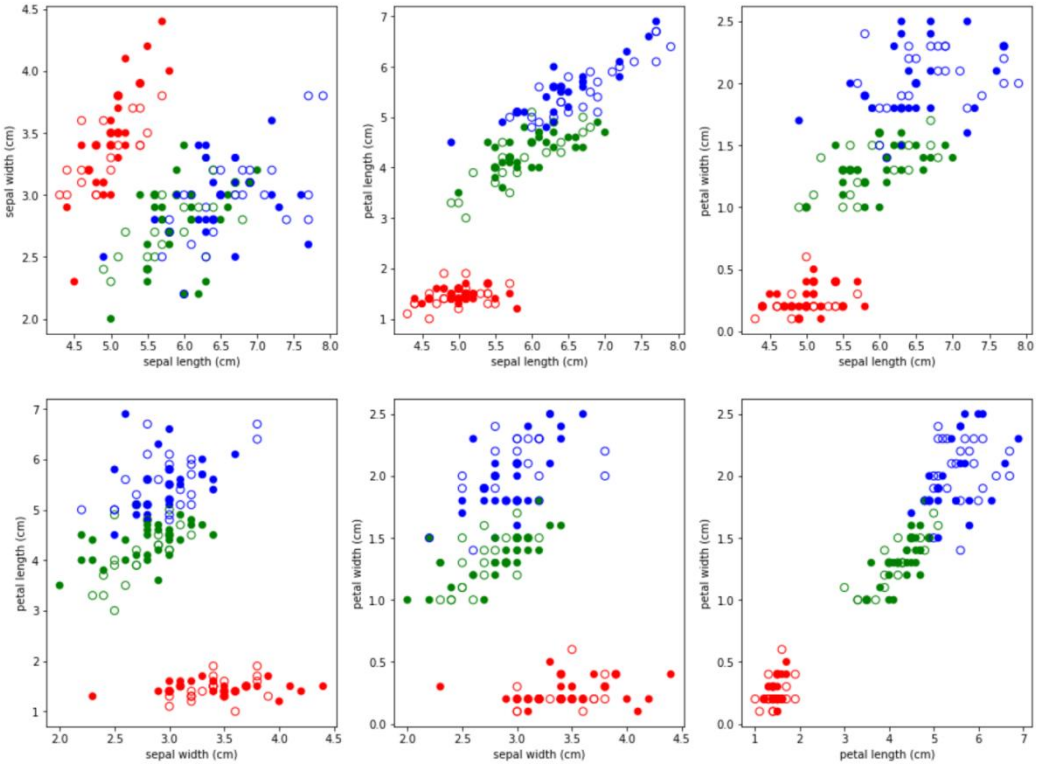
**Question 1:** Why do we have six different point clouds? What does each figure represent?

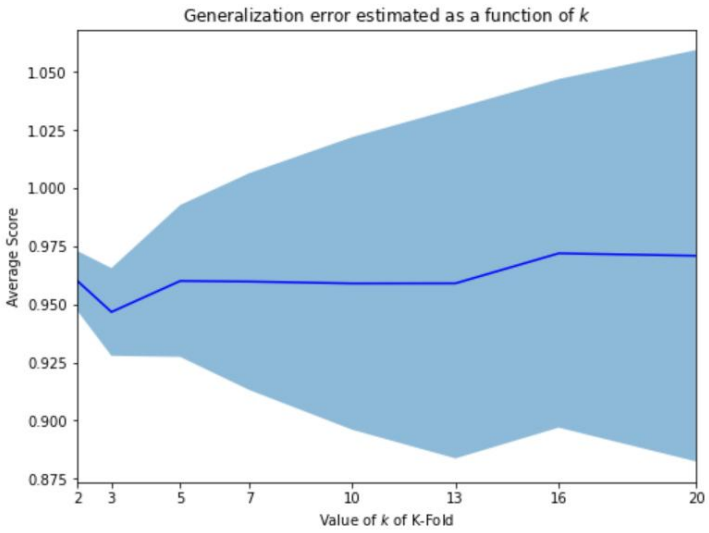
**Answer:** There are four variables of each data elements, which are sepal length, sepal width, petal length and petal width. Each time we choose two variables, then we got six combinations totally, so there are six different point clouds.

Each figure represents the points distributions by two dimensions chosen of the data available, as we see, sepal length* sepal width, sepal length* petal length, sepal length* petal width, sepal width* petal length, sepal width* petal width, petal length* petal width respectively.

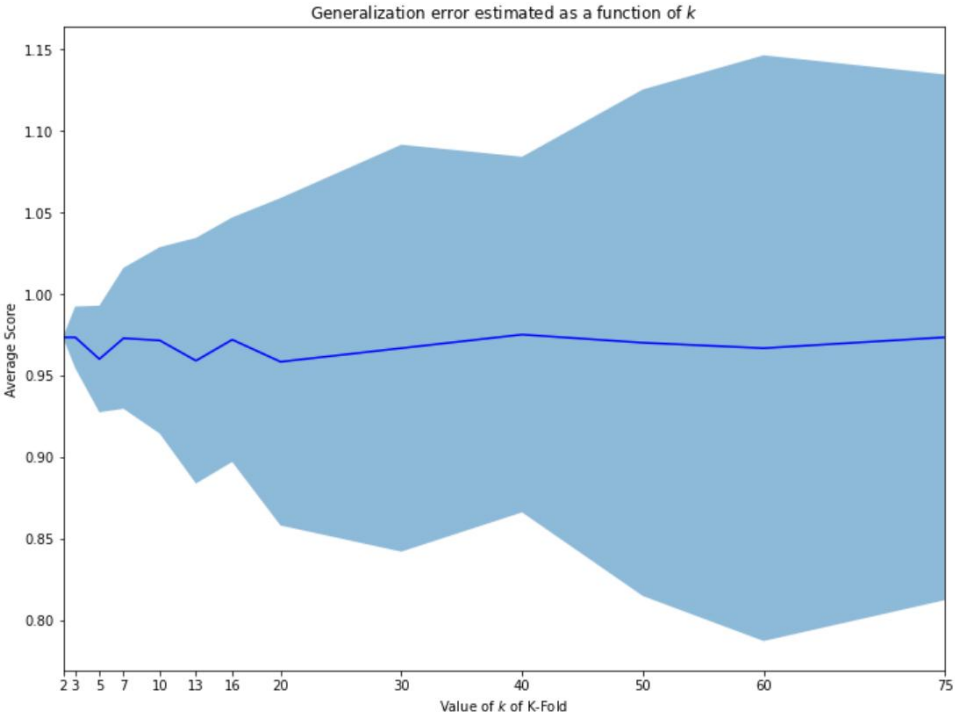



**Question 2:** What do you see when looking at this graph? Add values for k (eg 40, 100, be careful it will take longer…) and examine the graph again

**Answer:** From the graph we can see that the average score fluctuates with the value of k, and the standard variance increases as k increases. In this test, the average score reaches the peak value 0.972 at k=16.

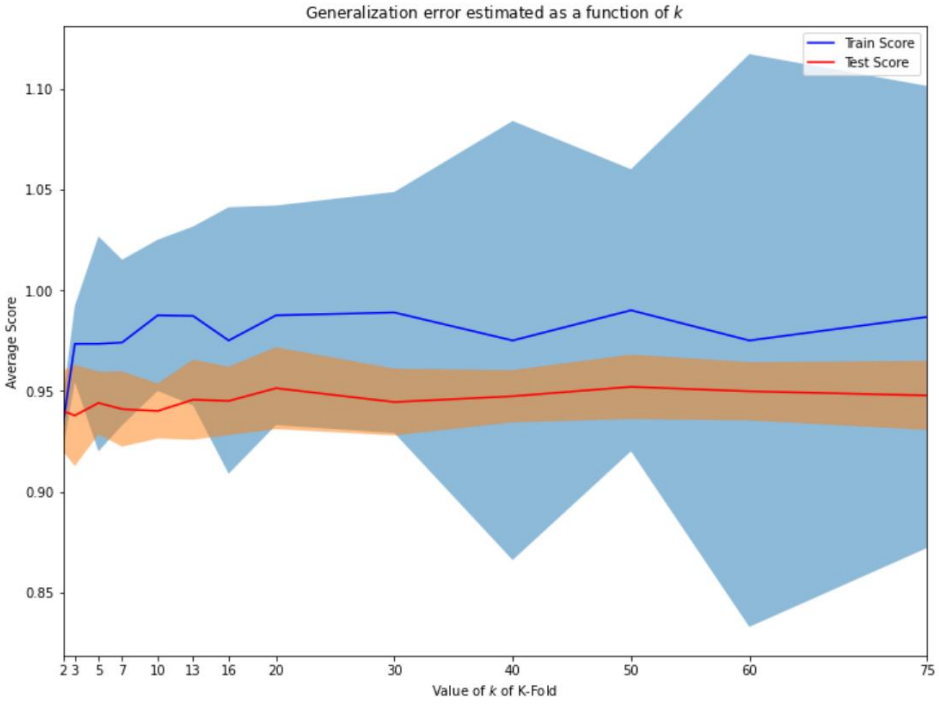



When adding more bigger values for k (30, 40, 50, 60, 75, because there are only 150 rows so the maximum k will be 150/2=75), we can see that the average score fluctuates slightly when k >20. In this test, the average score reaches the peak value 0.975 at k=40.



**Question 3:** For each model learned by k-fold cross-validation, add its evaluation on the test data set aside at the start *X_test, y_test*. Display the curves on the same graph. What do you notice?

**Answer:** When evaluated on the test data, we can see that the average test score fluctuates with the train score, and the value is lower than average train score, but the standard variance of test score is much less than that of train score. In this test, both average train score and test score reach the peak value k=50, with 0.99 and 0.95 respectively.




**Question 4:** Perform performance estimation using leave one out (LOO) cross-validation. What do you see comparing the results of *k-fold* and leave one out?

**Answer:** When we use LOO cross-validation, we can see that both the train score and test score are the same as using k-fold cross-validation with k=75.

In terms of accuracy, LOO often results in high variance as an estimator for the test error. Intuitively, since (n-1) of the (n) samples are used to build each model, models constructed from folds are virtually identical to each other and to the model built from the entire training set.

**Question 5:** How many MLPs are learned in total in this example?

**Answer:** There are 6*4=24 MLPs learned in this example in total.

Total number of MLPs = number of *hidden_layer_sizes* * number of *alpha*.

**Question 6:** What is the meaning of the *refit* parameter of *GridSearchCV*?

**Answer**: Refit parameter is used to refit an estimator using the best-found parameters on the whole dataset. It can be bool, str, or callable, default=True.

For multiple metric evaluation, it needs to be a str denoting the scorer that would be used to find the best parameters for refitting the estimator at the end. We can define a function to be passed to the refit parameter, it will implement the custom strategy to select the best candidate from the cv_results_ attribute of the GridSearchCV. Once the candidate is selected, it is automatically refitted by the GridSearchCV instance.

**Question 7:** Take a more complete look at the contents of *clf.cv_results*.

**Answer:** *clf.cv_results* contain 'mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'mean_test_score', 'std_test_score', 'rank_test_score' for each MLP learned. For fit time, it increases as parameter *hidden_layer_sizes* increases.

**Question 8:** Evaluate the selected model on the test data (*X_test*, *y_test*).

**Answer:** In this test, the selected model's parameter is hidden_layer_sizes=20, alpha=0.001, and best train score is 0.973. When this model is evaluated on test data, the test score is 0.933.

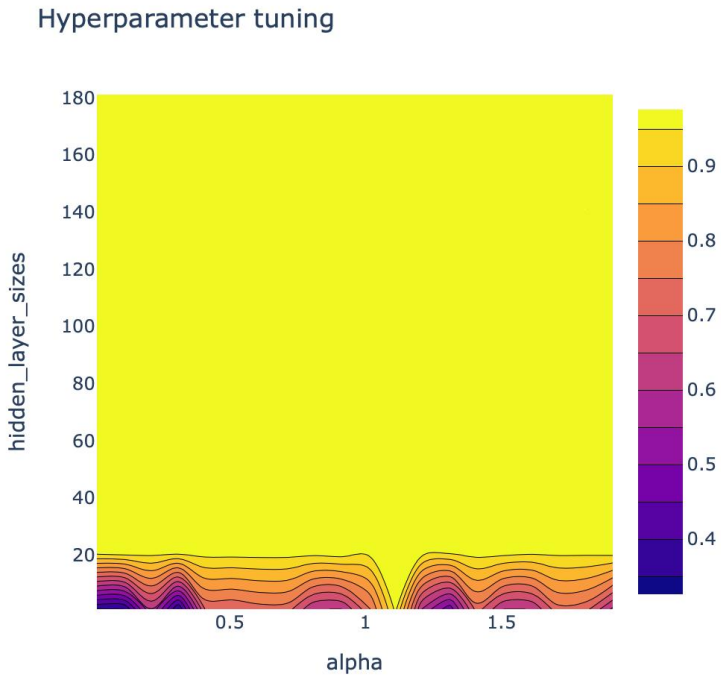
**Question 9:** Do the appearance of the results encourage you to refine the grid? Edit the grid and review the new results.

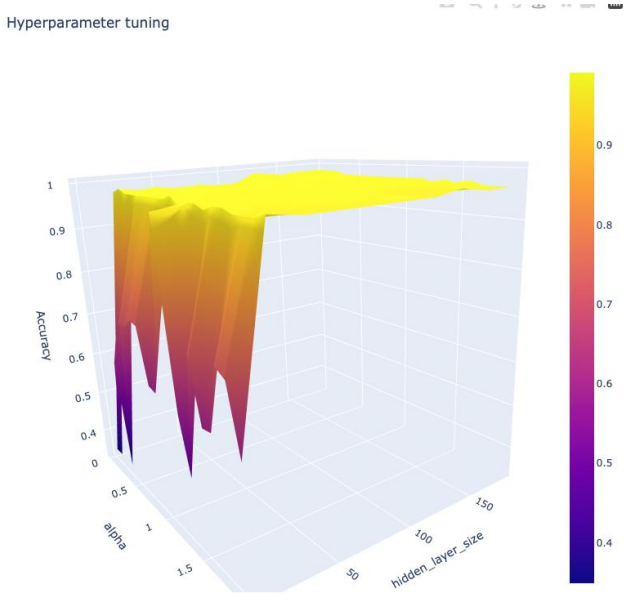
**Answer:** We set the *hidden_layer_sizes* from 1 to 100, added *alpha* value 0.0001, set max_iter=1000, and got a better model with parameter hidden_layer_sizes=31, alpha=0.0001, and best train score increased to 0.987, but the test score on test data is still 0.933.

```
tuned_parameters = { #'hidden_layer_sizes':[(5,), (20,), (50,), (100,), (150,), (200,)],
                     'hidden_layer_sizes': np.arange(1,200),
                     'alpha':   [0.0001, 0.001, 0.01, 1, 2]}
```

```
clf = GridSearchCV(MLPClassifier(solver='lbfgs', tol=5e-3, max_iter=1000), tuned_parameters, cv=5, refit=True)

# grid search execution
clf.fit(X_train, y_train)
```




We plot the GridSearchCV as 2D and 3D surface, so we can have a general picture of the parameters. As we can see above, as the hidden layer increase, go above 20 layers, the

alpha is not matter anymore. But if we only use 1-20 layers, the best alpha is around 1.1.

**Question 10:** Use randomized search with RandomizedSearchCV. The "cost" (total number of evaluated combinations) can be set with n_iter. Explain the choice of the laws used for drawing the values of the two (hyper)parameters hidden_layer_sizes and alpha.

**Answer:** By using *RandomizedSearchCV*, we set *hidden_layer_sizes* as random integer from 1 to 200, set *alpha* as random value from [0.0001, 0.001, 0.01, 1, 2], and set n_iter=100, with 100 evaluated combinations in total.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
```

```
param_dist = {"hidden_layer_sizes": randint(1, 200),
              "alpha": [0.0001, 0.001, 0.01, 1, 2]}
```

```
clf = RandomizedSearchCV(MLPClassifier(solver='lbfgs', tol=5e-3, max_iter=1000),
                         param_dist,n_iter=100, cv=5, refit=True,random_state=0)
```

```
clf.fit(X_train, y_train)
```

In this test, we got the best model with parameter hidden_layer_sizes=68, alpha=0.0001, and best train score is 0.987, and the test score on test data is 0.907.

**Question 11:** Experiment with *HalvingGridSearchCV*. Compare the final score and the calculation time with that obtained with *GridSearchCV*.

**Answer:** By using *HalvingGridSearchCV*, we set the same hidden_layer_sizes and alpha parameter as *GridSearchCV* in Question 9.

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
```

```
param_grid = {"hidden_layer_sizes": np.arange(1,200),
              "alpha": [0.0001, 0.001, 0.01, 1, 2]}
```

```
clf = HalvingGridSearchCV(MLPClassifier(solver='lbfgs', tol=5e-3, max_iter=1000),
                          param_grid, max_resources=30,random_state=0)
```

```
clf.fit(X_train, y_train)
```

In this test, we got the best model with parameter hidden_layer_sizes=23, alpha=0.0001.

The final score: best train score 1.0, the test score on test data 0.96.

Both train score and test score are better than the best result of GridSearchCV, which is 0.987 and 0.933 respectively.

For the computation time, the HalvingGridSearchCV is faster than GridSearchCV, because HalvingGridSearchCV starts evaluating all the candidates with a small amount of resources and iteratively selects the best candidates, using more and more resources. So

HalvingGridSearchCV can avoid waste time in bad parameters.