hw2_programming_base_notebook

June 22, 2024

[83]:
```
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import time
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
```

1 Problem 1

1.1 Dataset Generation

Write a function to **generate a training set** of size m - randomly generate a weight vector $w \in \mathbb{R}^{10}$, normalize length - generate a training set $\{(x_i, y_i)\}$ of size m - x_i: random vector in $\mathbb{R}^{10}$ from $\mathbf{N}(0, I)$ - y_i: $\{0, +1\}$ with $P[y = +1] = \sigma(w \cdot x_i)$ and $P[y = 0] = 1 - \sigma(w \cdot x_i)$

[84]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def generate_data(m):
    # returns the true w as well as X, Y data
    w_star = np.random.normal(0,1,10)
    #unitify vector
    w_star = w_star / LA.norm(w_star)
    X = np.random.normal(0,1,(m, 10))
    Y = np.array([1 if np.random.uniform(0, 1) <= sigmoid(np.dot(w_star, X[i])) ␣
 ↪else 0 for i in range(m)])
    return X, Y, w_star
# print(Y.shape)
# print(X.shape)
# X, Y = generate_data(100)
```

1.2 Algorithm 1: logistic regression

The goal is to learn w. Algorithm 1 is logistic regression (you may use the built-in method LogisticRegression for this. Use max_iter=1000).

[85]:
```
def train_logistic_regression(X,Y):
    model = LogisticRegression()
    model.fit(X, Y)
    return model.coef_[0]
```

1.3 Algorithm 2: gradient descent with square loss

Define square loss as

$$L_i(w^{(t)}) = \frac{1}{2}\left(\sigma(w^{(t)} \cdot x) - y_i\right)^2$$

Algorithm 2 is gradient descent with respect to square loss (code this up yourself – run for 1000 iterations, use step size eta = 0.01).

[86]:
```
def train_gradient_descent(X):
    #generate vector w
    m, d = X.shape
    w = np.random.randn(d)
    for i in np.arange(0,1000):
        prediction = sigmoid(np.dot(X,w))
        error = Y - prediction
        loss = error * error * 1/2
        # gradient = np.gradient(loss)
        gradient = np.dot(X.T, error) / m
        w = w - gradient * 0.01
    return w
print(w.shape)
```

```
(10,)
```

1.4 Algorithm 3: stochastic gradient descent with square loss

Similar to gradient descent, except we use the gradient at a single random training point every iteration.

[87]:
```
def train_stochastic_gradient_descent(X):
    m, d = X.shape
    w = np.random.randn(d)
    for i in np.arange(0,1000):
        index = np.random.randint(m)
        x = X[index]
        y = Y[index]
        prediction = sigmoid(np.dot(x,w))
        error = y - prediction
        loss = error * error * 1/2
        gradient = np.dot(x.T, error) / m
        w = w - gradient * 0.01
    return w
print(w.shape)
```

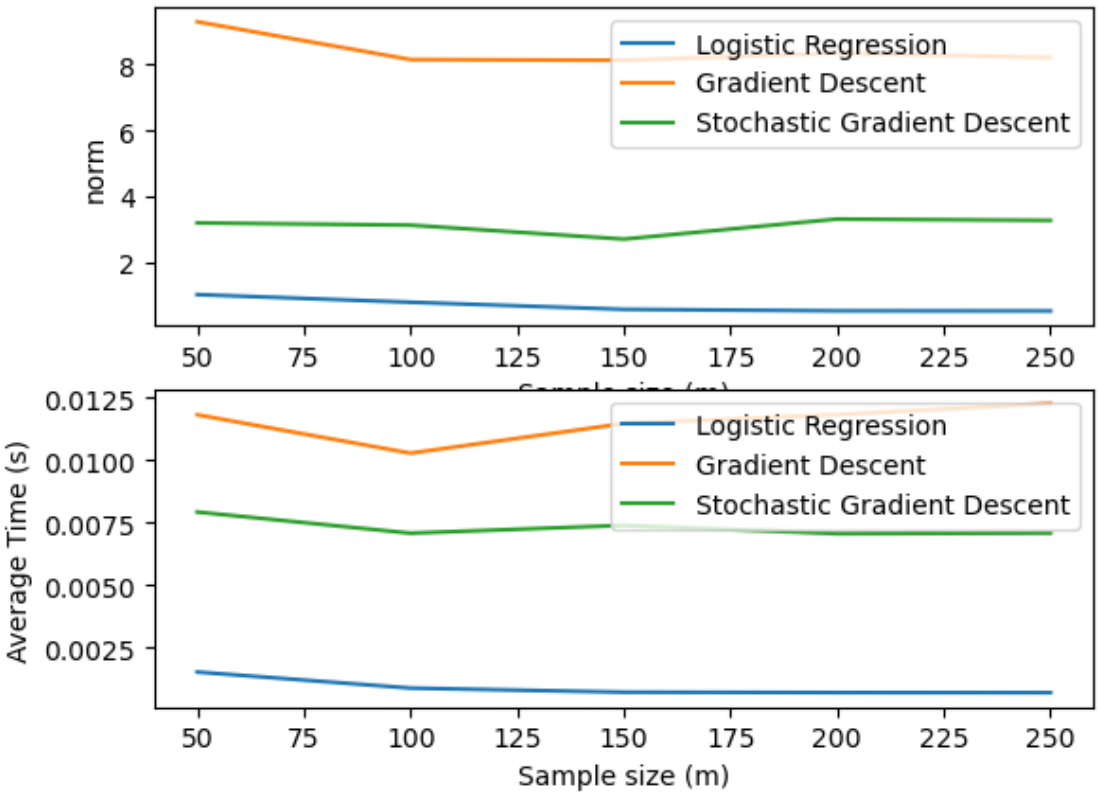
(10,)

1.5 Evaluation

Measure error $\|w - \hat{w}\|_2$ for each method at different sample size. For any fixed value of m, choose many different w's and average the values $\|w - \hat{w}\|_2$ for Algorithms 1, 2 and 3. Plot the results for for each algorithm as you make m large (use $m = 50, 100, 150, 200, 250$). Also record, for each algorithm, the time taken to run the overall experiment.

[88]:
```
m = [50,100,150,200,250]
results = {
    "logisticRegression": [],
    "gradientDecent": [],
    "stochasticGradientDecent": [],
    "time_logistic": [],
    "time_gd": [],
    "time_sgd": []
}
for i in m:
    dist_logistic, dist_gd, dist_sgd = 0, 0, 0
    time_logistic, time_gd, time_sgd = 0, 0, 0
    for _ in range(10):
        X,Y,w_star = generate_data(i)
        start = time.time()
        w_logistic = train_logistic_regression(X, Y)
        time_logistic += time.time() - start
        dist_logistic += LA.norm(w_star - w_logistic)

        # Train gradient descent and measure time
        start = time.time()
        w_gd = train_gradient_descent(X)
        time_gd += time.time() - start
        dist_gd += LA.norm(w_star - w_gd)

        # Train stochastic gradient descent and measure time
        start = time.time()
        w_sgd = train_stochastic_gradient_descent(X)
        time_sgd += time.time() - start
        dist_sgd += LA.norm(w_star - w_sgd)

    results["logisticRegression"].append(dist_logistic / 10)
    results["gradientDecent"].append(dist_gd / 10)
    results["stochasticGradientDecent"].append(dist_sgd / 10)
    results["time_logistic"].append(time_logistic / 10)
    results["time_gd"].append(time_gd / 10)
    results["time_sgd"].append(time_sgd / 10)

plt.subplot(2, 1, 1)
```

```
plt.plot(m, results["logisticRegression"], label='Logistic Regression')
plt.plot(m, results["gradientDecent"], label='Gradient Descent')
plt.plot(m, results["stochasticGradientDecent"], label='Stochastic Gradient␣
 ↪Descent')
plt.xlabel('Sample size (m)')
plt.ylabel('norm')
plt.legend(loc='upper right')

# Plot average time taken by each algorithm
plt.subplot(2, 1, 2)
plt.plot(m, results["time_logistic"], label='Logistic Regression')
plt.plot(m, results["time_gd"], label='Gradient Descent')
plt.plot(m, results["time_sgd"], label='Stochastic Gradient Descent')
plt.xlabel('Sample size (m)')
plt.ylabel('Average Time (s)')
plt.legend(loc='upper right')
plt.figure(figsize=(12,8))
plt.show()
```




```
<Figure size 1200x800 with 0 Axes>
```

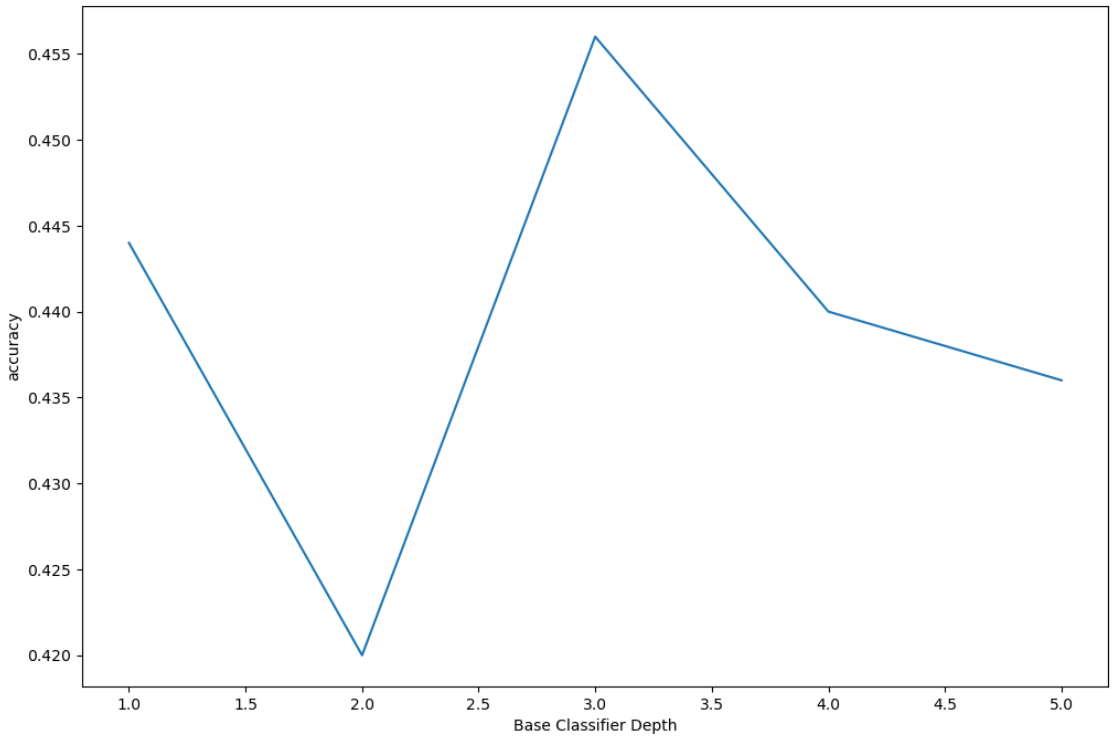
2 Problem 2

[89]:
```
from sklearn import datasets
```

[90]:
```
cancer = datasets.load_breast_cancer()
```

For each depth in $1, \ldots, 5$, instantiate an AdaBoost classifier with the base learner set to be a decision tree of that depth (set `n_estimators=10` and `learning_rate=1`), and then record the 10-fold cross-validated error on the entire breast cancer data set. Plot the resulting curve of accuracy against base classifier depth. Use 101 as your random state for both the base learner as well as the AdaBoost classifier every time.

[92]:
```
depths = [1,2,3,4,5]
errors = []
for depth in depths:
    base_estimator = DecisionTreeClassifier(max_depth=depth, random_state=101)
    ada_boost = AdaBoostClassifier(estimator=base_estimator, n_estimators=10,␣
 ↪learning_rate=1, random_state=101)
    scores = cross_val_score(ada_boost, X, Y, cv=10)
    errors.append(1 - scores.mean())


plt.figure(figsize=(12,8))
plt.plot(depths, errors)
plt.xlabel('Base Classifier Depth')
plt.ylabel('accuracy')
plt.show()
```


0.455
0.450
0.445
0.440
0.435
0.430
0.425
0.420
accuracy
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Base Classifier Depth