In [100]:
```python
import numpy as np
import sklearn
from sklearn import model_selection
import pandas as pd
import sys
import matplotlib.pyplot as plt
```

In [101]:
```python
disease_data= pd.read_csv(r'D:\Google_Download\Artificial Intelligence\Assignment\assignment 2-supp.csv')
```

In [102]:
```python
disease_data.head()
```

Out[102]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [103]:
```python
disease_data.shape
```

Out[103]: (768, 9)

In [104]:
```python
train, test = model_selection.train_test_split(disease_data, test_size=0.2, random_state=1234)
```

In [108]:
```python
real_train, validation=model_selection.train_test_split(train, test_size=0.25, random_state=1234)
```

In [105]:
```python
train.shape
```

Out[105]: (614, 9)

In [106]:
```python
test.shape
```

Out[106]: (154, 9)

In [109]:
```python
real_train.shape
```

Out[109]: (460, 9)

In [110]:
```python
validation.shape
```

Out[110]: (154, 9)

In [111]:
```python
real_train_y=real_train.Outcome
real_train_x=real_train.drop("Outcome", axis=1)
validation_y=validation.Outcome
validation_x=validation.drop("Outcome", axis=1)
test_y=test.Outcome
test_x=test.drop("Outcome", axis=1)
```

In [128]:
```python
class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=10000, fit_intercept=True, verbose=True):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose=verbose
        self.loss_value=[]

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        # add the bias for X
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
    def __loss(self, h, y):
        # Loss function-> 1/m*(-y * np.log(h(theta' * x)) - (1 - y) * np.log(1 - h(theta' * x)))
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)
        # weights initialization -> can use all 0 initialization, but here I use random initialization to eliminate the coincidence
        #self.theta = np.random.randn(X.shape[1])
        self.theta=np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            tmp=self.__loss(h, y)
            if tmp!=float('inf'):
                self.loss_value.append(self.__loss(h, y))      # append the loss value to see whether it reached overfit
            # Batch gradient discent
            # Calculate the gradient:
            gradient = np.dot(X.T, (h - y)) / y.size
            self.theta -= self.lr * gradient
            if(self.verbose == True and i % self.num_iter == 0):
                # Print the final loss
                z = np.dot(X, self.theta)
                h = self.__sigmoid(z)
                print(f'loss: {self.__loss(h, y)} \t')

    def predict_prob(self, X):
        # probability calculation function, get the final theta value, and predict the probability of each input vector
        if self.fit_intercept:
            X = self.__add_intercept(X)
        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X, threshold):
        # Predict function: if the p is larger then threshold, then predict it to be 1. Otherwise 0.
        return self.predict_prob(X) >= threshold
```

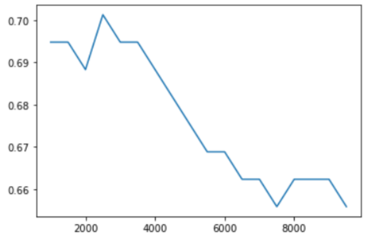
In [129]:
```
model = LogisticRegression(lr=0.001, num_iter=10000)
%time model.fit(real_train_x, real_train_y)
```

```
loss: 1.0648685585737285
Wall time: 13.1 s
```

In [135]:
```
accu_result=[]
for iteration in range(1000,10000,500):
    model = LogisticRegression(lr=0.001, num_iter=iteration)
    model.fit(real_train_x, real_train_y)
    preds = model.predict(validation_x,0.94)
    # accuracy
    accu_result.append((preds == validation_y).mean())
```

```
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
loss: 1.0648685585737285
```

In [136]:
```
plt.plot(np.arange(1000,10000,500),accu_result)
plt.show()     # ensure the iteration number = 7500
```



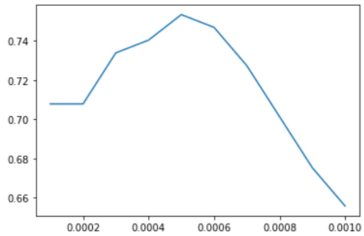
In [141]:
```
np.linspace(0.0001,0.001,10)
```

Out[141]:
```
array([0.0001,  0.0002,  0.0003,  0.0004,  0.0005,  0.0006,  0.0007,  0.0008,
       0.0009,  0.001 ])
```

In [142]:
```
acu_result_learing_rate=[]
for l_rate in np.linspace(0.0001,0.001,10):
    model = LogisticRegression(lr=l_rate, num_iter=7500)
    model.fit(real_train_x, real_train_y)
    preds = model.predict(validation_x,0.94)
    # accuracy
    acu_result_learing_rate.append((preds == validation_y).mean())
```

```
loss: 0.6793210937379046
loss: 0.6798812137912853
loss: 0.6940311656030363
loss: 0.7206036837364072
loss: 0.7582280054685907
loss: 0.8054757003353221
loss: 0.8609648903117317
loss: 0.9234232503073785
loss: 0.9917187773612244
loss: 1.0648685585737285
```

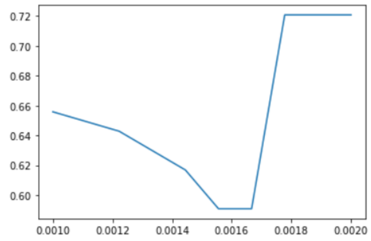
In [143]:
```
plt.plot(np.linspace(0.0001,0.001,10),acu_result_learing_rate)
plt.show()     # ensure the iteration number = 7500
```



In [149]:
```
acu_result_learing_rate=[]
for l_rate in np.linspace(0.001,0.002,10):
    model = LogisticRegression(lr=l_rate, num_iter=7500)
    model.fit(real_train_x, real_train_y)
    preds = model.predict(validation_x,0.94)
    # accuracy
    acu_result_learing_rate.append((preds == validation_y).mean())
```

```
loss: 1.0648685585737285
loss: 1.1508240129020302
```

In [150]:
```
plt.plot(np.linspace(0.001,0.002,10),acu_result_learing_rate)
plt.show()    # ensure the iteration number = 7500
```



In [144]:
```
train_y=train.Outcome
train_x=train.drop("Outcome",axis=1)
```

In [158]:
```
model = LogisticRegression(lr=0.0016, num_iter=7500)
model.fit(train_x,train_y)
```

```
loss: 1.7242433603491103
```

```
<ipython-input-128-bc9fbba4b6c8>:18: RuntimeWarning: divide by zero encountered in log
  return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

In [173]:
```
preds = model.predict(test_x,0.95)
# accuracy
(preds == test_y).mean()
```

Out[173]:
```
0.6233766233766234
```

In [169]:
```
model.theta
```

Out[169]:
```
array([-0.43852211,  0.73429886,  0.04561872, -0.17067624, -0.05802802,
       -0.01903002,  0.0093603 ,  0.09285052, -0.09620783])
```

In [ ]: