```
# tells us that the first 60,000 rows constitute the training set.
X_train, X_test = X.iloc[:60000], X.iloc[60000:]
y_train, y_test = y.iloc[:60000].astype(int), y.iloc[60000:].astype(int)
```

- What do `X_train` and `y_train` actually look like?

In [4]:
```
X_train
```

Out[4]:

	pixel1	pixel2	pixel3	pixel4	...	pixel781	pixel782	pixel783	pixel784
0	0	0	0	0	...	0	0	0	0
1	0	0	0	0	...	0	0	0	0
2	0	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
59997	0	0	0	0	...	0	0	0	0
59998	0	0	0	0	...	0	0	0	0
59999	0	0	0	0	...	0	0	0	0

60000 rows × 784 columns

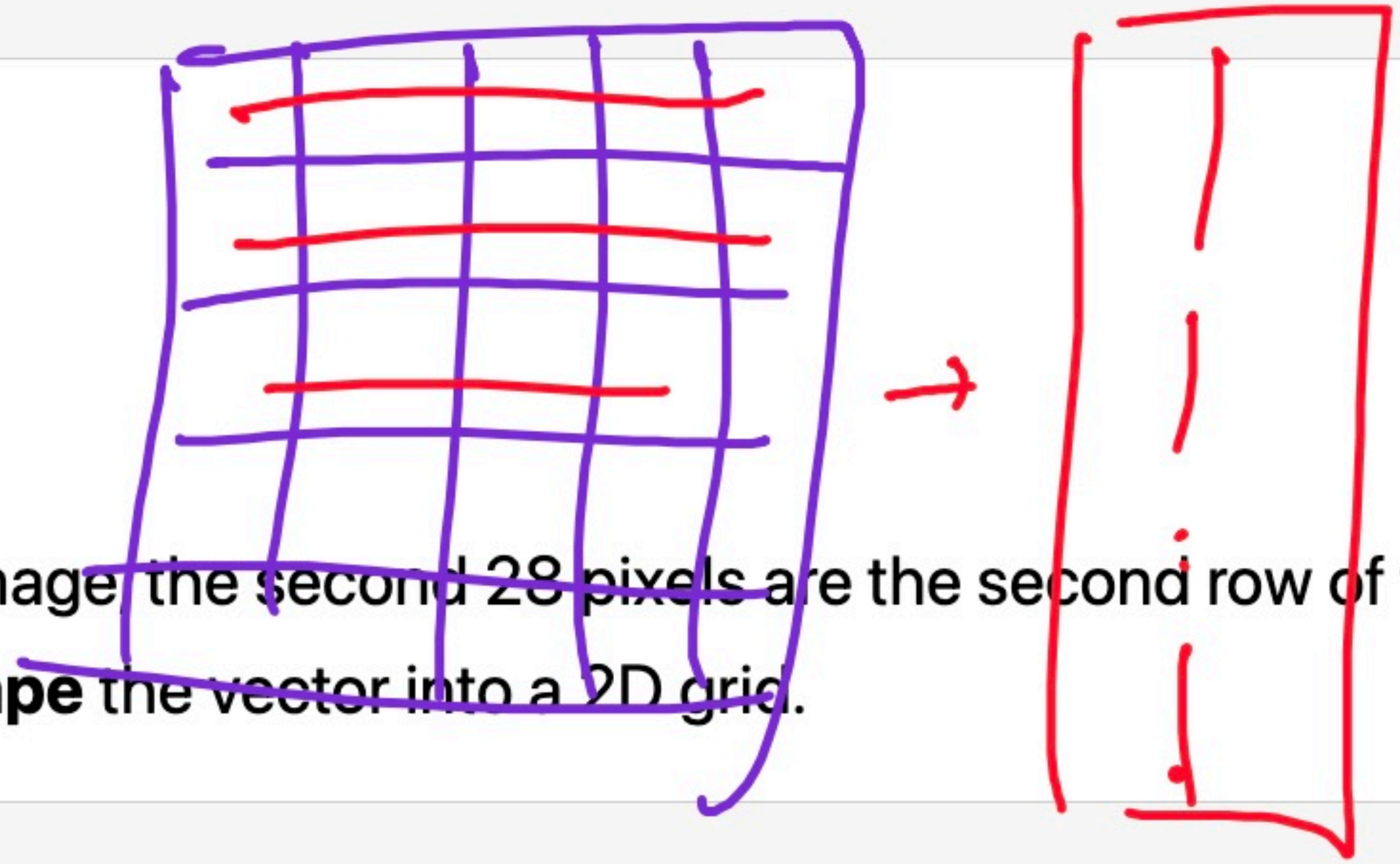
28×28

In [5]:
```
y_train
```

Out[5]:
```
0        5
1        0
2        4
        ..
59997    5
59998    6
59999    8
Name: class, Length: 60000, dtype: int64
```

```
In [6]: X_train.iloc[98]
```

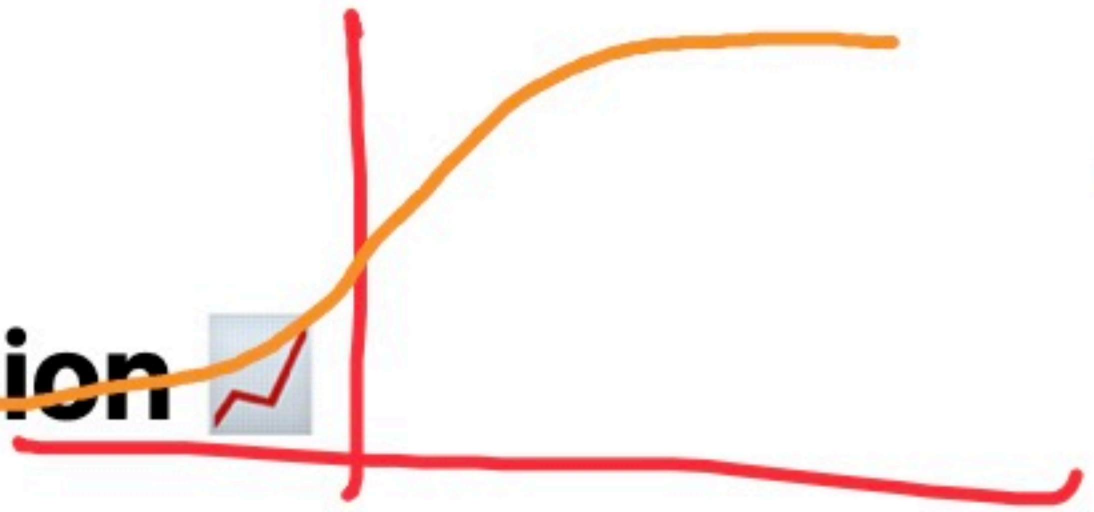
```
Out[6]: pixel1      0
        pixel2      0
        pixel3      0
                   ..
        pixel782    0
        pixel783    0
        pixel784    0
        Name: 98, Length: 784, dtype: int64
```



- The first 28 pixels are the first **row** of the image, the second 28 pixels are the second row of the image, and so on. To view the image, we can **reshape** the vector into a 2D grid.

```
In [7]: X_train.iloc[98].to_numpy().reshape((28, 28))
```

```
Out[7]: array([[0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               ...,
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0]])
```

logistic function:  $\sigma(x) = \dfrac{1}{1+e^{-x}}$

Model #2: Multinomial logistic regression 📈

- Multinomial logistic regression, or softmax regression, predicts the probability that an image $\vec{x}_i \in \mathbb{R}^{784}$ belongs to each class.

$$P(\text{image } \vec{x}_i \text{ is of digit } j) = P(y_i = j | \vec{x}_i) = \frac{e^{\vec{w}_j \cdot \text{Aug}(\vec{x}_i)}}{\sum_{k=0}^{9} e^{\vec{w}_k \cdot \text{Aug}(\vec{x}_i)}}$$

Here, j could be 0, 1, 2, ..., 9.

softmax function

$$\vec{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_d \end{bmatrix}$$

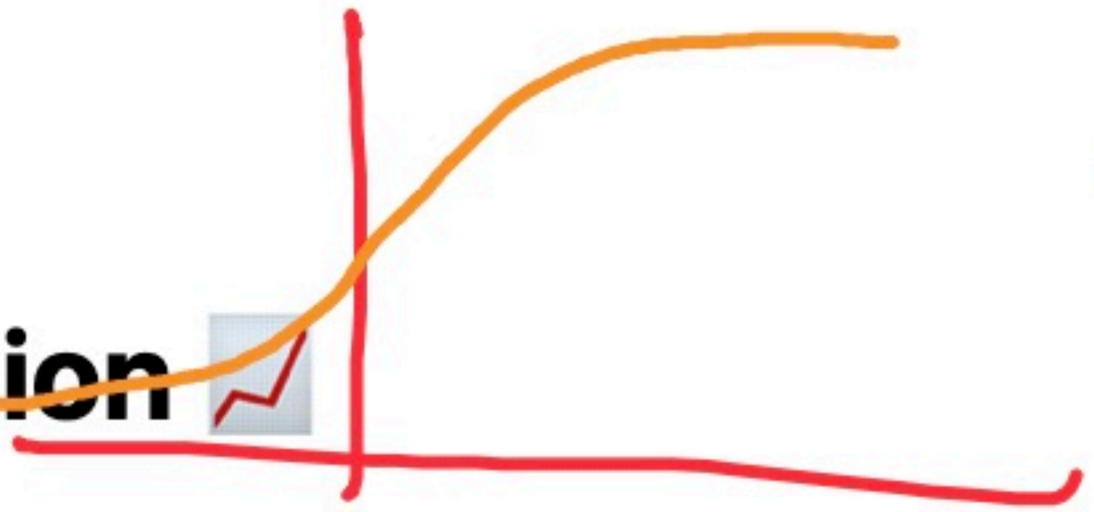
component i in the output is

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}$$

e.g. $\vec{z} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

$$\sigma(\vec{z}) = \begin{bmatrix} \dfrac{e^1}{e^1+e^2+e^3} \\ \dfrac{e^2}{e^1+e^2+e^3} \\ \vdots \end{bmatrix}$$

localhost

logistic function:  $\sigma(x) = \dfrac{1}{1+e^{-x}}$

Model #2: Multinomial logistic regression 📈

- Multinomial logistic regression, or softmax regression, predicts the probability that an image $\vec{x}_i \in \mathbb{R}^{784}$ belongs to each class.

$$P(\text{image } \vec{x}_i \text{ is of digit } j) = P(y_i = j | \vec{x}_i) = \frac{e^{\vec{w}_j \cdot \text{Aug}(\vec{x}_i)}}{\sum_{k=0}^{9} e^{\vec{w}_k \cdot \text{Aug}(\vec{x}_i)}}$$

Here, j could be 0, 1, 2, ..., 9.

softmax function

$$\vec{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_d \end{bmatrix}$$

component i in the output is

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}$$

normalizing

e.g. $\vec{z} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

$$\sigma(\vec{z}) = \begin{bmatrix} \dfrac{e^1}{e^1+e^2+e^3} \\ \dfrac{e^2}{e^1+e^2+e^3} \\ \vdots \end{bmatrix}$$

16 . 1

Model #2: Multinomial logistic regression 📈

- Multinomial logistic regression, or softmax regression, predicts the probability that an image $\vec{x}_i \in \mathbb{R}^{784}$ belongs to each class.

$$P(\text{image } \vec{x}_i \text{ is of digit } j) = P(y_i = j | \vec{x}_i) = \frac{e^{\vec{w}_j \cdot \text{Aug}(\vec{x}_i)}}{\sum_{k=0}^{9} e^{\vec{w}_k \cdot \text{Aug}(\vec{x}_i)}}$$

Here, j could be 0, 1, 2, ..., 9.

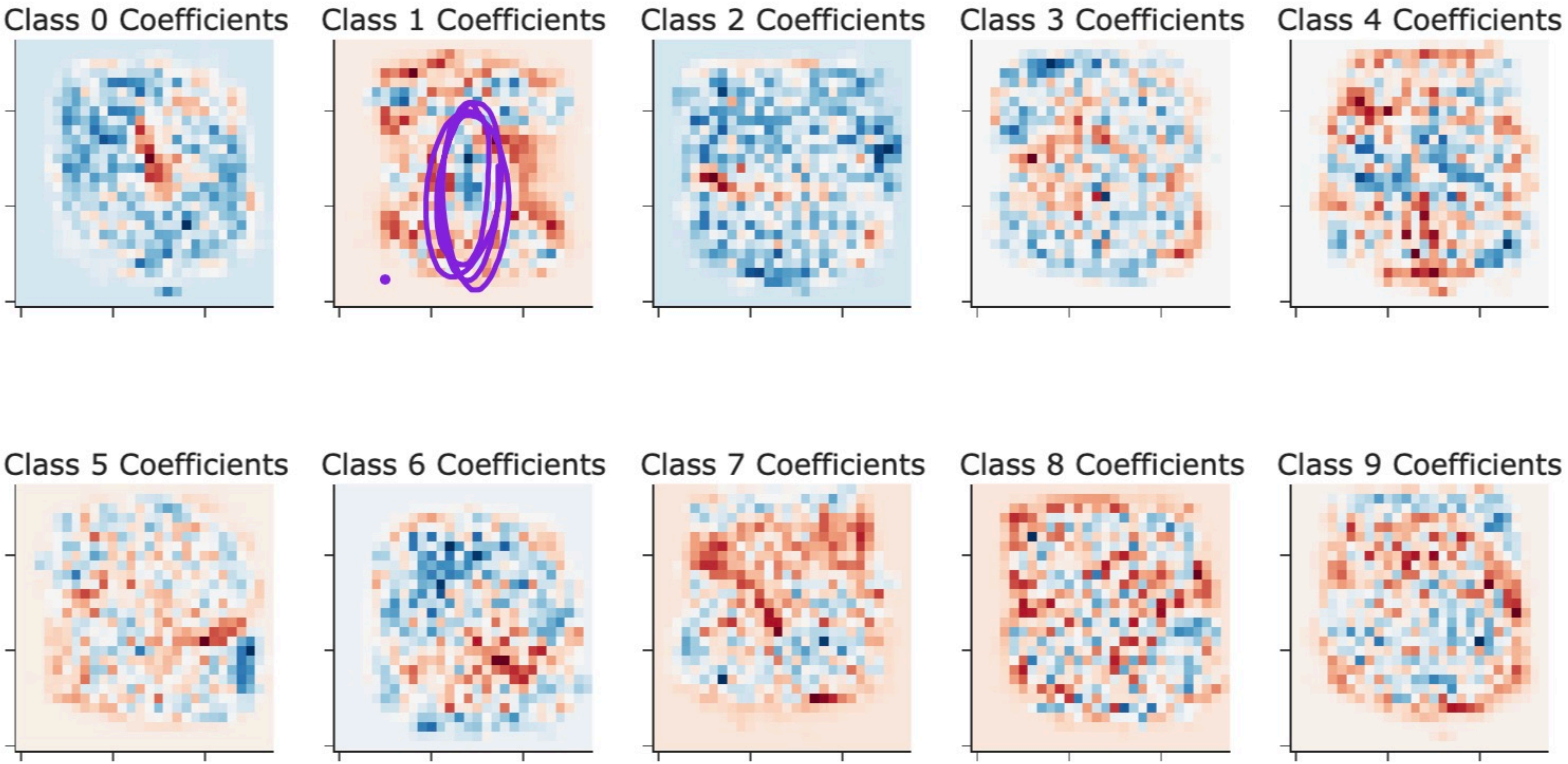
$\vec{w}_0^*$

$\vec{w}_3^*$ ___ parameter vector we use to find $P(y_i = 3 | \vec{x}_i)$.

16.1

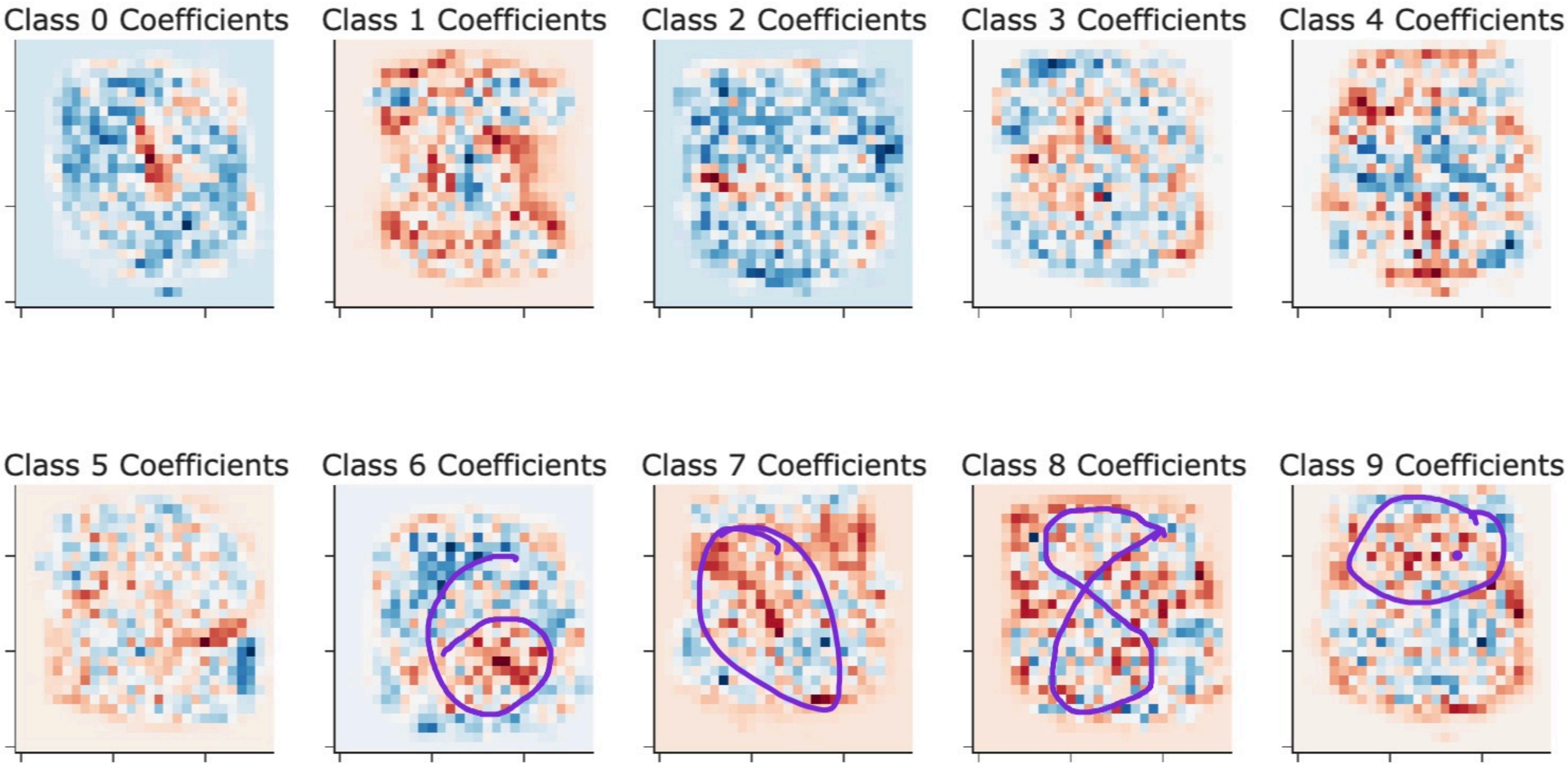
[87]:

```
1 util.plot_model_coefficients(model_log.coef_)
```



[87]:

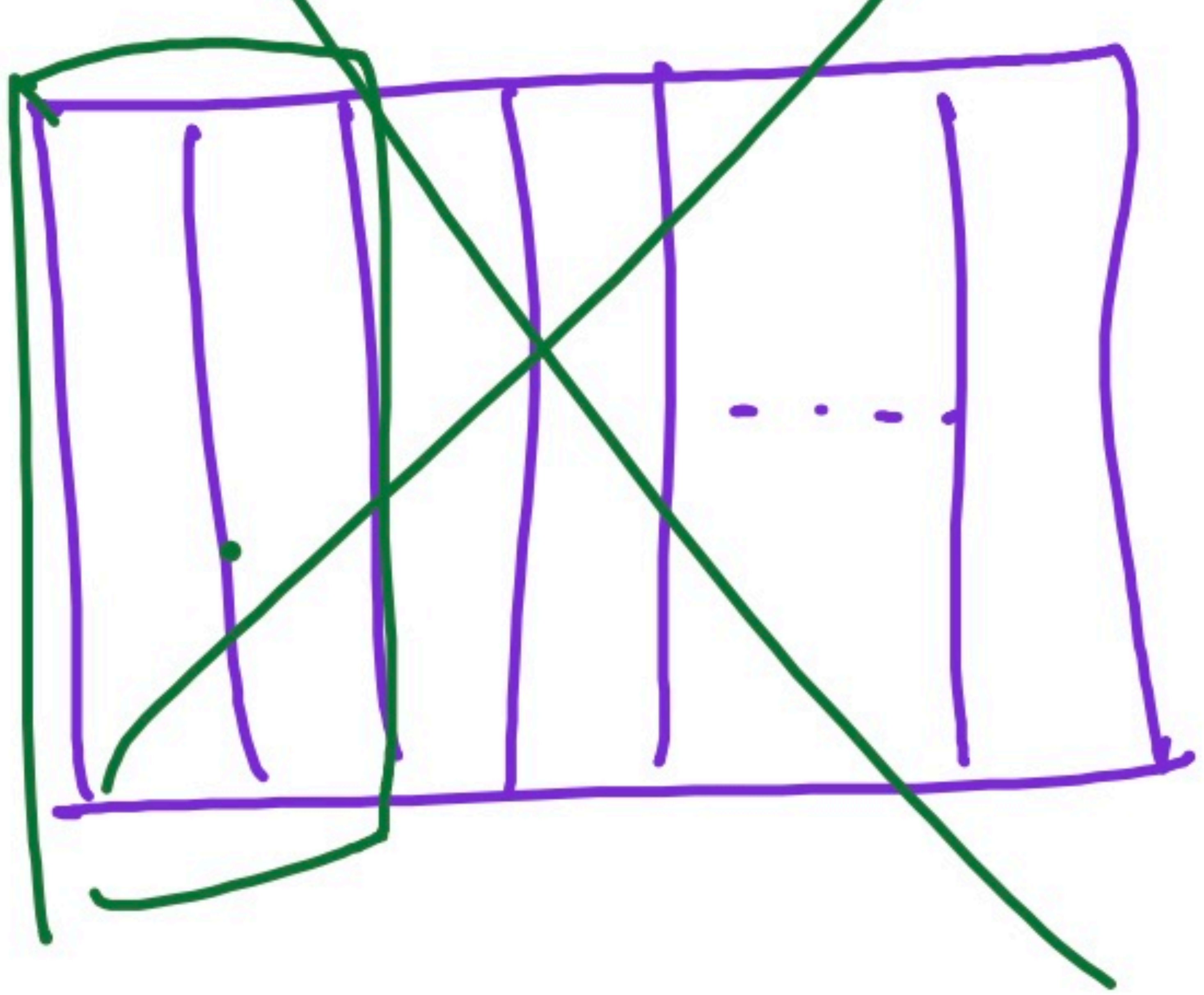
```
1 util.plot_model_coefficients(model_log.coef_)
```



Principal component analysis (PCA)

- Principal component analysis (PCA) is an **unsupervised learning** technique used for **dimensionality reduction**.

- It'll allow us to take:
  - `X_train`, which has 60,000 rows and 784 columns, and transform it into
  - `X_train_approx`, which has 60,000 rows and p columns, where p is as small as we want (e.g. $p = 2$).



Principal component analysis (PCA)

- Principal component analysis (PCA) is an **unsupervised learning** technique used for **dimensionality reduction**.

- It'll allow us to take:
  - `X_train`, which has 60,000 rows and 784 columns, and transform it into
  - `X_train_approx`, which has 60,000 rows and p columns, where p is as small as we want (e.g. $p = 2$).

- It creates p **new features**, each of which is a linear combination of all existing 784 features.

$$\text{new feature } 1_i = 0.05 \cdot \text{pixel } 1_i + 0.93 \cdot \text{pixel } 2_i + \ldots - 0.35 \cdot \text{pixel } 784_i$$

$$\text{new feature } 2_i = -0.06 \cdot \text{pixel } 1_i + 0.5 \cdot \text{pixel } 2_i + \ldots + 0.04 \cdot \text{pixel } 784_i$$

$$\ldots$$

These new features are chosen to capture as much variability (information) in the original data as possible.

- How? The details are out of scope for us, but it leverages the **singular value decomposition** from linear algebra:

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

```
PCA(n_components=2)
```

- Once `fit`, `pca` can transform `X_train` into a **2-column matrix** in a way that retains the bulk of the information:

$$\mathbb{R}^{60000 \times 784} \rightarrow \mathbb{R}^{60000 \times 2}$$

In [91]:
```python
X_train_approx = pca.transform(X_train)
X_train_approx.shape
```

Out[91]:
```
(60000, 2)
```

In [93]:
```python
X_train_approx
```

Out[93]:
```
array([[ 123.93,  312.67],
       [1011.72,  294.86],
       [ -51.85, -392.17],
       ...,
       [-178.05, -160.08],
       [ 130.61,    5.59],
       [-173.44,   24.72]])
```

each image is now represented by just 2 values

negatives are possible!