.349116                  0.259456
flours_df.loc[[0, 4, 12, 18, 24], ['training_size',
...                                              '
    mean_error_versicolor_sel',
...                                              '
    mean_error_versicolor_2d']]
training_size  mean_error_versicolor_sel
    mean_error_versicolor_2d
 120                  0.338667                  0.374667
 100                  0.324000                  0.382000
  60                  0.328000                  0.393333
  30                  0.326500                  0.381333
   3                  0.377007                  0.378912
```