Machine Learning HW #1 answer

1.

(1) Neural networks (particularly CNN) requires large dataset to train and take longer to train.

(2) The C4.5 decision tree is easy to make the branches grow more and more complicated during training, resulting in overfitting.

(3) Adaboost is sensitive to noise, i.e., labeling errors on the training set.

2.

a.

- Classification model is the task of predicting a discrete class label.
- Regression model is the task of predicting a continuous quantity.

We use regression techniques to perform rent prediction.

b.

Unsupervised-learning algorithm is for unlabeled data, so it's not suitable for this case.

3.

$$\mathrm{J} = \sum_{i=1}^{10} \left(y_i - (ax_i + b)\right)^2$$

$$= \sum_{i=1}^{10} ({y_i}^2 - 2y_i(ax_i + b) + (ax_i + b)^2)$$

$$= \sum_{i=1}^{10} ({y_i}^2 - 2ax_iy_i - 2by_i + (ax_i)^2 + 2abx_i + b^2)$$

$$\frac{\partial \mathrm{J}}{\partial \mathrm{b}} = \sum_{i=1}^{10} (-2y_i + 2ax_i + 2b) = 0$$

$$\Rightarrow b = \bar{y} - a\bar{x}$$

$$\frac{\partial \mathrm{J}}{\partial \mathrm{a}} = \sum_{i=1}^{10} (-2x_iy_i + 2a{x_i}^2 + 2bx_i) = 0$$

$$= \sum_{i=1}^{10} (-2x_iy_i + 2a{x_i}^2 + 2(\bar{y} - a\bar{x})x_i)$$

$$= \sum_{i=1}^{10} (-2x_iy_i + 2a{x_i}^2 + 2\bar{y}x_i - 2a\bar{x}x_i) = 0$$

$$\Rightarrow \sum_{i=1}^{10} (a{x_i}^2 - a\bar{x}x_i) = \sum_{i=1}^{10} (x_iy_i - \bar{y}x_i)$$

$$\Rightarrow \mathrm{a} = \frac{\sum_{i=1}^{10}(x_iy_i - \bar{y}x_i)}{\sum_{i=1}^{10}({x_i}^2 - \bar{x}x_i)} = \frac{\sum_{i=1}^{10}(y_i - \bar{y})\,x_i}{\sum_{i=1}^{10}(x_i - \bar{x})\,x_i}$$

$$b = \bar{y} - \frac{\sum_{i=1}^{10}(y_i - \bar{y})\,x_i}{\sum_{i=1}^{10}(x_i - \bar{x})\,x_i}\bar{x}$$

4.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

iris_dataset = datasets.load_iris()

df_X = iris_dataset.data[:, :4]
df_y = iris_dataset.target
```

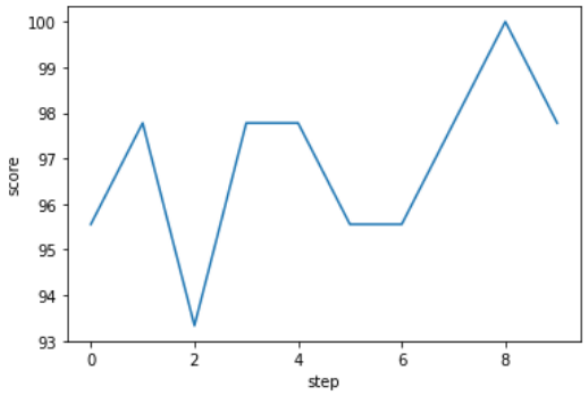
```
scores = []

for i in range(10):
        X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.3)

        classifier = KNeighborsClassifier(n_neighbors=7)
        classifier.fit(X_train, y_train)
        scores.append(round(accuracy_score(classifier.predict(X_test), y_test) * 100, 3))

plt.plot(scores)
plt.xlabel('step')
plt.ylabel('score')
plt.show()

print(f"Score = ", scores)
print(f"Score average: {np.mean(scores)}")
```




```
Score =  [95.556, 97.778, 93.333, 97.778, 97.778, 95.556, 95.556, 97.778, 100.0, 97.778]
Score average: 96.88910000000001
```

5.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import operator

iris_dataset = datasets.load_iris()
df_X = iris_dataset.data[:, :4]
df_y = iris_dataset.target
```

```
train_size = int(len(df_X)*0.6)
validate_size = int(len(df_X)*0.2)
test_size = len(df_X) - train_size - validate_size
train_X = np.empty((10, train_size, 4))
test_X = np.empty((10, test_size, 4))
train_y = np.empty((10, train_size))
test_y = np.empty((10, test_size))
k_scores = {3:[], 4:[], 5:[], 6:[], 7:[], 8:[], 9:[], 10:[], 11:[]}
scores = {3:[], 4:[], 5:[], 6:[], 7:[], 8:[], 9:[], 10:[], 11:[]}
```

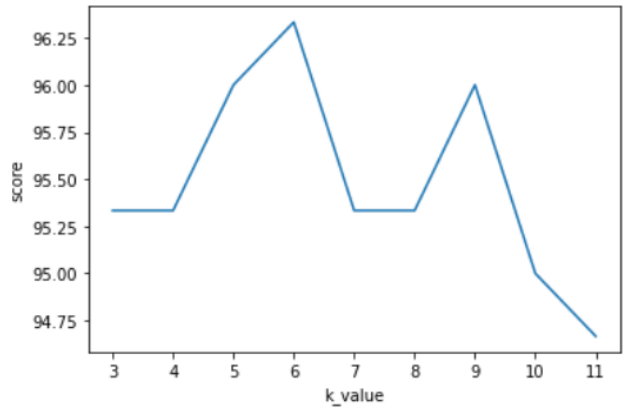
```
for i in range(10):
        X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.2, stratify=df_y)
        X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, stratify=y_train)
        train_X[i,:,:] = X_train
        test_X[i,:,:] = X_test
        train_y[i,:] = y_train
        test_y[i,:] = y_test
        for k in range(3, 12):
            classifier = KNeighborsClassifier(n_neighbors=k)
            classifier.fit(X_train, y_train)
            k_scores[k].append(accuracy_score(classifier.predict(X_val), y_val) * 100)

for k in range(3, 12):
    scores[k] = np.mean(k_scores[k])

plt.plot(list(scores.keys()), list(scores.values()))
plt.xlabel('k_value')
plt.ylabel('score')
plt.show()
```

```
best_k = (max(scores.items(), key=operator.itemgetter(1))[0])
score_list = []
for i in range(10):
    classifier = KNeighborsClassifier(n_neighbors=best_k)
    classifier.fit(train_X[i,:,:], train_y[i,:])
    score_list.append(round(accuracy_score(classifier.predict(test_X[i,:,:]), test_y[i,:]) * 100, 3))

print(f"The best K: ", (best_k))
print(f"Score=", score_list)
print(f"Average=", round(np.mean(score_list), 3))
```




```
The best K:  6
Score= [100.0, 100.0, 100.0, 100.0, 100.0, 96.667, 96.667, 96.667, 96.667, 100.0]
Average= 98.667
```