Problem I: SVM on MNIST dataset

1. Code:

The code can be divided into 5 parts:

- Linear kernel
- Polynomial kernel
- RBF kernel
- Precomputed kernel (linear + RBF)
- Main

i. Linear kernel part:

→ Convert csv data files into numpy arrays.

```
def gen_data(filename):
    return np.genfromtxt(filename, delimiter=',')

```

→ Set svm_train parameter into '-t 0' so that the kernel function is linear.

→ Applying grid-search to tune the parameter c (i.e. set '-c c_try' where c_try = $\{2^{-5}, 2^{-3}, \ldots, 2^{13}, 2^{15}\}$). The chosen trial of c is based on the description of LIBSVM guide.

→ Compare results of grid-search. (The result is shown in **Result comparison** section.)

```python
def train_with_linear_kernel():
    y = gen_data('Y_train.csv')
    x = gen_data('X_train.csv')
    yt = gen_data('Y_test.csv')
    xt = gen_data('X_test.csv')

    """
    Linear kernel: -t 0
    Applying grid search: c = {2^-5, 2^-3, 2^-1, ..., 2^11, 2^13, 2^15}
    """
    C_grid_search = [2**c for c in range(-5, 16, 2)]
    compare = []
    for c_try in C_grid_search:
        param = '-t 0 -c {}'.format(c_try)
        model = svm_train(y, x, param)

        print('test:')
        p_label, p_acc, p_val = svm_predict(yt, xt, model)
        print(p_acc[0])
        compare.append(p_acc[0])
    print('compare: ')
    print(compare)

```

ii. Polynomial kernel part:

→ Convert csv data files into numpy arrays.

→ Set '-t 1' to use polynomial kernel function.

→ Applying grid-search on parameter c, gamma (try c in $\{2^{-5}, 2^{-3}, \ldots, 2^{13}, 2^{15}\}$, gamma in $\{2^{-15}, 2^{-13}, \ldots, 2^{1}, 2^{3}\}$). The chosen trial of c and gamma is based on the description of LIBSVM guide.

→ Compare results of grid-search. (The result is shown in **Result comparison** section.)

```python
def train_with_polynomial_kernel():
    y = gen_data('Y_train.csv')
    x = gen_data('X_train.csv')
    yt = gen_data('Y_test.csv')
    xt = gen_data('X_test.csv')
    """
    Linear kernel: -t 1
    Applying grid search on c: c = {2^-5, 2^-3, 2^-1, ..., 2^11, 2^13, 2^15}
    Applying grid search on gamma: gamma = {2^-15, 2^-13, ..., 2^1, 2^3}
    """
    C_grid_search = [2**c for c in range(-5, 16, 2)]
    Gamma_grid_search = [2**g for g in range(-15, 4, 2)]
    compare = []
    for c_try in C_grid_search:
        gamma_iter_compare = []
        for g_try in Gamma_grid_search:
            param = '-t 1 -c {} -g {}'.format(c_try, g_try)
            model = svm_train(y, x, param)

            print('test:')
            p_label, p_acc, p_val = svm_predict(yt, xt, model)
            print(p_acc[0])
            gamma_iter_compare.append(p_acc[0])
        compare.append(gamma_iter_compare)
    print('compare: ')
    print(compare)
```

iii. <u>RBF kernel part</u>:

Set parameter to ‘-t 2’ so that the kernel function is RBF. And the remaining procedures are same as polynomial kernel part.

```python
def train_with_RBF_kernel():
    y = gen_data('Y_train.csv')
    x = gen_data('X_train.csv')
    yt = gen_data('Y_test.csv')
    xt = gen_data('X_test.csv')
    """
    Linear kernel: -t 2
    Applying grid search on c: c = {2^-5, 2^-3, 2^-1, ..., 2^11, 2^13, 2^15}
    Applying grid search on gamma: gamma = {2^-15, 2^-13, ..., 2^1, 2^3}
    """
    C_grid_search = [2**c for c in range(-5, 16, 2)]
    Gamma_grid_search = [2**g for g in range(-15, 4, 2)]
    compare = []
    for c_try in C_grid_search:
        gamma_iter_compare = []
        for g_try in Gamma_grid_search:
            param = '-t 2 -c {} -g {}'.format(c_try, g_try)
            model = svm_train(y, x, param)

            print('test: c={}, g={}'.format(c_try, g_try))
            p_label, p_acc, p_val = svm_predict(yt, xt, model)
            print(p_acc[0])
            gamma_iter_compare.append(p_acc[0])
        compare.append(gamma_iter_compare)
    print('compare: ')
    print(compare)
```

iv. Precomputed kernel function (linear + RBF kernel):

→ Set svm_parameter to '-t 4' so that the model can use precomputed kernel function.

→ Calculate linear kernel function: $K(x1, x2) = x1 \cdot x2^T$ (Refer to this link)

```python
def train_with_precomputed_kernel():
    def linear_kernel(x1, x2):
        n1 = x1.shape[0]
        n2 = x2.shape[0]
        K_train = np.zeros((n1, n2 + 1))
        K_train[:, 1:] = np.dot(x1, x2.T)
        K_train[:, 0] = np.arange(n1) + 1
        return K_train

```

→ Calculate RBF kernel function: $K(x1, x2) = \exp(-\gamma * ||x1-x2||^2) =$

exp^(-gamma*($x1^2$ + $x2^2$ – 2·x1·$x2^T$) (Refer to this link)

```python
def RBF_kernel(x1, x2, gamma):
    n1 = x1.shape[0]
    n2 = x2.shape[0]
    K_train = np.zeros((n1, n2 + 1))
    x1_norm = np.sum(x1**2, axis=-1)
    x2_norm = np.sum(x2**2, axis=-1)
    dist = x1_norm[:, None] + x2_norm[None, :] - 2 * np.dot(x1, x2.T)
    K_train[:, 1:] = np.exp(-gamma * dist)
    K_train[:, 0] = np.arange(n1) + 1
    return K_train
```

→ Use Linear + RBF to be our new kernel function.

```python
def precomputed_kernel(x1, x2, gamma):
    linear_rbf = linear_kernel(x1, x2) + RBF_kernel(x1, x2, gamma)
    linear_rbf[:, 0] /= 2
    return linear_rbf
```

→ Use our new kernel function to compute our new training data and feed it into the svm model. Also, use grid-search to tune for proper parameters for c and gamma. Finally, compare results of grid-search.

```python
    y = gen_data('Y_train.csv')
    x = gen_data('X_train.csv')
    yt = gen_data('Y_test.csv')
    xt = gen_data('X_test.csv')

    C_grid_search = [2**c for c in range(-5, 16, 2)]
    Gamma_grid_search = [2**g for g in range(-15, 4, 2)]
    compare = []
    for c_try in C_grid_search:
        gamma_iter_compare = []
        for g_try in Gamma_grid_search:
            param = '-t 4 -c {} -g {}'.format(c_try, g_try)
            x_train = precomputed_kernel(x, x, g_try)
            model = svm_train(y, x_train, param)

            print('test: c={}, g={}'.format(c_try, g_try))
            x_test = precomputed_kernel(xt, x, g_try)
            p_label, p_acc, p_val = svm_predict(yt, x_test, model)
            print(p_acc[0])
            gamma_iter_compare.append(p_acc[0])
        compare.append(gamma_iter_compare)
    print('compare: ')
    print(compare)
```

v. Main:

Simple procedure calls for each kernel function type.

```python
if __name__ == "__main__":
    """ Uncommend the kernel_function_type you want for svm model """
    # train_with_linear_kernel()
    # train_with_polynomial_kernel()
    # train_with_RBF_kernel()
    train_with_precomputed_kernel()
```

2. Result comparison:

i. Linear kernel:

C	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	2^{9}	2^{11}	2^{13}	2^{15}
Accuracy	96%	95.92%	95.52%	95%	95%	95%	95%	95%	95%	95%	95%

✧ Linear kernel model has the best performance with parameter $c=2^{-5}$.

✧ When c grows, accuracy decreases and end up converges to 95%.

✧ The accuracy lies within 95% - 96% for linear kernel.

ii. Polynomial kernel:

Gamma \ C	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	2^{9}	2^{11}	2^{13}	2^{15}
2^{-15}	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%
2^{-13}	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	28.88%	43.88%	74.88%	88.84%
2^{-11}	28.88%	28.88%	28.88%	28.88%	28.88%	43.88%	28.88%	28.88%	93.64%	97.04%	97.8%
2^{-9}	28.88%	28.88%	43.88%	74.88%	88.84%	93.64%	74.88%	88.84%	97.48%	97.48%	97.48%
2^{-7}	74.88%	88.84%	93.64%	97.04%	97.8%	97.48%	97.04%	97.8%	97.48%	97.48%	97.48%
2^{-5}	97.04%	97.8%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%
2^{-3}	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%
2^{-1}	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%
2^{1}	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%
2^{3}	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%	97.48%

- ✧ Polynomial kernel model has the best performance with parameter (c, gamma) = $(2^{-3}, 2^{-5})$, $(2^{3}, 2^{-7})$, $(2^{9}, 2^{-7})$, $(2^{15}, 2^{-11})$
- ✧ Mostly, accuracy increases with growing gamma and end up converges to 97.48%. However, when c is also big, accuracy increases faster.
- ✧ In total, accuracy lies within 28.88% - 97.08% for polynomial kernel.

iii. <u>RBF kernel:</u>

C Gamma	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	2^{9}	2^{11}	2^{13}	2^{15}
2^{-15}	79.44%	79.44%	80.0%	90.68%	94.04%	95.16%	95.84%	96.0%	95.92%	95.52%	95.04%
2^{-13}	79.52%	80.08%	90.76%	94.0%	95.16%	95.84%	96.04%	96.0%	95.72%	95.39%	95.40%
2^{-11}	80.64%	90.8%	94.04%	95.16%	95.92%	96.2%	96.44%	96.08%	96.08%	96.08%	96.08%
2^{-9}	90.76%	94.04%	95.32%	96.24%	96.8%	97.24%	97.2%	97.24%	97.24%	97.24%	97.24%
2^{-7}	93.76%	95.48%	96.72%	97.64%	98.04%	98.04%	98.04%	98.04%	98.04%	98.04%	98.04%
2^{-5}	94.6%	96.6%	98.04%	98.52%	98.52%	98.52%	98.52%	98.52%	98.52%	98.52%	98.52%
2^{-3}	41.4%	45.6%	57.92%	83.24%	83.24%	83.24%	83.24%	83.24%	83.24%	83.24%	83.24%
2^{-1}	20.76%	20.76%	28.20%	43.76%	43.76%	43.76%	43.76%	43.76%	43.76%	43.76%	43.76%
2^{1}	20.08%	20.08%	20.08%	25.72%	25.72%	25.72%	25.72%	25.72%	25.72%	25.72%	25.72%
2^{3}	78.64%	78.64%	78.64%	20.64%	20.64%	20.64%	20.64%	20.64%	20.64%	20.64%	20.64%

- ✧ Polynomial kernel model has the best performance with parameter (c, gamma) = (c >= 2^{1}, 2^{-5})
- ✧ Accuracy increases with growing gamma when gamma <= 2^{-5}. When gamma > 2^{-5}, accuracy starts to decreases, which seems overfitting.
- ✧ For polynomial kernel, accuracy lies within 20% - 99%.

iv. <u>Precomputed kernel: (Linear + RBF kernel)</u>

C Gamma	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	2^{9}	2^{11}	2^{13}	2^{15}
2^{-15}	96.0%	95.92%	95.52%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%
2^{-13}	96.0%	95.92%	95.52%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%
2^{-11}	96.0%	95.92%	95.52%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%
2^{-9}	96.0%	95.92%	95.52%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%	95.0%
2^{-7}	96.04%	95.92%	95.6%	95.16%	95.16%	95.16%	95.16%	95.16%	95.16%	95.16%	95.16%
2^{-5}	96.12%	96.08%	95.8%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%
2^{-3}	96.0%	95.96%	95.76%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%
2^{-1}	95.96%	95.96%	95.8%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%
2^{1}	95.96%	95.96%	95.8%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%
2^{3}	95.96%	95.96%	95.8%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%	95.64%

- ✧ Linear + RBF kernel model has the best performance with parameter (c, gamma) = (2^{-5}, 2^{-5})
- ✧ According to the result table, we can find that no matter how we choose the parameter c and gamma, the accuracy will always be high (about 95%-96%).

v. Conclusion:

- ✧ According to the experiment results, I found that using RBF kernel can get higher accuracy, and using linear+RBF kernel can guarantee good and stable performance for all c and gamma.
- ✧ It is believed that an multiple kernel model has the ability to select for an optimal kernel. However, it is surprising that the result of linear+RBF is quite similar to linear kernel, and I don't know what exactly the reason is.

Problem II: Find out support vector

1. Code:

The code can be divided into 3 parts:

- Train with linear, polynomial, RBF, and linear+RBF kernel
- Find out support vectors
- Visualization

i. <u>Train:</u>

The code for training is similar to problem I. The only thing which is different is that I use default parameter given by LIBSVM library for linear, polynomial, and RBF kernel. For linear+RBF kernel, I set parameter gamma to 2^{-5} according to experiments so that it will get the best result.

```
def train_with_linear_kernel(y, x):
    param = '-t 0 -h 0'
    model = svm_train(y, x, param)

    print('test:')
    p_label, p_acc, p_val = svm_predict(y, x, model)
    print(p_label)
    p_label = np.array(p_label)
    return p_label, model
```

```
def train_with_polynomial_kernel(y, x):
    param = '-t 1 -h 0'
    model = svm_train(y, x, param)

    print('test:')
    p_label, p_acc, p_val = svm_predict(y, x, model)
    print(p_label)
    p_label = np.array(p_label)
    return p_label, model
```

```
def train_with_RBF_kernel(y, x):
    param = '-t 2 -h 0'
    model = svm_train(y, x, param)

    print('test:')
    p_label, p_acc, p_val = svm_predict(y, x, model)
    print(p_label)
    p_label = np.array(p_label)
    return p_label, model
```

```python
def train_with_precomputed_kernel(y, x):
    def linear_kernel(x1, x2):
        n1 = x1.shape[0]
        n2 = x2.shape[0]
        K_train = np.zeros((n1, n2 + 1))
        K_train[:, 1:] = np.dot(x1, x2.T)
        K_train[:, 0] = np.arange(n1) + 1
        return K_train

    def RBF_kernel(x1, x2, gamma):
        n1 = x1.shape[0]
        n2 = x2.shape[0]
        K_train = np.zeros((n1, n2 + 1))
        x1_norm = np.sum(x1**2, axis=-1)
        x2_norm = np.sum(x2**2, axis=-1)
        dist = x1_norm[:, None] + x2_norm[None, :] - 2 * np.dot(x1, x2.T)
        K_train[:, 1:] = np.exp(-gamma * dist)
        K_train[:, 0] = np.arange(n1) + 1
        return K_train

    def precomputed_kernel(x1, x2, gamma):
        linear_rbf = linear_kernel(x1, x2) + RBF_kernel(x1, x2, gamma)
        linear_rbf[:, 0] /= 2
        return linear_rbf

    param = '-t 4'
    x_train = precomputed_kernel(x, x, 2**-5)
    model = svm_train(y, x_train, param)

    print('test: ')
    p_label, p_acc, p_val = svm_predict(y, x_train, model)
    print(p_acc[0])
    p_label = np.array(p_label)
```

ii. Support vectors:

To find out support vectors for each model, I use get_sv_indices() method provided by LIBSVM library.

```python
    linear_sv_idx = np.array(linear_model.get_sv_indices()) - 1
    poly_sv_idx = np.array(poly_model.get_sv_indices()) - 1
    rbf_sv_idx = np.array(rbf_model.get_sv_indices()) - 1
    linear_rbf_sv_idx = np.array(linear_rbf_model.get_sv_indices()) - 1
```

iii. Visualization:

I delared a new class named Visualization to implement the visualization.

Firstly, in constructor, some variables which will be used when visualization was defined.

```python
class Visualization:
    def __init__(self, y, x):
        self.title = [
            'Linear kernel', 'Polynomial kernel', 'RBF kernel',
            'Linear + RBF kernel'
        ]
        self.y = y
        self.x = x
```

Secondly, in plot_svm_cluster method, different colors were given to distinguish different clusters. (i.e. red for cluster-0, green for cluster-1, blue for cluster-2). Also, different marker shapes were given to distinguish different support vectors. (i.e. square for linear kernel support vectors, triangle for polynomial kernel support vectors, X for RBF kernel support vectors, and diamond for linear+RBF kernel support vectors. If it’s not a support vector, the shape will be dot.)

```python
    def plot_svm_cluster(self, predict_label, graph_idx, sv_idx):
        n = int(predict_label.shape[0])
        shape_type = ['s', '^', 'x', 'd']
        marker_color = []
        marker_shape = []
        for i in range(n):
            # different color for different cluster
            if predict_label[i] == 0:
                marker_color.append('r')
            elif predict_label[i] == 1:
                marker_color.append('g')
            else:
                marker_color.append('b')

            # different shape for different support vector
            if i in sv_idx:
                marker_shape.append(shape_type[graph_idx])
            else:
                marker_shape.append('.')
        plt.subplot(2, 2, graph_idx + 1)
        plt.title(self.title[graph_idx])
        x1_axis = self.x.T[0]
        x2_axis = self.x.T[1]
        for i in range(n):
            plt.scatter(
                x1_axis[i],
                x2_axis[i],
                s=16,
                marker=marker_shape[i],
                c=marker_color[i])
```

Finally, show the graph.

```python
    def show_graph(self):
        plt.tight_layout()
        plt.show()
```

2. Result comparison:

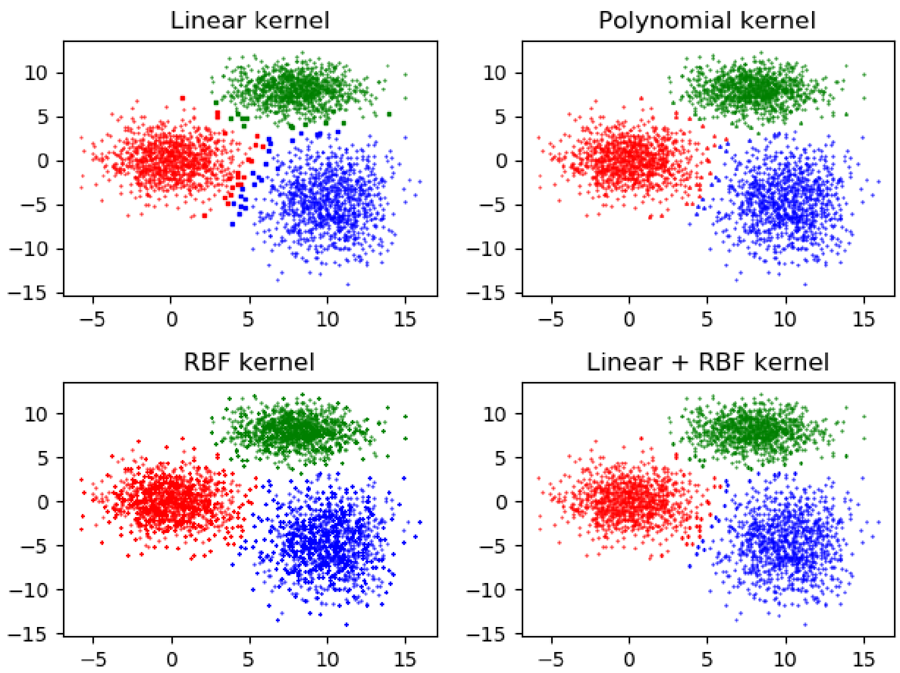
i. Accuracy and number of support vectors:

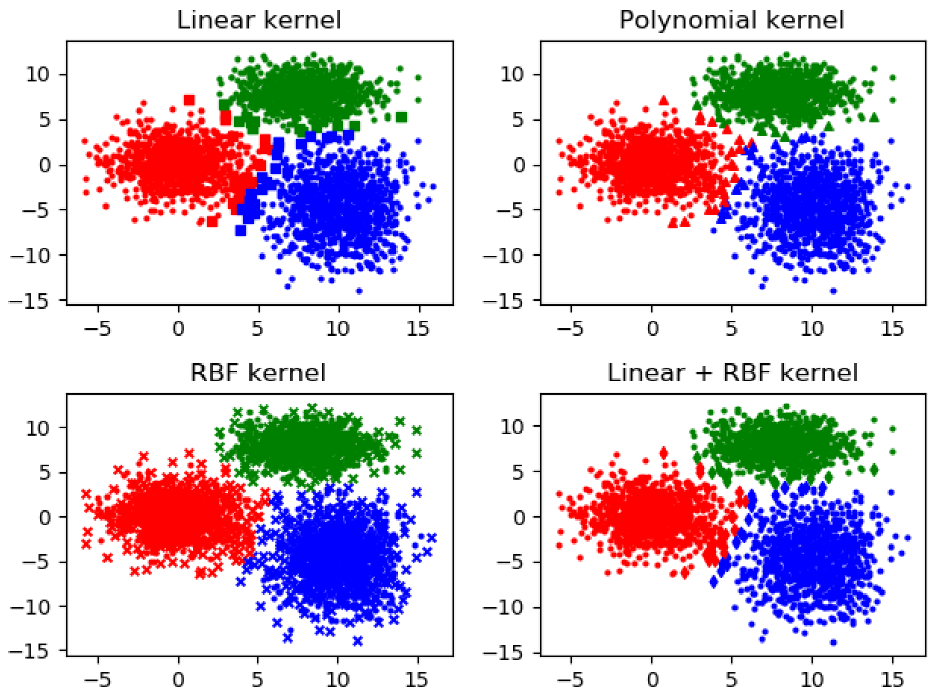
Kernel	Linear	Polynomial	RBF	Linear + RBF
Accuracy	99.57%	99.33%	99.47%	99.5%
# of SV	55	48	1116	54

ii. Visualization:

I plotted two figures so that the marker shapes can be clearly seen.

According to the figures, we can find the for RBF kernel, the number of support vectors are far more than that in others.





iii. Result discussion:

- ✧ In this dataset, linear+RBF kernel has best performance on accuracy due to the ability for multi kernel model to choose optimal kernel (linear kernel in this case).
- ✧ Often, a large number of support vectors is a sign of overfitting. Therefore, I found that RBF kernel seems easy to cause overfitting according to the result from problem I and the figure from problem ||.