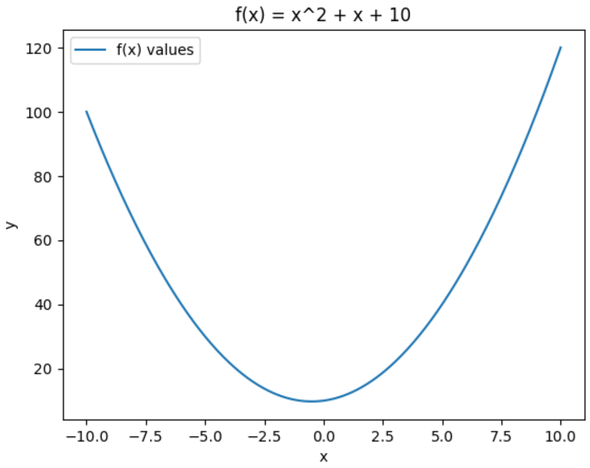
Q1

Q1.1.

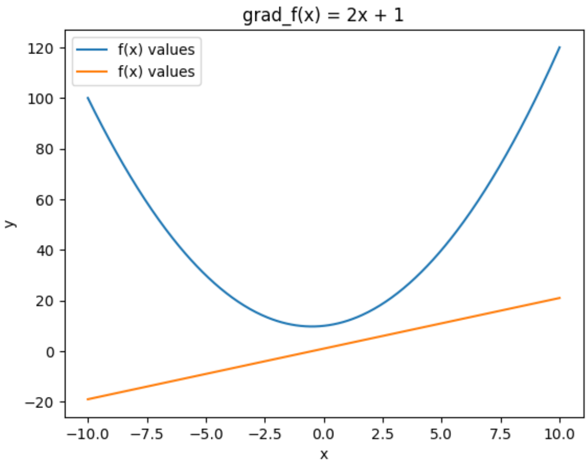
[1]:
```python
import matplotlib.pyplot as plt
import numpy as np

class F:
    const = [10,1,1] #a,b,c

    # calc f(x)
    @staticmethod
    def f(x):
        y = 0
        for i, const in enumerate(F.const):
            y += (x**i)*const
        return y

    # calc gardient of f(x)
    @staticmethod
    def grad_f(x):
        y = 0
        for i in range(1, len(F.const)):
            y += i * F.const[i] * (x ** (i - 1))
        return y
```

[2]:
```python
f_x = np.linspace(-10, 10, 400)
f_y = F.f(f_x)

plt.plot(f_x, f_y, '-', label='f(x) values')
plt.xlabel('x')
plt.ylabel('y')
plt.title('f(x) = x^2 + x + 10')
plt.legend()
plt.show()
```



Q1.2.

```
[3]: f_x = np.linspace(-10, 10, 400)
f_y = F.f(f_x)

grad_f_x = np.linspace(-10, 10, 400)
grad_f_y = F.grad_f(grad_f_x)

plt.plot(f_x, f_y, '-', label='f(x) values')
plt.plot(grad_f_x, grad_f_y, '-', label='f(x) values')

plt.xlabel('x')
plt.ylabel('y')
plt.title('grad_f(x) = 2x + 1')
plt.legend()
plt.show()
```



Q1.3.

min(f(x)): grad_f(x) = 2x+1 = 0 -> x = -0.5, y = 9.75

Q1.4.

```
[4]: def GD(init_x, epsilon, n, save_x):
    x = []
    x_t = float('inf') # x_(t)
    x_T = init_x  # x_(t+1)
    t = 0  # Iteration counter


    while abs(x_t - x_T) > epsilon:
        x_t = x_T
        x_T -= n * F.grad_f(x_T)
        if save_x: x.append(x_T)
        t += 1
        if t > 1000:  # Prevent infinite loops by setting a maximum number of iterations
            print("Maximum iterations reached")
            break

    return x_T, t, x
```

Q1.5.

```
[5]: # Running the gradient descent algorithm
init_x = 10
epsilon = 0.00001
n = 0.1 # Learning rate

result, t, x = GD(init_x, epsilon, n, False)
print(f"The minimum is aproxemtry at x = {result}, calc toke t = {t} iterations")
```

```
The minimum is aproxemtry at x = -0.4999607148359886, calc toke t = 56 iterations
```

the value we got is not equal to the minimum of f(x) because of the epsilon we defines as 0.00001

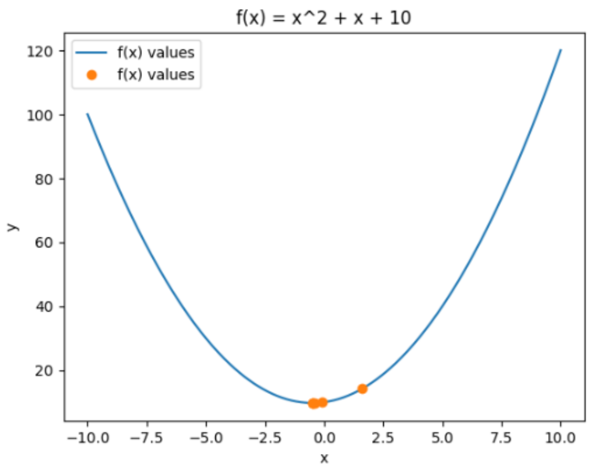
Q1.6.

[6]:
```
# Running the gradient descent algorithm
init_x = 10
epsilon = 0.01
n = 0.4 # Learning rate

result, t, x = GD(init_x, epsilon, n, True)
print(f"The minimum is aproxemtry at x = {result}, calc toke t = {t} iterations")
```

```
The minimum is aproxemtry at x = -0.499328, calc toke t = 6 iterations
```

Q1.7.

[7]:
```
f_x = np.linspace(-10, 10, 400)
f_y = F.f(f_x)


x = np.array(x)
y = F.f(x)


plt.plot(f_x, f_y, '-', label='f(x) values')
plt.plot(x, y, 'o', label='f(x) values')
plt.xlabel('x')
plt.ylabel('y')
plt.title('f(x) = x^2 + x + 10')
plt.legend()
plt.show()
```



Q2

Q2.1.

target func is: SVG(w,x,y)=$arg\text{-}min\_w\in R\text{^}d,\ b\in R$*∑($max${0, 1 − $yi(\langle w, xi\rangle + b)$} + λ||w||^2), we will break it down to a few func:

f(x) = $arg\text{-}min$(x) - convex func from thyrom we learnd in TA

g(x) = ∑(x) - convex func from thyrom we learnd in TA

u(x,y) = max{x,y} - convex func from thyrom we learnd in TA

h(w,x) = $\langle w, xi\rangle$ - convex func from thyrom we learnd in TA

k(W) = ||w||^2 - norm -> convex func from thyrom we learnd in TA and lemma from HW file

from here: SGD(w,x,y) = f(g(u(x, 1-y_i(h(w,x_i)+b)+λk(w)) -> sum of convex funcs** is convex

** liniar comb of convex and scalr multi of convex is convex

Q2.2

נראה שפונק' f תהיה ρ - ליפשיץ כיוון
ש w - צריך להתקיים:

$$\|f(w_1) - f(w_2)\| \le \rho \|w_1 - w_2\|$$

כאשר $\rho \cdot \max_h \|x_h\|$

נחלק למקרים

א. $l_1(w_1, x, y), l_2(w_2, x, y) > 0 \rightarrow$

$$\|l_1 - l_2\| = \|y_i\langle w_2, x_i\rangle - y_i\langle w_1, x_i\rangle\| =$$

$$\|y_i(\langle w_2, x_i\rangle - \langle w_1, x_i\rangle)\| \underset{y_i=1,\, y_i=-1}{=} \|(\langle w_2, x_i\rangle - \langle w_1, x_i\rangle)\| =$$

$$\|\langle w_2 - w_1, x_i\rangle\| \le \|x_i\| \cdot \|w_2 - w_1\| =$$

$$\|x_i\| \cdot \|w_1 - w_2\| \le \max_h \|x_h\| \cdot \|w_1 - w_2\| =$$

$$\rho \cdot \|w_1 - w_2\|$$

סיום.

$\leftarrow \quad \ell_1(w_1, x, y), \ell_2(w_2, x, y) > 0$ [illegible]

$\|\ell_1 - \ell_2\| = \|y_i \langle w_2, x_i \rangle - y_i \langle w_1, x_i \rangle\| =$

$\|y_i(\langle w_2, x_i \rangle - \langle w_1, x_i \rangle)\| = \|(\langle w_2, x_i \rangle - \langle w_1, x_i \rangle)\| =$
$\quad y_i = 1, y_i = -1$

$\|\langle w_2 - w_1, x_i \rangle\| \leq \|x_i\| \cdot \|w_2 - w_1\| =$

$\|x_i\| \cdot \|w_1 - w_2\| \leq \max_k \|x_k\| \cdot \|w_1 - w_2\| =$

[illegible]

$R \cdot \|w_1 - w_2\|$

$\leftarrow \quad \ell_1(w_1, x, y), \ell_2(w_2, x, y) = 0$ [illegible]

$\|\ell_1 - \ell_2\| = 0 - 0 \leq \underset{\geq 0}{R} \cdot \underset{\geq 0}{\|w_1 - w_2\|}$

$\leftarrow$ [illegible] $\ell_1(w_1, x, y) > 0 \quad , \ell_2(w_2, x, y) = 0$ [illegible]

$\|\ell_1 - \ell_2\| = 1 - y_i \langle w_1, x_i \rangle - 0 \overset{(*)}{=} y_i \langle w_2, x_i \rangle - y_i \langle w_1, x_i \rangle$

$\|\ell_1 - \ell_2\| \leq R \|w_1 - w_2\|$ [illegible]

$(*) \quad \ell_2(w_2, x, y) = 1 - y_i \langle w_2, x, y \rangle = 0 \rightarrow$

$y_i \langle w_2, x, y \rangle =$

Q2.3.

As we have seen in the tutorial:

(d/dw)SGD(w,x,y) = {2λw *if 1 − y_i(⟨w, x_i⟩ + b) ≤ 0*, −y_ix_i + 2λw *if 1 − y_i(⟨w, x_i⟩ + b) > 0*}

(d/db)SGD(w,x,y) = {0 *if 1 − y_i(⟨w, x_i⟩ + b) ≤ 0*, −y_i *if 1 − y_i(⟨w, x_i⟩ + b) > 0*}

Q2.4.

[8]:
```
def svm_with_sgd(X, y, lam=0, epochs=1000, l_rate=0.01, sgd_type='practical'):
    np.random.seed(2)
    m, d = X.shape
    b = np.random.uniform(0, 1, 1)
    w = np.random.uniform(0, 1, d)

    def SG(w, x, y, b, lam):
        if 1 - y * (w.dot(x) + b) <= 0:
            return (2 * lam * w, 0)
        else:
            return (-y * x + 2 * lam * w, -y)

    if sgd_type == 'practical':
        for j in range(epochs):
            per = np.random.permutation(np.arange(0, m))
            for i in range(m):
                k = per[i]
                subgrad = SG(w, X[k], y[k], b, lam)
                w = w - (l_rate * subgrad[0])
                b = b - (l_rate * subgrad[1])
        return (w, b)
    else:
        ws = [w]
        bs = [b]
        for i in range(epochs * m):
            choice = np.random.randint(0, m)
            subgrad = SG(w, X[choice], y[choice], b, lam)
            w = w - (l_rate * subgrad[0])
            b = b - (l_rate * subgrad[1])
            ws.append(w)
            bs.append(b)
        return (sum(ws) / len(ws), sum(bs) / len(bs))
```

Q2.5.

[9]:
```
def calculate_error(w,bias,X,y):
    pred = np.add(w.dot(X.T),bias)
    pred[pred> 0] = 1
    pred[pred<=0] = -1
    return np.sum(pred!=y)/X.shape[0]
```

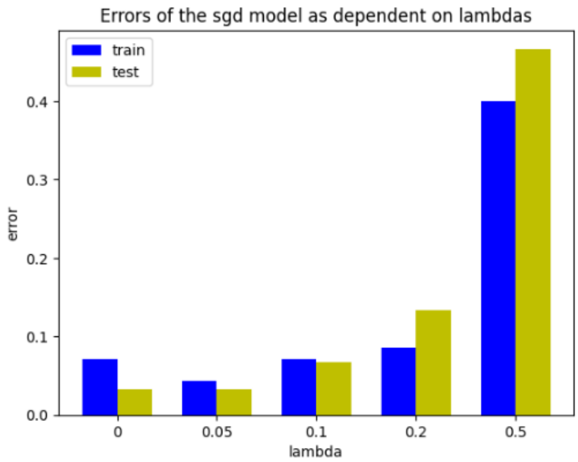
Q2.6.

[10]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# load data
X, y = load_iris(return_X_y=True)
X = X[y != 0]
y = y[y != 0]
y[y==2] = -1
X = X[:, 2:4]

# split data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=0)
```

[11]:
```
lambdas=[0, 0.05, 0.1, 0.2, 0.5]
train_errors =[]
test_errors = []
margins = []
for lam in lambdas:
    model = svm_with_sgd(X_train, y_train, lam)
    train_errors.append(calculate_error(model[0],model[1],X_train,y_train))
    test_errors.append(calculate_error(model[0],model[1],X_val,y_val))
    margins.append(1/np.linalg.norm(model[0]))
```

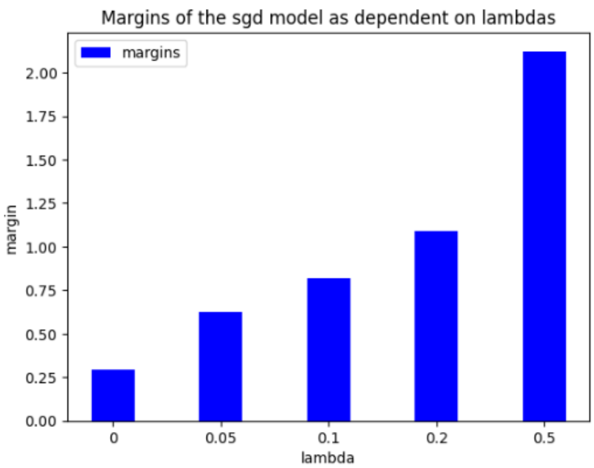
[17]:
```python
x = np.arange(len(lambdas))
plt.bar(x,train_errors,width=0.35,label='train', color='b')
plt.bar(np.add(x,0.35),test_errors,width=0.35,label='test', color='y')
plt.xlabel('lambda')
plt.ylabel('error')
plt.title('Errors of the sgd model as dependent on lambdas')
plt.xticks(np.add(x,0.35/2),lambdas)
plt.legend()
```

[17]: <matplotlib.legend.Legend at 0x832f520>



[18]:
```python
x = np.arange(len(lambdas))
plt.bar(x,margins,width=0.40,label='margins', color='b')
plt.xlabel('lambda')
plt.ylabel('margin')
plt.title('Margins of the sgd model as dependent on lambdas')
plt.xticks(x,lambdas)
plt.legend()
```

[18]: <matplotlib.legend.Legend at 0x83547b0>



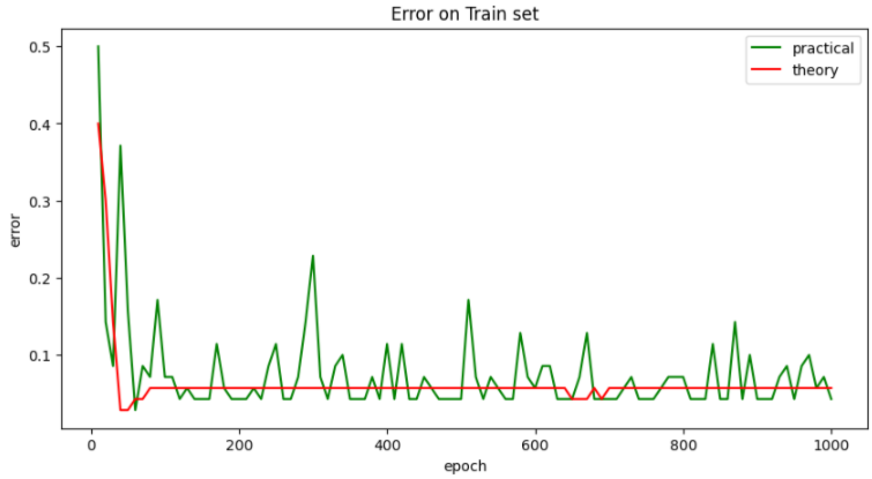
as we can see, for $\lambda = 0.05$ we get the smalest test and trai error. we dont get the smallest margin, be we get the second smalles, therfore we will chose $\lambda = 0.05$

Q2.7.a.

```
[14]: import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

def svm_with_sgd(X, y, lam=0, epochs=1000, l_rate=0.01, sgd_type='practical'):
    np.random.seed(2)
    m, d = X.shape
    b = np.random.uniform(0, 1, 1)
    w = np.random.uniform(0, 1, d)

    def SG(w, x, y, b, lam):
        if 1 - y * (w.dot(x) + b) <= 0:
            return (2 * lam * w, 0)
        else:
            return (-y * x + 2 * lam * w, -y)

    if sgd_type == 'practical':
        for j in range(epochs):
            per = np.random.permutation(np.arange(0, m))
            for i in range(m):
                k = per[i]
                subgrad = SG(w, X[k], y[k], b, lam)
                w = w - (l_rate * subgrad[0])
                b = b - (l_rate * subgrad[1])
        return (w, b)
    else:
        ws = [w]
        bs = [b]
        for i in range(epochs * m):
            choice = np.random.randint(0, m)
            subgrad = SG(w, X[choice], y[choice], b, lam)
            w = w - (l_rate * subgrad[0])
            b = b - (l_rate * subgrad[1])
            ws.append(w)
            bs.append(b)
        return (sum(ws) / len(ws), sum(bs) / len(bs))

def calculate_error(w, bias, X, y):
    pred = np.add(w.dot(X.T), bias)
    pred[pred > 0] = 1
    pred[pred <= 0] = -1
    return np.sum(pred != y) / X.shape[0]
```

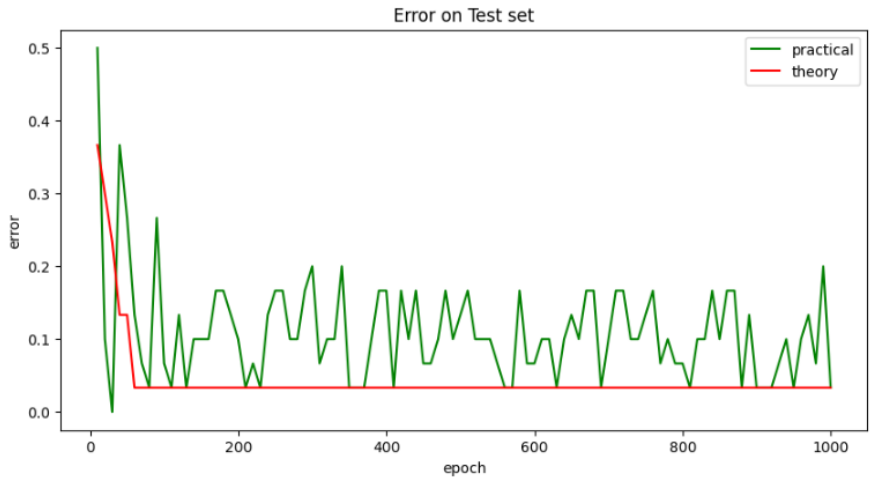
```
# Load data
X, y = load_iris(return_X_y=True)
X = X[y != 0]
y = y[y != 0]
y[y == 2] = -1
X = X[:, 2:4]

# Split data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=0)
epochs = np.arange(10, 1001, 10)
train_errors_practical = []
train_errors_theory = []
test_errors_practical = []
test_errors_theory = []

for epoch in epochs:
    modelP = svm_with_sgd(X_train, y_train, lam=0.05, epochs=epoch, sgd_type='practical')
    train_errors_practical.append(calculate_error(modelP[0], modelP[1], X_train, y_train))
    test_errors_practical.append(calculate_error(modelP[0], modelP[1], X_val, y_val))

    modelT = svm_with_sgd(X_train, y_train, lam=0.05, epochs=epoch, sgd_type='theory')
    train_errors_theory.append(calculate_error(modelT[0], modelT[1], X_train, y_train))
    test_errors_theory.append(calculate_error(modelT[0], modelT[1], X_val, y_val))
```

```
[19]: plt.figure(figsize=(10, 5))
plt.plot(epochs, train_errors_practical, label='practical', color='g')
plt.plot(epochs, train_errors_theory, label='theory', color='r')
plt.xlabel('epoch')
plt.ylabel('error')
plt.title('Error on Train set')
plt.legend()
plt.show()
```



Q2.7.b.

```
[20]: plt.figure(figsize=(10, 5))
plt.plot(epochs, test_errors_practical, label='practical', color='g')
plt.plot(epochs, test_errors_theory, label='theory', color='r')
plt.xlabel('epoch')
plt.ylabel('error')
plt.title('Error on Test set')
plt.legend()
plt.show()
```



we can see that in all aproches- theory, practical, train, test we get a highe error fro low epochs that decreases

the theory aproche is more stable then the practical aproche

in the train set we cancle the error in the theory aproche, and in the test set we balnce out around 0.05, likr the λ

שאלה 3

סעיף 1

```python
import numpy as np
import matplotlib.pyplot as plt
```

Executed at 2024.07.26 16:15:44 in 12ms

```python
def cross_validation_errors(X, y, model, folds):
    n = len(X)
    fold_size = n // folds
    train_errors = []
    val_errors = []

    for i in range(folds):
        start = i * fold_size
        end = (i + 1) * fold_size if i < folds - 1 else n  # Ensure the last fold includes all remaining data

        X_train = np.concatenate((X[:start], X[end:]))
        y_train = np.concatenate((y[:start], y[end:]))
        X_test = X[start:end]
        y_test = y[start:end]

        # Train the model
        model.fit(X_train, y_train)

        # Predict on training set
        y_train_pred = model.predict(X_train)
        train_error = np.mean(y_train_pred != y_train)
        train_errors.append(train_error)

        # Predict on validation set
        y_test_pred = model.predict(X_test)
        val_error = np.mean(y_test_pred != y_test)
        val_errors.append(val_error)

    average_train_error = np.mean(train_errors)
    average_val_error = np.mean(val_errors)

    return average_train_error, average_val_error
```

Executed at 2024.07.26 16:15:44 in 28ms

סעיף 2:

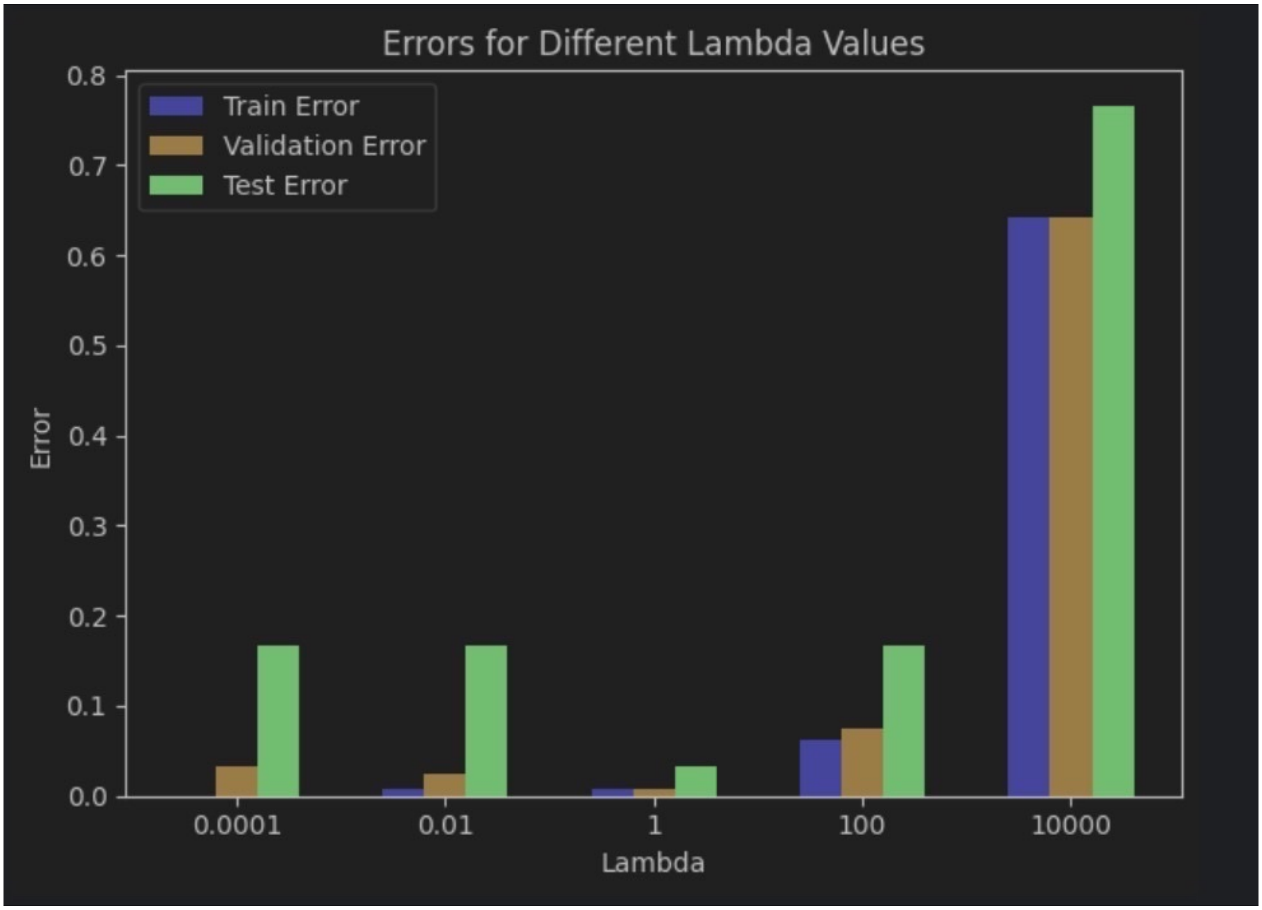
```python
import sklearn.svm as svm

def svm_results(X_train, y_train, X_test, y_test):
    results = {}
    lambda_values = [0.0001, 0.01, 1, 100, 10000]

    for lambda_val in lambda_values:
        # Define the SVM model with the given lambda
        model = svm.SVC(C=(1/lambda_val), kernel='linear')  # Example with polynomial kernel, adjust as needed

        # Calculate training and validation errors using cross-validation
        avg_train_error, avg_val_error = cross_validation_errors(X_train, y_train, model, folds=5)

        # Fit the model on the entire training set and compute test error
        model.fit(X_train, y_train)
        y_test_pred = model.predict(X_test)
        test_error = np.mean(y_test_pred != y_test)

        # Store the results in the dictionary
        results[f'lambda_{lambda_val}'] = (avg_train_error, avg_val_error, test_error)

    return results
```

סעיף 3:

```python
from sklearn.datasets import load_iris
iris_data = load_iris()
X, y = iris_data['data'], iris_data['target']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
```

Executed at 2024.07.26 16:15:44 in 58ms

```python
# Get the results
results = svm_results(X_train, y_train, X_test, y_test)

# Prepare data for plotting
lambda_values = [0.0001, 0.01, 1, 100, 10000]
train_errors = [results[f'lambda_{lambda_val}'][0] for lambda_val in lambda_values]
val_errors = [results[f'lambda_{lambda_val}'][1] for lambda_val in lambda_values]
test_errors = [results[f'lambda_{lambda_val}'][2] for lambda_val in lambda_values]

# Plotting
x = np.arange(len(lambda_values))  # the label locations
width = 0.2  # the width of the bars

fig, ax = plt.subplots()
bars1 = ax.bar(x - width, train_errors, width, label='Train Error', color='blue')
bars2 = ax.bar(x, val_errors, width, label='Validation Error', color='orange')
bars3 = ax.bar(x + width, test_errors, width, label='Test Error', color='green')

# Text for labels, title and x-axis tick labels
ax.set_xlabel('Lambda')
ax.set_ylabel('Error')
ax.set_title('Errors for Different Lambda Values')
ax.set_xticks(x)
ax.set_xticklabels(lambda_values)
ax.legend()

fig.tight_layout()

plt.show()
```



המודל הטוב ביותר לפי שיטת CV הוא גם המודל הטוב ביותר לפי Test Error
מודל זה הוא 1_lambda_SVM, כלומר עבור $\lambda=1$.
נשים לב כי לפי שיטת CV עבור ערכי λ קטנים המודל מבצע Overfitting
ל-(k-1) ה-folds שמשמשים כסט אימון. לכן השגיאה הממוצעת על סט
האימון נמוכה, והשגיאה הממוצעת על ה-fold שמשמש כסט מבחן גדולה.
בנוסף לפי שיטת CV עבור ערכי λ גדולים המודל מבצע הכללה. לכן השגיאה
הממוצעת על (k-1) ה-folds שמשמשים כסט אימון דומה לשגיאה הממוצעת על
ה-fold שמשמש כסט מבחן - אך שגיאה זו גדולה.

מכאן שעבור פרמטר רגולריזציה $\lambda=1$ נקבל "שיווי משקל" בין Overfitting להכללה
ולכן הוא הטוב ביותר מבין ערכי λ שבדקנו.
ניתן לראות כי הערכתנו טובה, בהרצת המודל על סט המבחן עליו לא התאמנו -
קיבלנו שאכן עבור $\lambda=1$ השגיאה הינה מינימלית (מבין ערכי λ שבדקנו).

שאלה 4

נתון: $g(w)=\max_{i\in[r]} g_i(w)$

לכל i, g_i קמורה וגזירה ב-$\mathbb{R}^d$.

$j\in\arg\max_{i} g_i(w)$

צ"ל: $\nabla g_j(w)$ סאב גרדיאנט של g ב-w.

נתון $j\in\arg\max_{i} g_i(w)$ ולכן $g(w)=\max_{i\in[r]} g_i(w)=g_j(w)$ *

יהי $u\in\mathbb{R}^d$. מתקיים עבורו:

אי-שיוויון הגרדיאנט (נתון g גזירה וקמורה)

$$g(u) \ge g(w)+\langle u-w,\nabla g(w)\rangle \underset{\text{הגדרת } g(w)}{=} g(w)+\langle u-w,\nabla\left(\max_{i\in[r]} g_i(w)\right)\rangle$$

$$\underset{*}{=} g(w)+\langle u-w,\nabla\left(g_j(w)\right)\rangle$$

$\Leftarrow$ $\nabla g_j(w)$ סאב גרדיאנט של g ב-w עפ"י הגדרה