Lab Course Machine Learning
Exercise 7

1.1 Data preprocessing:

Function name : read_data
Parameter : Filename and column names

This function reads the data and returns the data in the format of data frame along with the headings. The function numeric_class is used to convert the class names to numbers

```
irisdata = pd.read_csv("iris.data")
irisdata = numeric_class(irisdata)
```

Output:

Iris dataset

	sepal-length	sepal-width	petal-length	petal-width	class
7	5.0	3.4	1.5	0.2	1
45	4.8	3.0	1.4	0.3	1
28	5.2	3.4	1.4	0.2	1
52	6.9	3.1	4.9	1.5	2
60	5.0	2.0	3.5	1.0	2

Function name : check_na_null
Parameter : Dataframe and column names

This function checks the presence of null values or NA or nan values within the dataset and removes the same. This function also generates dummies for non-numeric values and returns the same.

```
def check_na_null(data,column_dummies):
    data.isnull().values.any()
    data.dropna(inplace = True)
    data = pd.get_dummies(data=data, columns=column_dummies)
    return data
```

Function name : numeric_class
Parameter : Dataframe

This function replaces the names of the classes to unique numerical values

```
def numeric_class(data):
    data['class'] = pd.factorize(irisdata['class'])[0] + 1
    return data
```

Wine dataset:

To begin with the problem, first we've to find the correlation between the input variables and the quality of the red wine. As the dataset contains numerical values, we can use the Pearson co-efficient correlation between two variables.

Function name : pearson_coefficient
Parameter : Ranked data for correlation

Reference: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php

The Pearson correlation coefficient (PCC) is a measure of the linear correlation between two variables X and Y. PCC or 'r' has a value between +1 and −1.

Assumptions:

1. The variables must be either interval or ratio measurements.
2. The variables must be approximately normally distributed.
3. There is a linear relationship between the two variables
4. Outliers are either kept to a minimum or are removed entirely.
5. There is homoscedasticity of the data.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

```
def pearson_coefficient(x,y):
    x = x - np.mean(x)
    y = y - np.mean(y)
    return (np.sum(x*y)/np.sqrt(np.sum(x*x)*np.sum(y*y)))
```

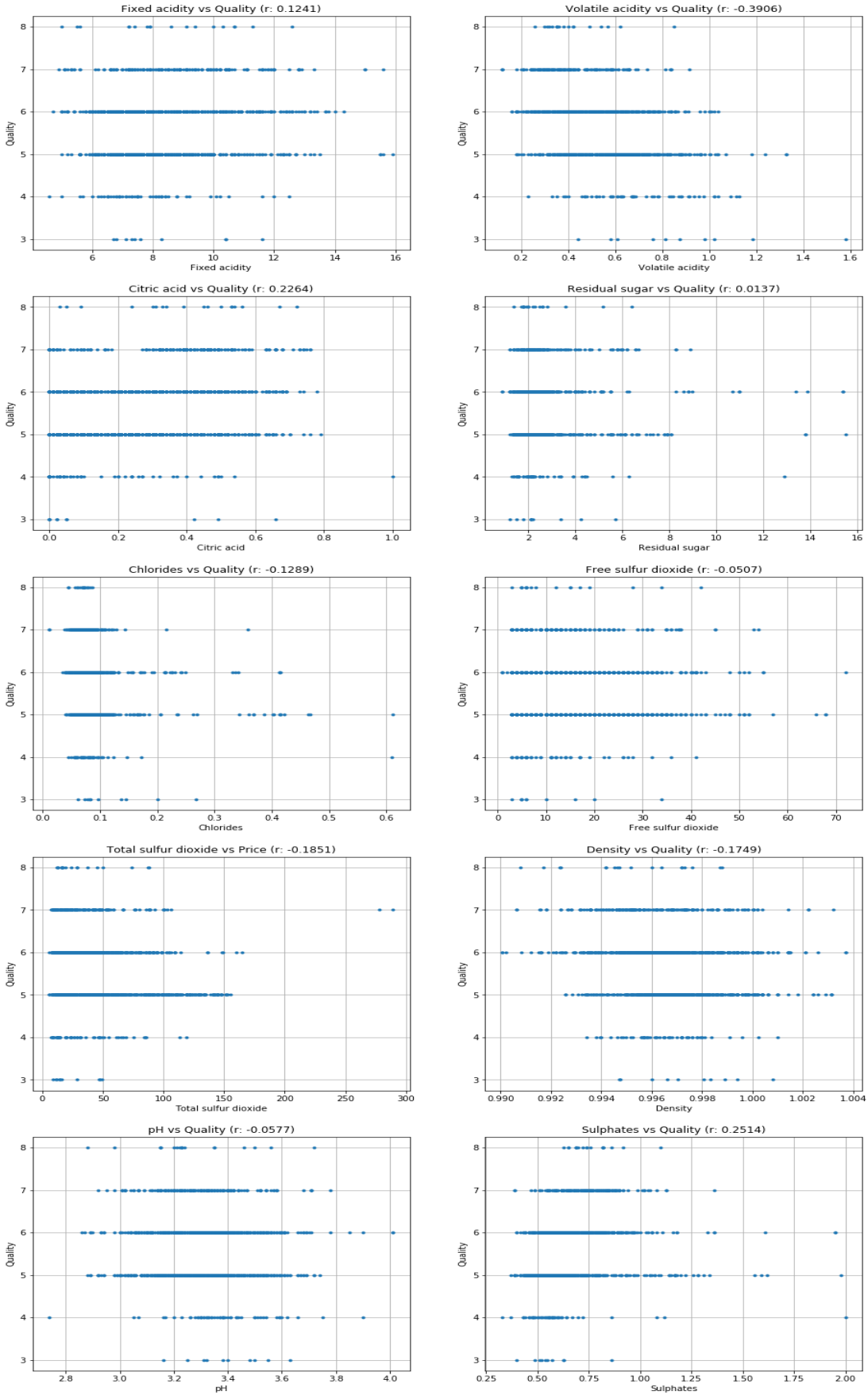
Output:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0.1241	-0.3906	0.2264	0.0137	-0.1289	-0.0507	-0.1851	-0.1749	-0.0577	0.2514	0.4762	NaN

From the correlation between the parameters and quality, we can conclude that there is no high relation between residual sugar, free sulphur dioxide, pH and quality. Hence, I am dropping out these columns from the dataset to be used for creating the model. After creating the model, as it could be seen that there's poor correlation between fixed acidity, chlorides, total sulphur dioxide and quality. At last, these columns are also dropped down from the dataset.

	volatile acidity	citric acid	density	sulphates	alcohol	quality
0	0.70	0.00	0.9978	0.56	9.4	5
1	0.88	0.00	0.9968	0.68	9.8	5
2	0.76	0.04	0.9970	0.65	9.8	5
3	0.28	0.56	0.9980	0.58	9.8	6
4	0.70	0.00	0.9978	0.56	9.4	5

Correlation between variables and Quality



Exercise 1: Implement K-Nearest Neighbour (KNN)

Function name : euclidean_distance
Parameter : Two queries for which distances is to be measured

This function is used to find the Euclidean distance between two queries.

$$d(x_i, x_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \ldots + (x_{ip} - x_{lp})^2}$$

```
def euclidean_distance(train_var,test_var):
    distance = 0
    for i in range(0,len(train_var)):
        distance += (train_var[i] - test_var[i]) ** 2
    return np.sqrt(distance)
```

Function name : predict_class
Parameter : list of the index of neighbours

This function is used to predict the class of the input query based on the majority voting of the neighbour data points' class. In case of a tie, a random class is selected from the tie.

```
def predict_class(neighbours):

    neighbours_count = Counter(neighbours)
    predictedclass = list()
    maximum = max(neighbours_count.values())
    for i in (neighbours_count):
        if neighbours_count[i] == maximum:
            predictedclass.append(i)
#     return predictedclass[0]
    if len(predictedclass)>1:
        return random.choice(predictedclass)
    else:
        return predictedclass[0]
```

Function name : accuracy
Parameter : list of the truth and predicted values of target

This function is used to find the accuracy of the predicted target from the truth value of the target.

```
def accuracy(y_test,y_pred_test):
    misclassified_test = 0
    for index,i in enumerate(y_test):
        if i != y_pred_test[index]:
            misclassified_test += 1
    return (len(y_test) - misclassified_test)/len(y_test)
```

Function name : split_data
Parameter : Data of X, target and the split size

This function is used to split the data into test and train for the data points and the target.

```
def split_data(X,y,testsize):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testsize)
    return X_train, X_test, y_train, y_test
```

Function name : knn_
Parameter : Train and test data of input query and target and value of k-neighbour

This function is used to find the top K neighbours for a given query, based on majority voting, the query is classified to the majority voted class of the neighbours and the prediction quality is given in the form of accuracy. As it is a classification problem, based on the misclassification rate, higher the accuracy, better the the model of prediction will be. As a result of which, accuracy is as a quality of prediction

```
def knn_(X_train, X_test, y_train, y_test,k_val):

    mcrate_train = list()
    mcrate_test = list()
    y_pred = list()
    for k in range (1,k_val+1):
        neighbours_test = list()
        distance_test = dict()
        y_pred_test = list()
        neighbours_train = list()
        distance_train = dict()
        y_pred_train = list()
        for i in range(0,len(X_test)):
            test_var = X_test[i]
            for j in range(0,len(X_train)):
                train_var = X_train[j]
                distance_test[(euclidean_distance(train_var,test_var)),j,i] =j
            for it,a in enumerate(sorted(distance_test)):
                if it < k:
                    neighbours_test.append(y_train[distance_test[a]])
            y_pred_test.append(predict_class(neighbours_test))
            distance_test = dict()
            neighbours_test = list()
        mcrate_test.append(accuracy(y_test,y_pred_test))
        y_pred.append(y_pred_test)
    return mcrate_test,y_pred
```

Output:

```
K:  15
Accuracy of prediction:  0.9777777777777777
Number of misclassifications:  1.0
```

	sepal-length	sepal-width	petal-length	petal-width	class	Prediction Class
0	7.7	3.8	6.7	2.2	Iris-Virginica	Iris-Virginica
1	7.2	3.6	6.1	2.5	Iris-Virginica	Iris-Virginica
2	6.3	2.5	5.0	1.9	Iris-Virginica	Iris-Virginica
3	4.5	2.3	1.3	0.3	Iris-Setosa	Iris-Setosa
4	6.8	3.0	5.5	2.1	Iris-Virginica	Iris-Virginica
5	7.4	2.8	6.1	1.9	Iris-Virginica	Iris-Virginica
6	5.5	2.4	3.7	1.0	Iris-Versicolour	Iris-Versicolour
7	6.7	3.1	4.7	1.5	Iris-Versicolour	Iris-Versicolour
8	6.7	3.1	4.4	1.4	Iris-Versicolour	Iris-Versicolour
9	5.4	3.4	1.7	0.2	Iris-Setosa	Iris-Setosa

Exercise 2: Optimize and Compare KNN algorithm

Part A: Determine Optimal Value of K in KNN algorithm:

1. How can you choose value of K for KNN. Give a criterion to choose an optimal value of K:

An optimal value of K can be chosen using k-fold cross validation on the train data. In this exercise, I've used a 10-fold cross validation on the test data for a range of 15 nearest neigbours. As a result of which, for every nearest neighbour, 10 cross validation accuracies were obtained and the mean of it is chosen. The value of K for which the mean cross validation accuracy is higher is chosen as the optimal value of K.

2. Implement the criterion for choosing the optimal value of K.

Function name : kfold
Parameter : Training data of input query and target and the number of folds

This function is used to choose the optimal value of K among the range of given nearest neighbours. Within this funciton, knn_ function is called to predict the class of the input query.

```
def kfold(X_train,Y_train,fold,k):
    split = math.ceil(len(X_train)/fold)
    fold_mcrate = np.zeros((fold,k))

    for i in range (0,fold):
        start = i*split
        end = i*split+split

        if start ==0:
            fold_Xtest = X_train[start:end]
            fold_Xtrain = X_train[end:]
            fold_Ytest = Y_train[start:end]
            fold_Ytrain = Y_train[end:]
            fold_Xtrainold = fold_Xtrain
        else:
            fold_Xtest = X_train[start:end]
            fold_Xtrain1 = X_train[0:start]
            fold_Xtrain2 = X_train[end:-1]
            fold_Xtrain = np.concatenate((fold_Xtrain1,fold_Xtrain2))
            fold_Ytest = Y_train[start:end]
            fold_Ytrain1 = Y_train[0:start]
            fold_Ytrain2 = Y_train[end:-1]
            fold_Ytrain = np.concatenate((fold_Ytrain1,fold_Ytrain2))

        fold_mcrate[i],pred = knn_(fold_Xtrain,fold_Xtest,fold_Ytrain,fold_Ytest,k)

    return fold_mcrate
```

```
fold_mcrate = kfold(X_train,y_train,10,k)
mcrate_cv = list()
for i in range(0,k):
    mcrate_cv.append(np.average(fold_mcrate[:,i]))
```

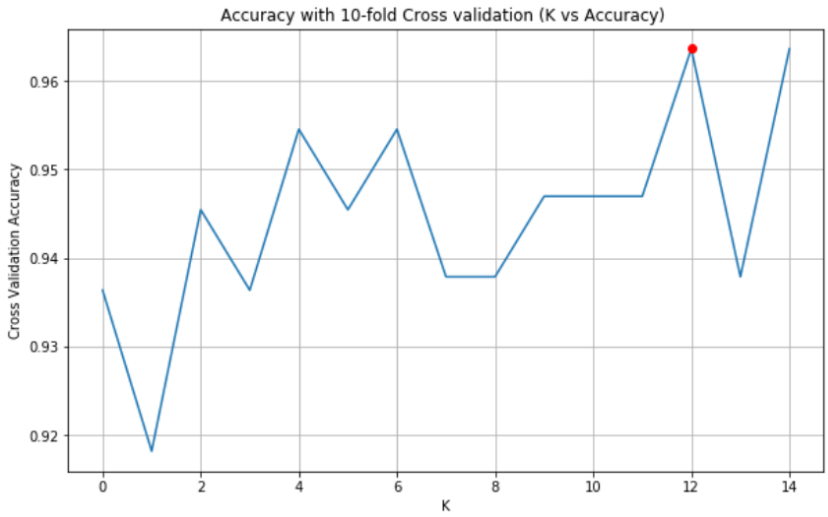
3.Experimental Results:

Iris data:

Mean Cross Validation Scores of 10-folds for a range of 15 nearest neighbours

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.936364	0.918182	0.945455	0.936364	0.954545	0.945455	0.954545	0.937879	0.937879	0.94697	0.94697	0.94697	0.963636	0.937879	0.963636

```
Optimal K:  13
Accuracy:  0.9636363636363636
```

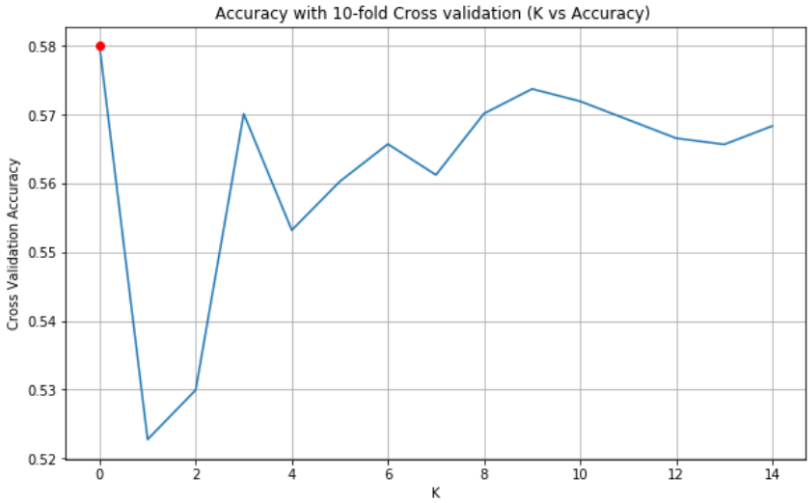


Wine data:

Mean Cross Validation Scores of 10-folds for a range of 15 nearest neighbours

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.579963	0.522756	0.529947	0.570125	0.553193	0.56028	0.565718	0.561229	0.570158	0.573737	0.571943	0.569257	0.566562	0.565669	0.568348

```
Optimal K:  1
Accuracy:  0.5799629987129987
```



PART B: Compare KNN algorithm with Tree based method:

Function name : knn_classifier

Parameter : Training and testing data of input query and target and the number of neighbours

```
def knn_classifier(X_train,X_test,y_train,y_test,k):
    k_scores = list()
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    return y_pred
```

Iris data:

```
K:  15
Accuracy of prediction:  0.9777777777777777
Number of misclassifications:  1.0
```

	sepal-length	sepal-width	petal-length	petal-width	class	Prediction Class
17	6.4	2.8	5.6	2.1	Iris-Virginica	Iris-Virginica
37	6.5	3.0	5.8	2.2	Iris-Virginica	Iris-Virginica
10	6.9	3.1	4.9	1.5	Iris-Versicolour	Iris-Versicolour
14	7.7	3.0	6.1	2.3	Iris-Virginica	Iris-Virginica
22	5.1	3.5	1.4	0.3	Iris-Setosa	Iris-Setosa
24	6.1	3.0	4.9	1.8	Iris-Virginica	Iris-Virginica
21	6.0	2.9	4.5	1.5	Iris-Versicolour	Iris-Versicolour
13	4.6	3.6	1.0	0.2	Iris-Setosa	Iris-Setosa
15	6.8	3.2	5.9	2.3	Iris-Virginica	Iris-Virginica
33	5.1	3.8	1.6	0.2	Iris-Setosa	Iris-Setosa

Wine data

```
K:  15
Accuracy of prediction:  0.6125
Number of misclassifications:  186.0
```

	volatile acidity	citric acid	density	sulphates	alcohol	quality	Prediction quality
4	0.65	0.02	0.99498	0.62	10.4	6	6
362	0.12	0.45	0.99552	0.76	11.9	7	6
437	0.37	0.52	0.99613	0.58	11.1	6	6
24	0.61	0.14	0.99690	0.59	9.5	5	5
252	0.49	0.36	0.99702	0.78	10.9	6	6
336	0.43	0.21	0.99660	0.91	9.5	5	5
136	0.47	0.00	0.99220	0.48	12.3	6	6
90	0.73	0.24	0.99670	0.59	9.3	5	5
61	0.54	0.26	0.99760	0.60	9.3	6	5
70	0.52	0.38	0.99666	0.52	9.4	5	5

Function name : DecTreeClassifier

Parameter : Training and testing data of input query and target and the number of neighbours

```
def DecTreeClassifier(X_train,X_test,y_train,y_test):

    classifier = DecisionTreeClassifier(random_state=0)
    classifier.fit(X_train,y_train)
    print(classifier.get_params)
    y_pred = classifier.predict(X_test)
    return y_pred
```

Iris data:

```
Accuracy of prediction:  0.9777777777777777
Number of misclassifications:  1.0
```

	sepal-length	sepal-width	petal-length	petal-width	class	Prediction Class
17	6.4	2.8	5.6	2.1	Iris-Virginica	Iris-Virginica
37	6.5	3.0	5.8	2.2	Iris-Virginica	Iris-Virginica
10	6.9	3.1	4.9	1.5	Iris-Versicolour	Iris-Versicolour
14	7.7	3.0	6.1	2.3	Iris-Virginica	Iris-Virginica
22	5.1	3.5	1.4	0.3	Iris-Setosa	Iris-Setosa
24	6.1	3.0	4.9	1.8	Iris-Virginica	Iris-Virginica
21	6.0	2.9	4.5	1.5	Iris-Versicolour	Iris-Versicolour
13	4.6	3.6	1.0	0.2	Iris-Setosa	Iris-Setosa
15	6.8	3.2	5.9	2.3	Iris-Virginica	Iris-Virginica
33	5.1	3.8	1.6	0.2	Iris-Setosa	Iris-Setosa

Wine data

```
Accuracy of prediction:  0.5916666666666667
Number of misclassifications:  196.0
```

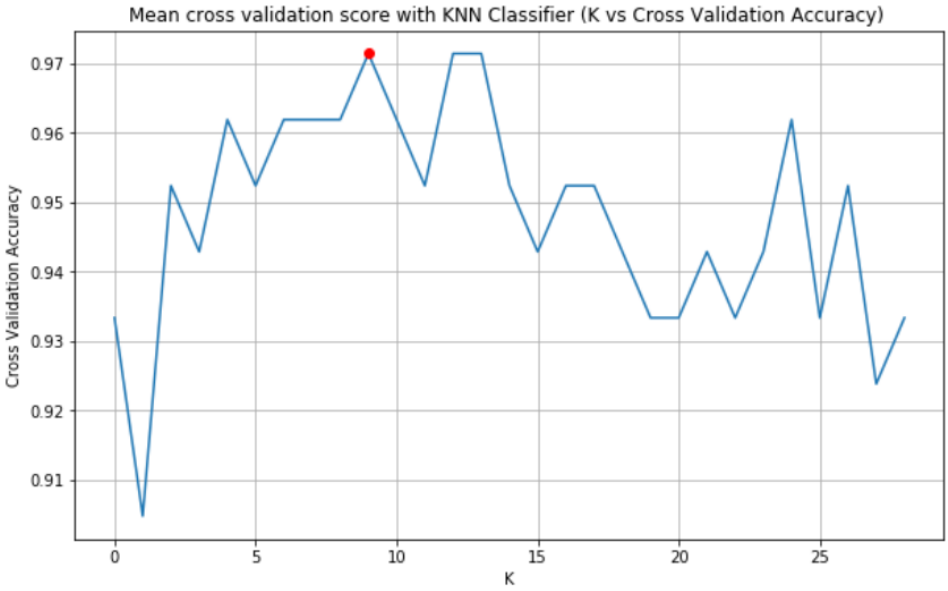
	volatile acidity	citric acid	density	sulphates	alcohol	quality	Prediction quality
34	0.61	0.01	0.99746	0.73	10.5	5	5
426	0.39	0.47	0.99820	0.75	9.8	6	6
238	0.34	0.52	1.00140	0.81	9.5	6	6
309	0.36	0.64	0.99800	0.66	12.5	6	6
78	0.52	0.38	0.99666	0.52	9.4	5	5
331	0.55	0.03	0.99560	0.48	9.0	4	4
106	0.54	0.04	0.99870	0.91	9.4	6	6
428	0.50	0.18	0.99761	0.72	9.6	6	6
203	0.55	0.15	0.99314	0.82	11.6	6	7
267	0.57	0.09	0.99417	0.74	12.7	8	6

Finding Optimal Hyperparameters

Iris data - KNN:

```
k_range = list(range(1, 30))
param_grid = dict(n_neighbors=k_range)
print(param_grid)
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')
grid.fit(X_train, y_train)
```

```
{'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]}
```

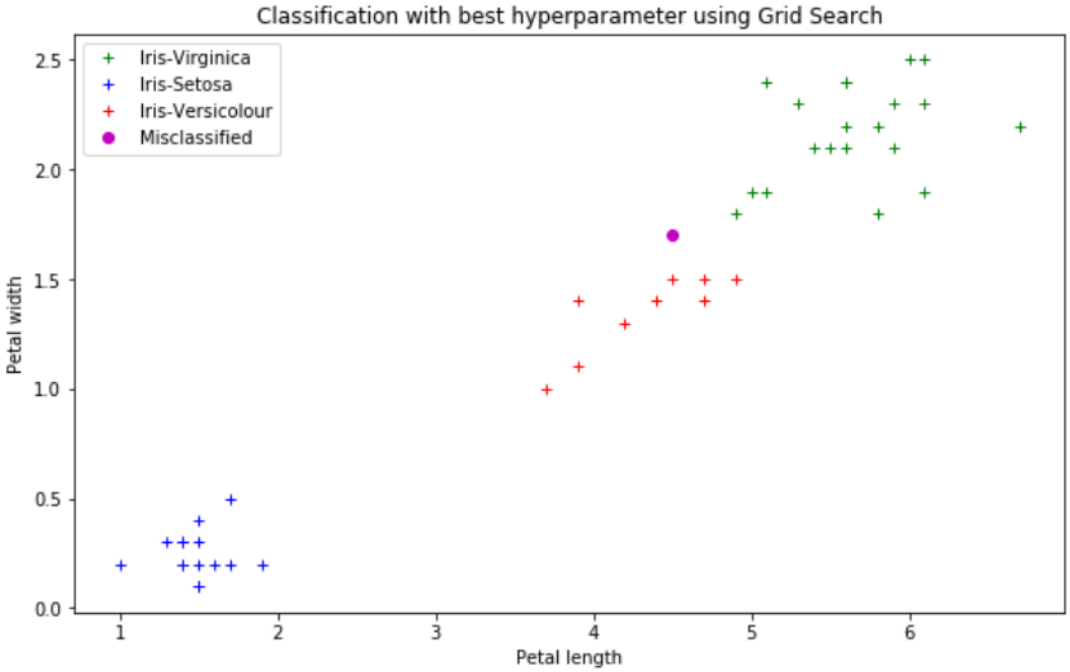


Predicting the test data with optimal hyperparameter

```
knn = KNeighborsClassifier(n_neighbors = grid.best_estimator_.n_neighbors)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
```

```
Optimal parameter:  10
Accuracy of prediction:  0.9777777777777777
Number of misclassifications:  1.0
```

	sepal-length	sepal-width	petal-length	petal-width	class	Prediction Class
5	7.4	2.8	6.1	1.9	Iris-Virginica	Iris-Virginica
0	7.7	3.8	6.7	2.2	Iris-Virginica	Iris-Virginica
28	5.0	3.6	1.4	0.2	Iris-Setosa	Iris-Setosa
27	5.4	3.4	1.5	0.4	Iris-Setosa	Iris-Setosa
11	6.7	3.1	5.6	2.4	Iris-Virginica	Iris-Virginica
34	4.8	3.4	1.9	0.2	Iris-Setosa	Iris-Setosa
36	5.6	2.5	3.9	1.1	Iris-Versicolour	Iris-Versicolour
20	4.8	3.0	1.4	0.3	Iris-Setosa	Iris-Setosa
30	5.4	3.7	1.5	0.2	Iris-Setosa	Iris-Setosa
42	5.2	2.7	3.9	1.4	Iris-Versicolour	Iris-Versicolour



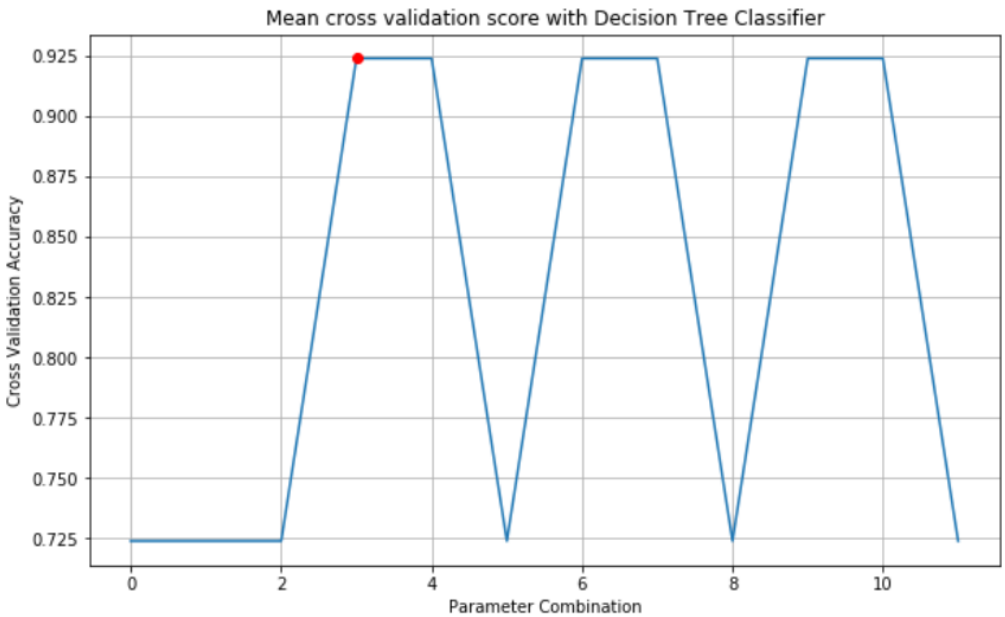
Iris data - Decision Tree Classifier:

```
min_samples_splits = np.linspace(0.1, 1.0, 3, endpoint=True)
max_depth = range(1,5)
param_grid = dict(min_samples_split=min_samples_splits,max_depth = max_depth)
print(param_grid)
```

Hyperparameters

```
{'min_samples_split': array([0.1 , 0.55, 1.  ]), 'max_depth': range(1, 5)}
```

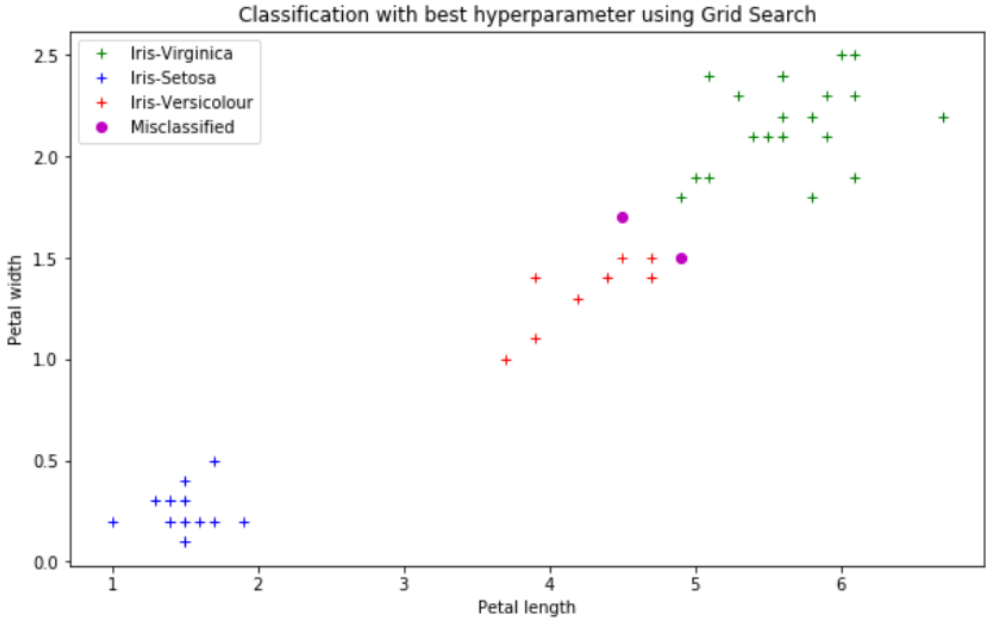
```
grid = GridSearchCV(DecisionTreeClassifier(), param_grid)
```



Prediction with optimal hyperameter found using Grid search

```
{'min_samples_split': 0.1, 'max_depth': 2}
Accuracy of prediction:  0.9555555555555556
Number of misclassifications:  2.0
```

	sepal-length	sepal-width	petal-length	petal-width	class	Prediction Class
24	6.1	3.0	4.9	1.8	Iris-Virginica	Iris-Virginica
0	7.7	3.8	6.7	2.2	Iris-Virginica	Iris-Virginica
23	5.8	2.8	5.1	2.4	Iris-Virginica	Iris-Virginica
2	6.3	2.5	5.0	1.9	Iris-Virginica	Iris-Virginica
20	4.8	3.0	1.4	0.3	Iris-Setosa	Iris-Setosa
25	4.9	3.1	1.5	0.1	Iris-Setosa	Iris-Setosa
19	6.4	2.8	5.6	2.2	Iris-Virginica	Iris-Virginica
36	5.6	2.5	3.9	1.1	Iris-Versicolour	Iris-Versicolour
14	7.7	3.0	6.1	2.3	Iris-Virginica	Iris-Virginica
6	5.5	2.4	3.7	1.0	Iris-Versicolour	Iris-Versicolour



Comparision:

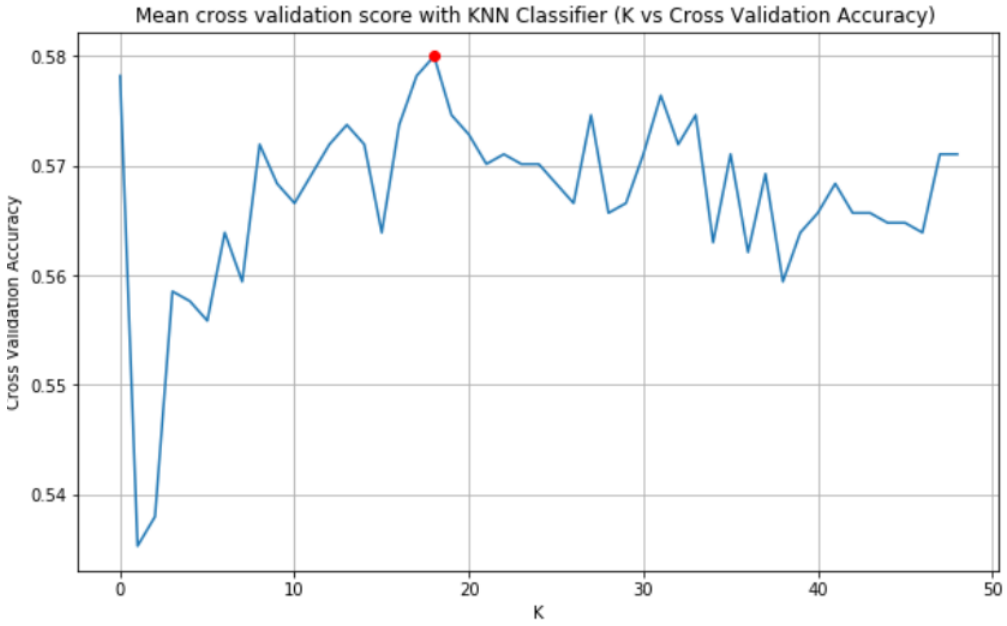
For the iris data classification, comparing KNN classifier with decision tree classifier, it's obvious that the KNN outperforms Decision tree method with an overall accuracy of 97.7% whereas; it is 95.5% in the case of decision tree method. Also the number of misclassified points is more in decision tree method.

	KNN	Decision Tree
Accuracy	97.70 %	95.56%
No. of misclassification	1	2
Misclassification rate	0.022	0.0909

Wine data - KNN:

Hyperparameter:

```
{'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]}
```



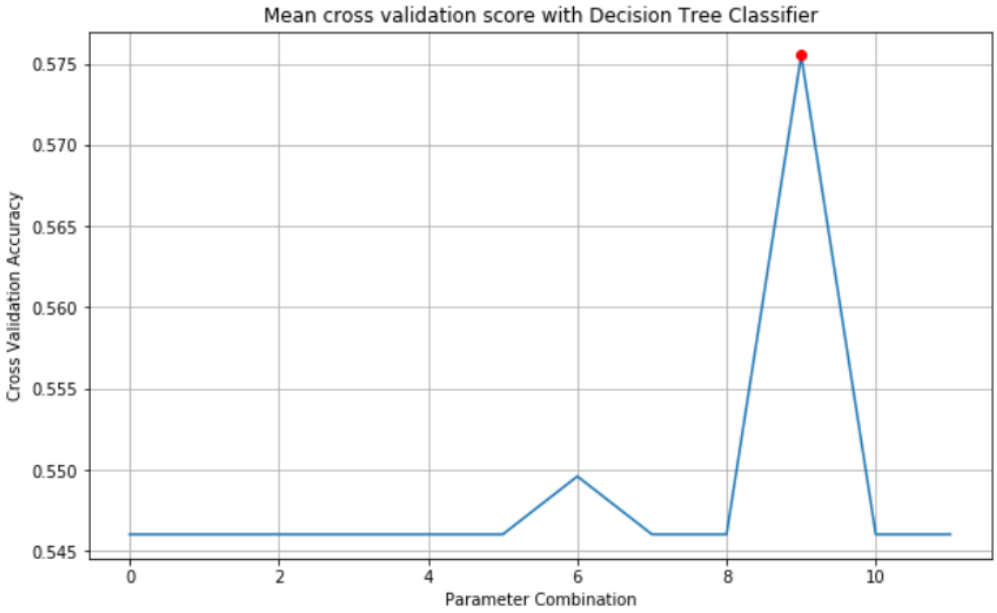

```
K:  19
Accuracy of prediction:  0.6145833333333334
Number of misclassifications:  185.0
```

	volatile acidity	citric acid	density	sulphates	alcohol	quality	Prediction quality
476	0.65	0.02	0.99498	0.62	10.4	6	6
178	0.31	0.40	0.99536	0.68	11.2	7	7
433	0.65	0.37	1.00260	0.64	10.4	6	6
169	0.43	0.25	0.99615	0.58	10.5	6	6
59	0.58	0.00	0.99562	0.58	11.3	6	6
132	0.47	0.27	0.99518	0.85	11.1	6	6
269	0.60	0.01	0.99514	0.61	10.9	6	6
91	0.45	0.36	0.99780	0.83	10.5	5	6
276	0.50	0.11	0.99545	0.79	9.5	5	5
206	0.35	0.47	0.99585	0.52	12.0	6	6

Wine data – Decision Tree Classifier:

Hyperparameters

```
{'min_samples_split': array([0.1 , 0.55, 1.  ]), 'max_depth': range(1, 5)}
```




```
{'min_samples_split': 0.1, 'max_depth': 4}
Accuracy of prediction:  0.5791666666666667
Number of misclassifications:  202.0
```

	volatile acidity	citric acid	density	sulphates	alcohol	quality	Prediction quality
438	0.630	0.02	0.99712	0.75	9.8	5	5
67	0.280	0.28	0.99064	0.39	11.7	7	6
296	0.640	0.23	0.99980	0.59	9.7	5	5
270	0.580	0.20	0.99322	0.49	11.7	5	6
420	0.480	0.24	1.00000	0.56	10.0	6	5
245	0.795	0.00	0.99378	0.52	11.6	5	6
397	0.670	0.55	0.99680	0.62	9.4	5	5
479	0.590	0.49	0.99910	0.56	9.6	4	5
65	0.280	0.47	0.99686	0.67	10.6	7	6
386	0.180	0.34	0.99470	0.78	11.8	6	7

Comparision:

For the wine data classification, comparing KNN classifier with decision tree classifier, it can be seen that the KNN outperforms Decision tree method with an overall accuracy of 61.45% whereas; it is 57.91% in the case of decision tree method. Also the number of misclassified points is more in decision tree method.

	KNN	Decision Tree
Accuracy	61.45 %	57.91%
No. of misclassification	185	202
Misclassification rate	0.358	0.421