CS164 Final Projects

Viet Hoang Tran Duong

Fall 2019

Problem Statement: We receive tons of emails every day, many of them are spam emails. How can we filter the spam email, using the concepts learned from this course?

Approach: We try two approaches and compare the accuracy of them: Logistic Regression and Support Vector Machine (SVM), both using the cvxpy package.

Constraints: Logistic Regression and SVM try to optimize the defined loss function, with the decision variable θ, which is a coefficient of a linear function. By nature, θ does not have constraints as it runs freely in R^n.

I will add layers of constraints by adding the l_1 and l_∞ Regularization. To convert the norms into linear optimization, I introduce slack variables as constraints, which will be discussed below.

Data: The data is from Kaggle SMS Spam Collection Dataset, which contains 5572 messages, tagged according to being ham (legitimate) or spam.

The data is subset into train set (80%) and test set (20%) randomly by sklearn train_test_split.

Data Processing:

Step 1: Remove punctuations and stopwords: Because the data is in texts (natural language), we need to preprocess it to fit into the optimization models. First, I remove the punctuation and stop words ("stopwords" from nltk package).

Step 2: Vectorize the text: To vectorize the texts, first, I use the term frequency-inverse document frequency method. However, it yields more dimensions than my data (9000+ dimensions $>$ 5572 data points). Hence, I use a better approach: python spacy package "en_core_web_lg." This package will map the SMS content into the 300-dimension vector space.

As we have little data, 300 dimensions are still too large, which might lead to overfitting the train set. Hence, I apply Principal Component Analysis (sklearn PCA) to reduce the dimensionality while retaining as much variance in the data as possible. I selected the top 30 dimensions that capture most of the variance in the data. Our data becomes (5572, 30): 5572 data points in 30 dimensions.

Step 3: Transform the type decode for 2 algorithms: Because Logistic Regression works best with {0,1} while SVM works better with {-1,1}. I map the {ham, spam} type to these values.

Outline for the methods:

- Logistic Regression: 3 versions: the standard version, with l_1 regularization, and with l_∞ regularization.
- SVM: 3 versions: the standard version, with l_1 regularization, and with l_∞ regularization.

All methods are then evaluated by the train set and test set errors.

Regularization: Introducing the norm (l_1, l_∞) into the cost functions will punish the factor that causes overfit the train data. l_1 norm will push the coefficient to be towards 0: reducing the number of features in the model altogether. l_∞ will punish the most expensive coefficient factor (the biggest absolute value). Overall, regularization avoids overfitting, and subsequently, increases the accuracy.

Logistic Regression: Input: X (features matrix), Y (desired output), Output θ: coefficient
There are m data points, each with n features (in the data, it's 5572 data points and 30 features).

Feasible region: $\theta \in R^n$

Decision function: $sigmoid(x) = \frac{1}{1+e^{-\theta x}}$: the $sigmoid$ function will return the value between 0 and 1: which is the probability of the email is spam. If the value is larger than 0.5, then return 1 (spam), else, return 0 (not spam).

We will minimize the cost the function of the log-likelihood function of the data:

$$L(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right)$$

We will prove that $L(\theta)$ is convex:
$g(\theta) = \ln\left(1+e^{\theta x}\right)$
$\Rightarrow g'(\theta) = \frac{\left(1+e^{\theta x}\right)'}{1+e^{\theta x}} = \frac{xe^{\theta x}}{1+e^{\theta x}} = \frac{x}{1+e^{-\theta x}}$
$\Rightarrow g''(\theta) = \frac{x'\left(1+e^{-\theta x}\right) - \left(1+e^{-\theta x}\right)'x}{\left(1+e^{-\theta x}\right)^2} = \frac{0\left(1+e^{-\theta x}\right) - (-x)e^{-\theta x}x}{\left(1+e^{-\theta x}\right)^2} = \frac{x^2e^{-\theta x}}{\left(1+e^{-\theta x}\right)^2} \geq 0\ \forall\theta, x \in D$
(because $x^2 \geq 0,\ e^{-\theta x} \geq 0, \left(1+e^{-\theta x}\right)^2 \geq 0$)

As the second-order derivative of $g(x)$ is always nonnegative, hence, $g(x)$ is convex

We have:

- $y_i\theta x_i$ is a linear function: affine $\Rightarrow$ concave
- $g(\theta) = \ln\left(1+e^{\theta x}\right)$: convex $\forall\theta, x \Rightarrow -\ln\left(1+e^{\theta x_i}\right)$: concave $\forall\theta, x_i$
- Sum of concave functions is a concave function

$\Rightarrow \sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right)$: concave function. Multiply with $-\frac{1}{m} < 0$ will turn this into convex:

$\Rightarrow L(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right)$: convex function. (*1)

Hence, this problem is a convex optimization problem.

To avoid overfitting, I will perform regularization with a l_1 and l_∞ norm: penalize the loss function if the norms become too big. The regularization term is $\lambda|\theta|$, with λ is a set parameter.

For l_1: $\Rightarrow L^*(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right) + \lambda|\theta|$

We introduce a slack variable: $s, s \in R^{n\times 1}$ such that: $|\theta| < s \Leftrightarrow -s \leq \theta \leq s$
<u>The loss function becomes</u>: $L^*(\theta, s) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right) + \lambda 1^T s$

As $\lambda 1^T s$ is a linear function: affine $\Rightarrow$ convex, $L(\theta)$ is a convex function (*1), hence, the sum of these 2 functions: $L^*(\theta, s) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right) + \lambda 1^T s$: convex. Therefore, even with the regularization term, this is still a convex optimization problem. (*2)

For $\boldsymbol{l}_\infty$: $\Rightarrow\ L^*(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right) + \lambda|\theta|_\infty$

We introduce a slack variable: $s, s \in R$ such that: $\max(|\theta|) < s \Leftrightarrow |\theta_i| \leq s\ \forall i \ \Leftrightarrow 1 * -s \leq \theta \leq 1 * s$
<u>The loss function becomes</u>: $L^*(\theta, s) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right) + \lambda s$

As λs is a linear function: affine $\Rightarrow$ convex, $L(\theta)$ is a convex function (*1), hence, the sum of these 2 functions: $L^*(\theta, s) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\theta x_i - \ln\left(1+e^{\theta x_i}\right)\right) + \lambda s$: convex. Therefore, even with the regularization term, this is still a convex optimization problem. (*3)

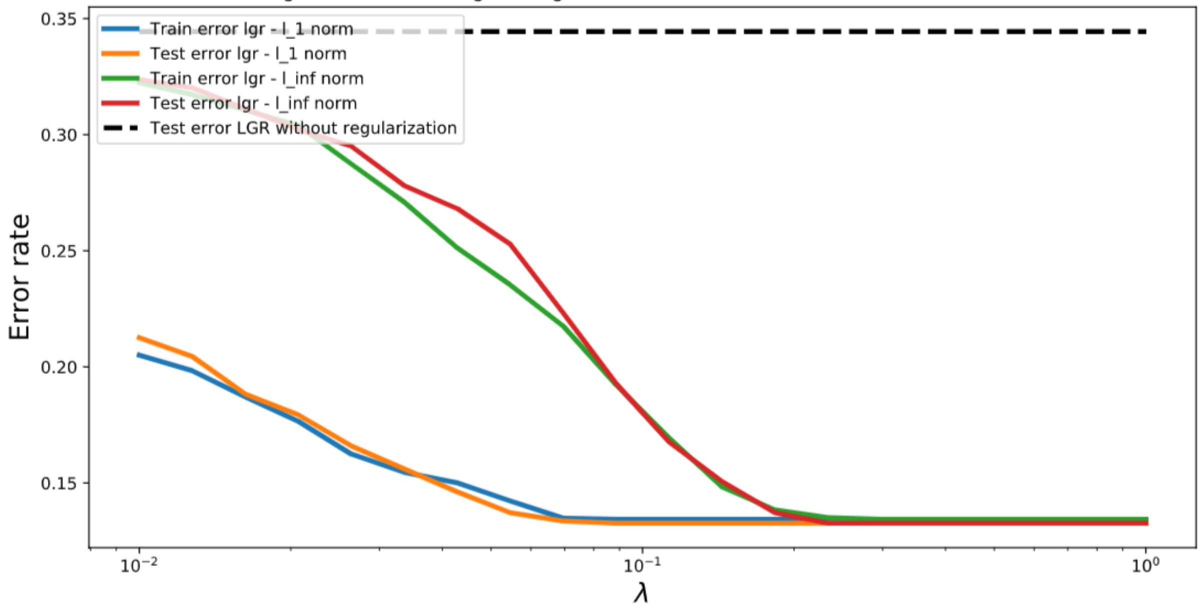
From (*1), (*2), (*3): All the loss function (with/without regularization) are all convex, and we are trying to minimize them. Hence, it is guaranteed to find the optimal solutions.

<u>Without the regularization term, here are the results</u>:

```
Results for logistic regression without regularization:
Status: optimal
Train error: 0.34305586717522996
Test error: 0.34439461883408073
```

<u>With the regularization term:</u> I vary the value of λ with the 10 exponents of 20 uniformly distributed values in [-2,0]: $\lambda \in [10^{-2}, 10^{0}]$

Fig.1: Error rate of logistic regression for train and test sets with values of λ



SVM: Input: X (features matrix), Y (desired output); Output $\boldsymbol{\theta}$: coefficient
There are m data points, each with n features (in the data, it's 5572 data points and 30 features).

Feasible region: $\theta \in R^n$

Decision function: $y = \text{sign}(\theta x - v)$: if $\theta x - v > 0$: $y = 1$ (spam); else and $y = -1$ (not spam)

Objective function: minimize loss function:

$$L(\theta, v) = \frac{1}{m}\sum_{i=1}^{m}\left(1 - y_i(\theta x_i - v)\right)$$

We have, $L(\theta, v)$ is a linear function, hence, convex. (*4)
Hence, this is a convex optimization problem.

To avoid overfitting, I will perform regularization with a l_1 and l_∞ norm: penalize the loss function if the norms become too big. The regularization term is $\lambda|\theta|$, with λ is a set parameter.

For $\boldsymbol{l_1}$: $\Rightarrow\ L^*(\theta, v) = \frac{1}{m}\sum_{i=1}^{m}(1 - y_i(\theta x_i - v)) + \lambda|\theta|_1$

We introduce a slack variable: $s, s \in R^{n\times 1}$ such that: $|\theta|_1 < s \Leftrightarrow -s \leq \theta \leq s$

The loss function becomes: $L^*(\theta, v, s) = \frac{1}{m}\sum_{i=1}^{m}(1 - y_i(\theta x_i - v)) + \lambda 1^T s$

As $\lambda 1^T s$ is a linear function: affine $\Rightarrow$ convex, $L(\theta)$ is a convex function (*), hence, the sum of these 2 functions: $L^*(\theta, v, s) = \frac{1}{m}\sum_{i=1}^{m}(1 - y_i(\theta x_i - v)) + \lambda 1^T s$: convex. Therefore, even with the regularization term, this is still a convex optimization problem. (*5)

For $\boldsymbol{l_\infty}$: $\Rightarrow\ L^*(\theta, v) = \frac{1}{m}\sum_{i=1}^{m}(1 - y_i(\theta x_i - v)) + \lambda|\theta|_\infty$

We introduce a slack variable: $s, s \in R$ such that: $\max(|\theta|) < s \Leftrightarrow |\theta_i| \leq s\ \forall i\ \Leftrightarrow 1 * -s \leq \theta \leq 1 * s$

The loss function becomes: $L^*(\theta, s) = \frac{1}{m}\sum_{i=1}^{m}(1 - y_i(\theta x_i - v)) + \lambda s$

As λs is a linear function: affine $\Rightarrow$ convex, $L(\theta)$ is a convex function (*), hence, the sum of these 2 functions: $L^*(\theta, s) = \frac{1}{m}\sum_{i=1}^{m}(1 - y_i(\theta x_i - v)) + \lambda s$: convex. Therefore, even with the regularization term, this is still a convex optimization problem. (*6)

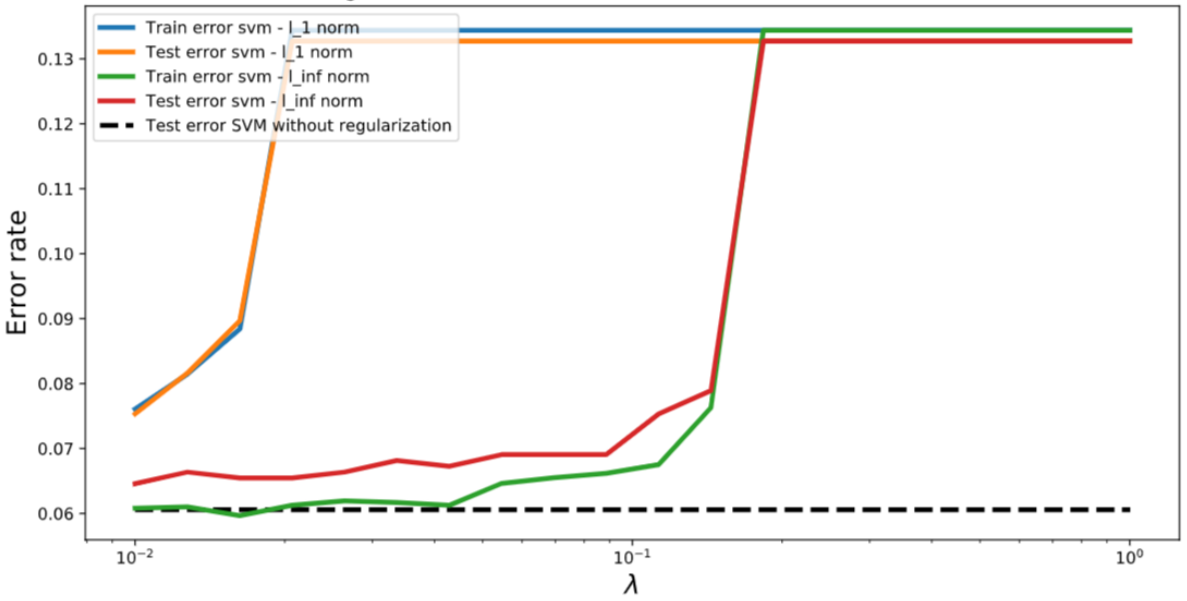
From (*40, (*5), (*6): All the loss function (with/without regularization) are all convex, and we are trying to minimize them, it is guaranteed to find the optimal solutions.

Without the regularization term, here are the results:

```
Results for SVM without regularization:
Status: optimal
Train error: 0.060578864707202154
Test error: 0.060578864707202154
```

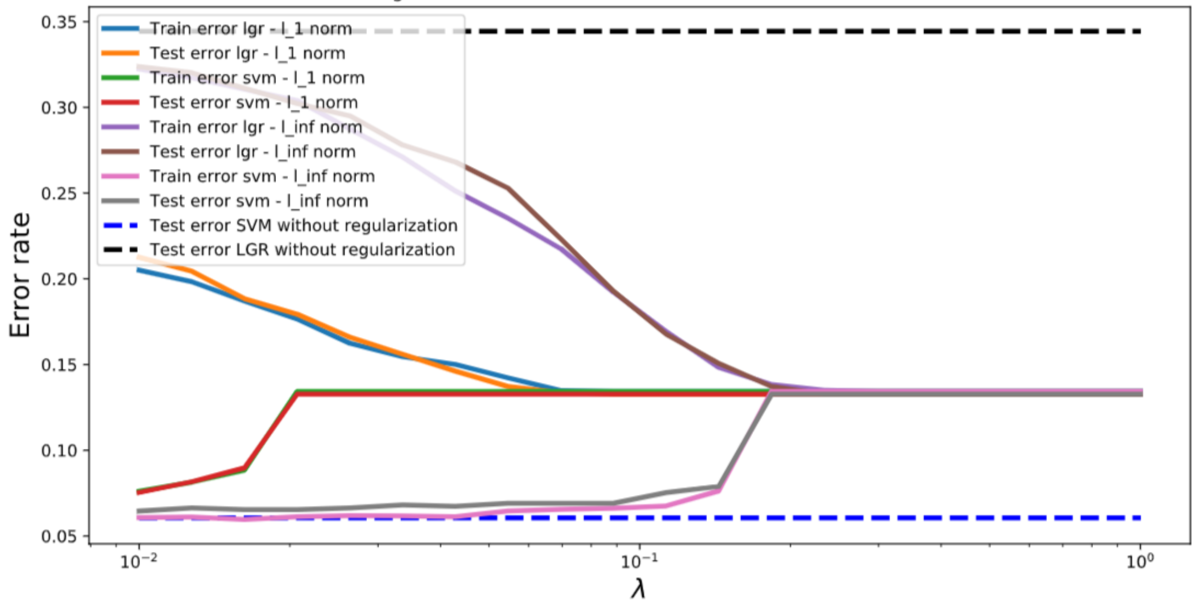
With the regularization term, I vary the value of λ with the 10 exponent of 20 uniformly distributed values in [-2,0]: $\lambda \in [10^{-2}, 10^{0}]$

Fig 2: Error rate of SVM for train and test sets with values of λ



Total results:

Fig 3: Error rate of train and test sets with values of λ



Conclusion:

Overall, these classic methods can be solved by convex optimization, as the nature of the loss functions is convex. Furthermore, we can add regularization through slack variables, which still maintain the convex property (while adding linear constraints).
SVM outperformed the Logistic Regression, especially when $\lambda = 10^{-2}$. For logistics regression, increasing λ increases the accuracy, while increasing λ makes the performance of SVM worse.

HCs applications:

#context: I tailored my writing to fit the context of a mathematics and code report, which are concise, on point, adding with multiple illustrations (figures) to compare between methods. This paper has demonstrated proper use of #context, where I understand the context of this assignment and build content upon.

#biasidentification + #biasmitigation: I apply the concept of bias into the data science: the bias-variance tradeoff: I apply regularization to reduce the variance of the estimator by simplifying it (discourage learning a more complex model by shrinking the coefficient estimates towards 0), which will avoid the risk of overfitting. The bias in Empirical Analysis relates to bias-variance reduction for a better model, which is essentially achieved by reducing overfitting using regularization. This paper is an excellent application of #biasidentification (acknowledging the existence of high variance), and fixing the overfitting by using #biasmitigation,

#connotation: The connotation (tone, style) of the language within this document is clear, concise, with correct terminology and methods application. All these components contribute to a concise report that suitable for the targeted audience: those interested in simple methods (logistics, SVM) to solve abstract natural language problems.

LOs applications:

#convexprogramming: For this project, the objective functions are all convex, even with or without the regularization factor. I also implement all the code in python, with combined use of numpy, cvxpy, pandas, and other packages. Finally, I automate the system to solve multiply convex optimizations problems to evaluate the parameters' performance.

#convexity: I analytically show the loss function of the logistic regression (the log-likelihood function) is convex by using the second-order derivative. I also analytically showed all functions to be convex, turning the norm factors into linear programming using slack variables. These proofs are demonstrations of the proper application of #convexity.

References:

Dataman. (2019, October 24). Dimension Reduction Techniques with Python. Retrieved from https://towardsdatascience.com/dimension-reduction-techniques-with-python-f36ca7009e5c.

Loh, P.-S. (n.d.). Convexity - math.cmu.edu. Retrieved from http://www.math.cmu.edu/~ploh/public_html/docs/math/mop2013/convexity-soln.pdf.

MLMath.io. (2019, February 16). Math behind Support Vector Machine(SVM). Retrieved from https://medium.com/@ankitnitjsr13/math-behind-support-vector-machine-svm-5e7376d0ee4d.

Sanjeevi, M., & Mady. (2017, September 26). Chapter 2.0 : Logistic Regression with Math. Retrieved from https://medium.com/deep-math-machine-learning-ai/chapter-2-0-logistic-regression-with-math-e9cbb3ec6077.

UCI Machine Learning. (2016, December 2). SMS Spam Collection Dataset. Retrieved from https://www.kaggle.com/uciml/sms-spam-collection-dataset/data.

Link to code:

- https://colab.research.google.com/drive/1nusjx9nli2fO_ri0_2f7Ybruzw5ZF5Ai
- Addition: 4 layers Neural Network built from scratch train on the same dataset: https://colab.research.google.com/drive/1Q0BvysgFPHhuK91Vb5lIqAegNsZ_m0zL

```
# import packages: numpy, pandas, nltk, spacy, and sklearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import spacy
from sklearn.decomposition import PCA
import time
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
import cvxpy as cvx
import random
import copy
import string
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

# set seed for consistency
np.random.seed(2019)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# package installation

#!spacy download en_core_web_lg
```

```
# dimension setting function
# if this code fell to run (can't recognize package en_core_web_lg), then run the li
# !spacy download en_core_web_lg
# then restart the kernel.

nlp = spacy.load('en_core_web_lg')
```

Read the data

```
# spam dataset: https://www.kaggle.com/uciml/sms-spam-collection-dataset/data
data = pd.read_csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vRzkLSK-20SdsUIL

# only takes the 1st 2 columns (type and content)
data = data[["v1", "v2"]]
# rename the columns
data.columns = ["Type", "Content"]

# check null values
for column in data.columns:
  if sum(data[column].isnull()) != 0:
    print("Exist Null value: ", column)

print("Number of data points:", len(data))
print("")
data.head()
```

```
Number of data points: 5572
```

	Type	Content
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
# quick description on the data
data.groupby("Type").describe()
```

	Content			
	count	unique	top	freq
Type				
ham	4825	4516	Sorry, I'll call later	30
spam	747	647	Please call our customer service representativ...	4

Data Preprocessing

```
# remove punctuations and stopwords

def txt_preprocess(txt):
  txt = txt.translate(str.maketrans('', '', string.punctuation))
  txt = [word for word in txt.split() if word.lower() not in stopwords.words('englis
  return " ".join(txt)

# make a copy of the data so that we won't change the original dataframe
process_text = copy.deepcopy(data["Content"])

# preprocess the data
process_text = process_text.apply(txt_preprocess)
```

```
# map the data into 300 dimensions spacy vector space
train_nlps = [nlp(text) for text in process_text]
train_vecs = [text.vector for text in train_nlps]

# feature matrix
X = np.array(train_vecs)
```

```
# use pca to remove the dimension of the data: 300 -> 30 dimensions: keep those with
pca = PCA(n_components=30)
X = pca.fit_transform(X)
```

```
# transform the type data: logistic regression {0,1}; SVM {-1,1}
Y_lgr = np.array([data["Type"].map({"ham":  0, 'spam':1})]).T
Y_svm = np.array([data["Type"].map({"ham": -1, 'spam':1})]).T
```

```
# split the dataset into train and test
X_train, X_test, Y_train_lgr, Y_test_lgr = train_test_split(X,
                                                         Y_lgr, test_size=0.2, randor

# this code is to have consistency between train and test (for both approaches to ha
# logistics: {0,1} * 2 --> {0,2} - 1 --> {-1,1}: the values suitable for SVM
Y_train_svm = Y_train_lgr*2-1
Y_test_svm = Y_test_lgr*2-1
```

```
# check dimension
m = X_train.shape[0]
n = X_train.shape[1]
ones = np.ones((n,1))

print(m, "data points")
print(n, "features")
```

```
4457 data points
30 features
```

```

```

logistic with L_1 norm

```
# error check
def error(scores, labels):
  return np.mean((np.sign(scores + 1)//2) != labels)
```

```
# define variables
coeff = cvx.Variable((n,1))
slack_1 = cvx.Variable((n,1), nonneg = True)

```

```
# Parameters
lambd = cvx.Parameter(nonneg=True)

# cost function
cost = cvx.sum(cvx.multiply(Y_train_lgr, X_train @ coeff) - cvx.logistic(X_train @ c

# objective function: -1/m*cost + regularization
obj = cvx.Minimize(-cost/m + lambd * (ones.T@slack_1))

# constraints
con = [coeff >= -slack_1, coeff <= slack_1]

# define problem
problem = cvx.Problem(obj, con)
```

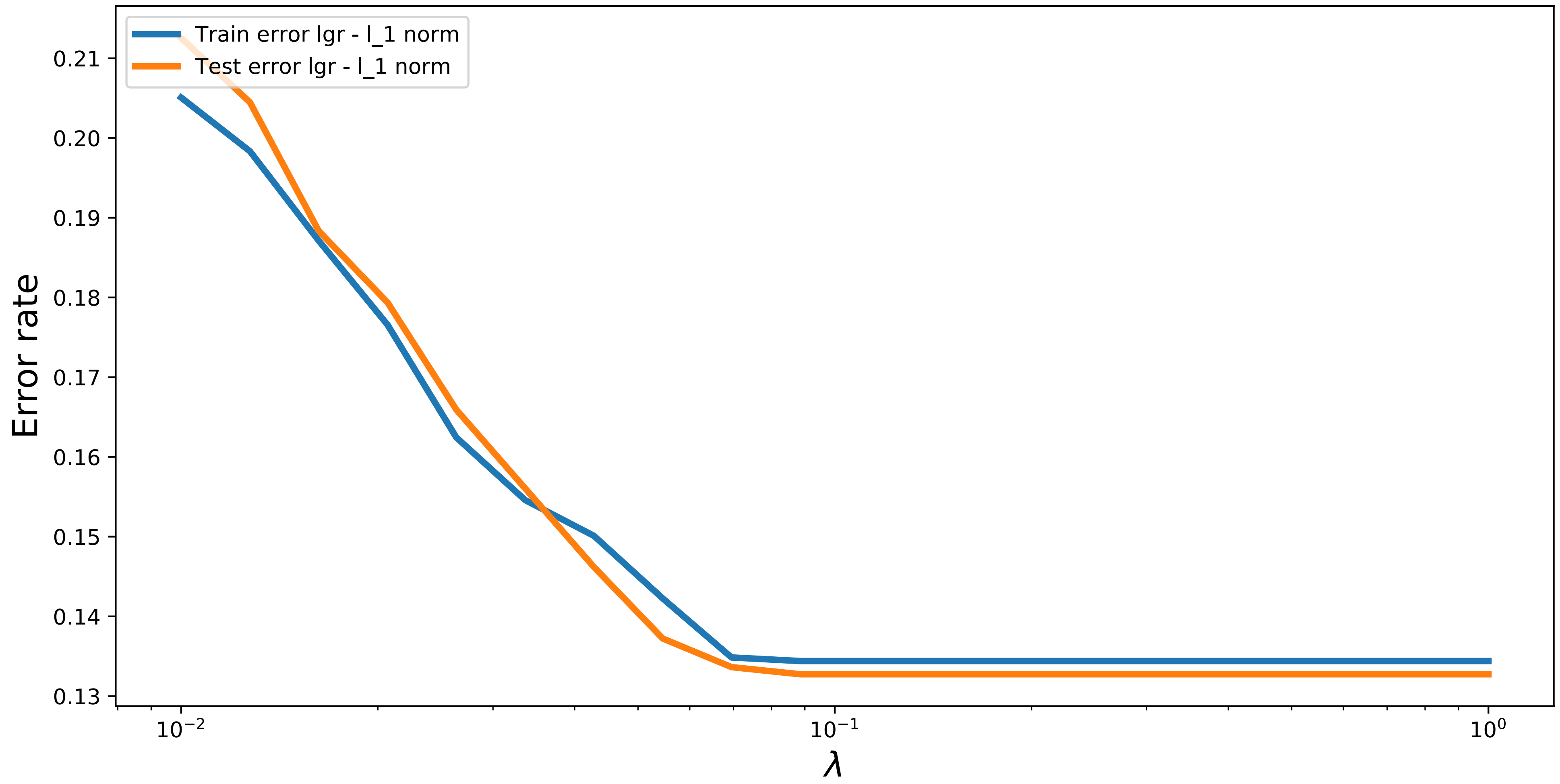
```
# Try out with no regularization: lambd = 0
lambd.value = 0
problem.solve(solver = cvx.ECOS)
print("Results for logistic regression without regularization:")
print("Status:", problem.status)

# calculate train/test error
train_error_lgr = error((X_train @ coeff).value, Y_train_lgr)
test_error_lgr = error((X_test @ coeff).value, Y_test_lgr)

print("Train error:", train_error_lgr)
print("Test error:", test_error_lgr)
```

```
Results for logistic regression without regularization:
Status: optimal
Train error: 0.34305586717522996
Test error: 0.34439461883408073
```

```
# number of values for lambd:
trials = 20

# to store the error rate
train_error_lgr_1 = []
```

```
test_error_lgr_1 = []

# space out the values by log of 10: to 10^-2 to 10^0
lambda_vals = np.logspace(-2, 0, trials)

# try out with different lambd values
for i in range(trials):
  lambd.value = lambda_vals[i]
  problem.solve(solver = cvx.ECOS)

  # store the error rate
  train_error_lgr_1 += [error((X_train @ coeff).value, Y_train_lgr)]
  test_error_lgr_1 += [error((X_test @ coeff).value, Y_test_lgr)]
```

```
# plot the error rates
plt.figure(figsize=(12,6))
plt.plot(lambda_vals, train_error_lgr_1, label="Train error lgr - l_1 norm", linewic
plt.plot(lambda_vals, test_error_lgr_1, label="Test error lgr - l_1 norm", linewidtl
plt.xscale("log")
plt.legend(loc="upper left")
plt.xlabel(r"$\lambda$", fontsize=16)
plt.ylabel("Error rate", fontsize=16)
plt.show()
```



```
```

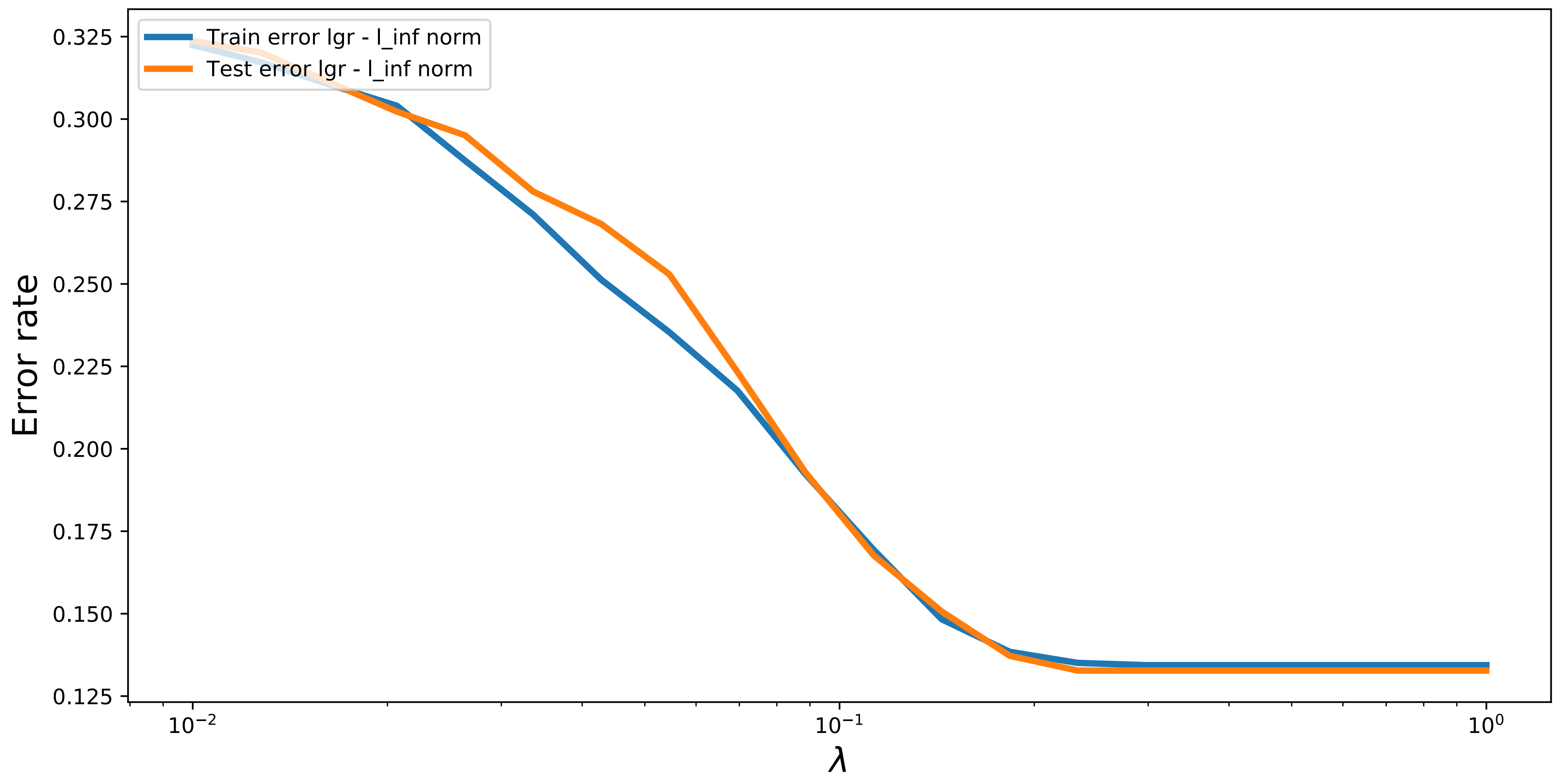
logistic with linf norm

```
# variables
coeff = cvx.Variable((n,1))
slack_inf = cvx.Variable((1,1))
```

```

# define parameters
lambd = cvx.Parameter(nonneg=True)

# cost functiton
cost = cvx.sum(cvx.multiply(Y_train_lgr, X_train @ coeff) - cvx.logistic(X_train @ c

# objective function
obj = cvx.Minimize(-cost/m + lambd * slack_inf)

# constraint for slack variable
con = [coeff >= -slack_inf@ones, coeff <= slack_inf@ones]

# define problem
problem = cvx.Problem(obj, con)
```

```
# number of values for lambd:
trials = 20

# to store the error rate
train_error_lgr_inf = []
test_error_lgr_inf = []

# space out the values by log of 10: to 10^-2 to 10^0
lambda_vals = np.logspace(-2, 0, trials)

# try out with different lambd values
for i in range(trials):
  lambd.value = lambda_vals[i]
  problem.solve(solver = cvx.ECOS)

  # store the error rate
  train_error_lgr_inf += [error((X_train @ coeff).value, Y_train_lgr)]
  test_error_lgr_inf += [error((X_test @ coeff).value, Y_test_lgr)]
```

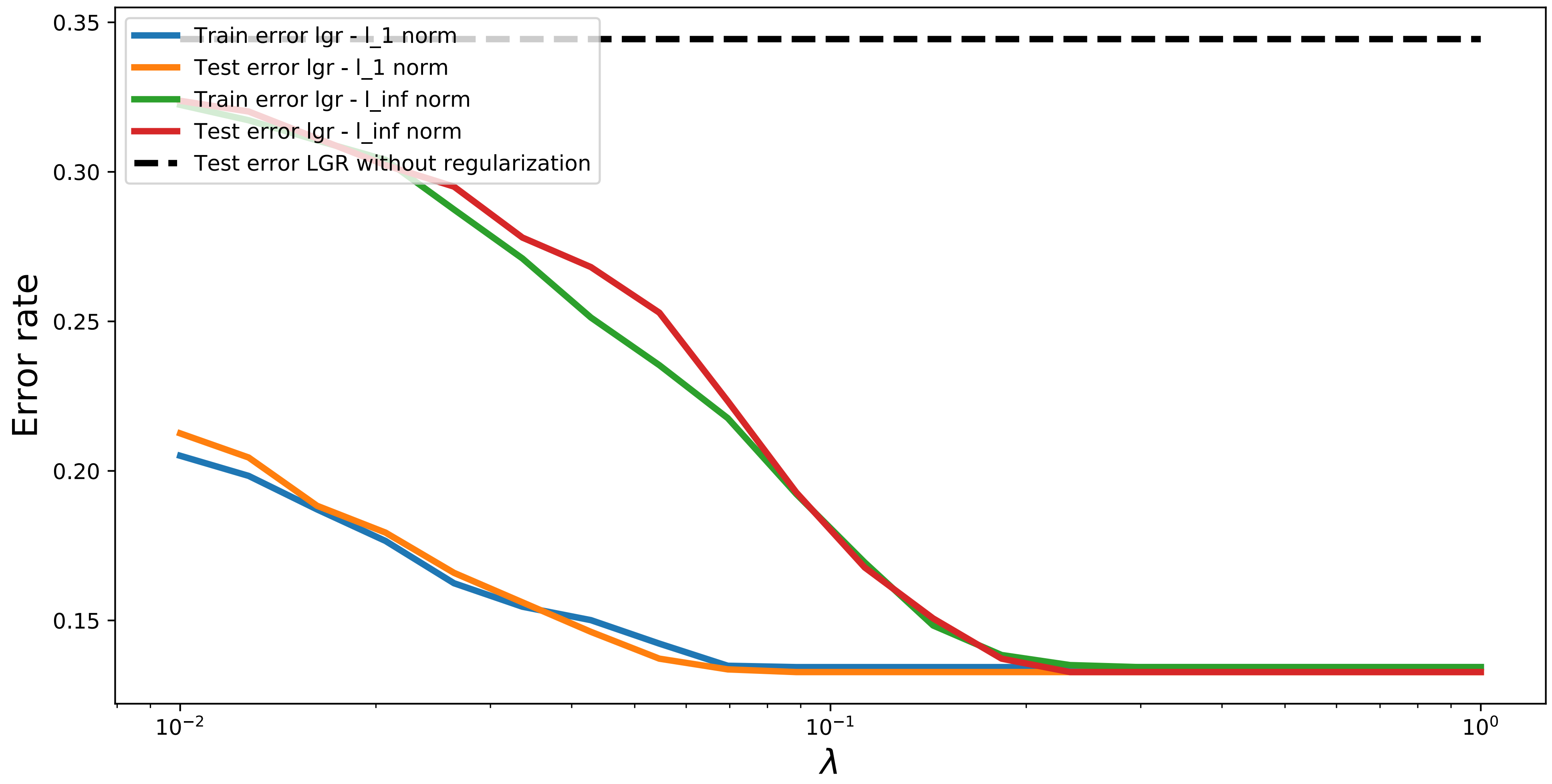
```
# plot the error rates
plt.figure(figsize=(12,6))
plt.plot(lambda_vals, train_error_lgr_inf, label="Train error lgr - l_inf norm", lir
plt.plot(lambda_vals, test_error_lgr_inf, label="Test error lgr - l_inf norm", linew
plt.xscale("log")
plt.legend(loc="upper left")
plt.xlabel(r"$\lambda$", fontsize=16)
plt.ylabel("Error rate", fontsize=16)
plt.show()
```



Logistic summary

```
# plot the combines error rates
plt.figure(figsize=(12,6))
plt.plot(lambda_vals, train_error_lgr_1, label="Train error lgr - l_1 norm", linewic
plt.plot(lambda_vals, test_error_lgr_1, label="Test error lgr - l_1 norm", linewidtl
plt.plot(lambda_vals, train_error_lgr_inf, label="Train error lgr - l_inf norm", lir
plt.plot(lambda_vals, test_error_lgr_inf, label="Test error lgr - l_inf norm", linew
plt.hlines(test_error_lgr, 0.01, 1, linestyle = "--", label="Test error LGR without
plt.xscale('log')
plt.legend(loc='upper left')
plt.xlabel(r"$\lambda$", fontsize=16)
plt.ylabel("Error rate", fontsize=16)
plt.title("Fig.1: Error rate of logistic regression for train and test sets with val
plt.show()
```

Fig.1: Error rate of logistic regression for train and test sets with values of λ



```

```

SVM with L1 norm

```
# define variables
coeff = cvx.Variable((n,1))
```

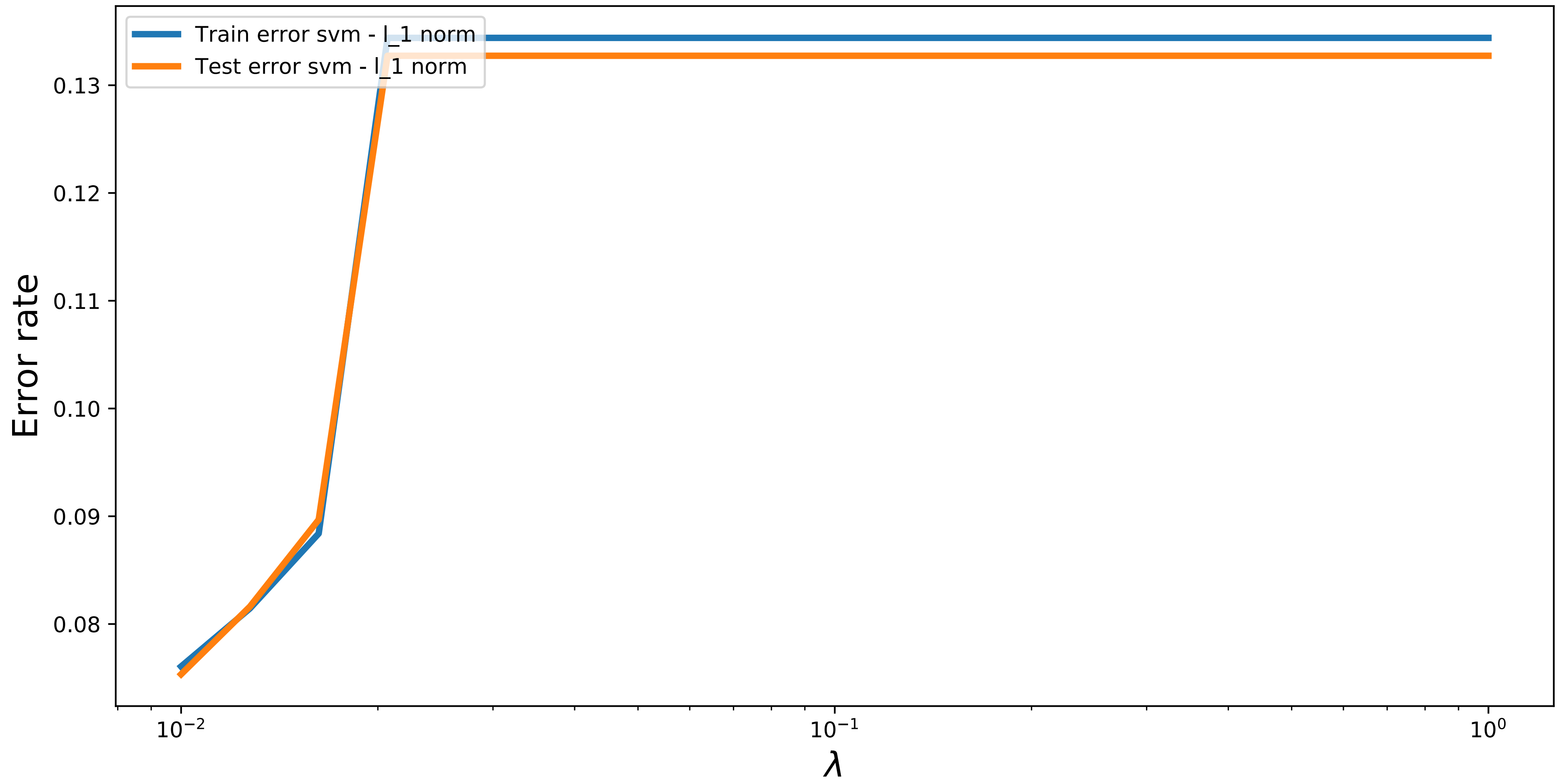
```
v = cvx.Variable()
slack_1 = cvx.Variable((n,1), nonneg = True)

# cost functions
cost = cvx.sum(cvx.pos(1 - cvx.multiply(Y_train_svm, X_train*coeff - v)))

# parameters
lambd = cvx.Parameter(nonneg=True)

# objective function
obj = cvx.Minimize(cost/m + lambd * (ones.T@slack_1))

# constraint
con = [coeff >= -slack_1, coeff <= slack_1]

# define problem
problem = cvx.Problem(obj, con)
```

```
# try out with out regularization
lambd.value = 0
problem.solve(solver = cvx.ECOS)
print("Results for SVM without regularization:")
print("Status:", problem.status)

train_error_svm = np.mean(Y_train_svm != np.sign(X_train.dot(coeff.value) - v.value
test_error_svm = np.mean(Y_train_svm != np.sign(X_train.dot(coeff.value) - v.value)

print("Train error:", train_error_svm)
print("Test error:", test_error_svm)
```

```
Results for SVM without regularization:
Status: optimal
Train error: 0.060578864707202154
Test error: 0.060578864707202154
```

```
# number of values for lambd:
trials = 20

# to store the error rate
# to store the error rate
train_error_svm_1 = []
test_error_svm_1 = []

# space out the values by log of 10: to 10^-2 to 10^0
lambda_vals = np.logspace(-2, 0, trials)

# try out with different lambd values
for i in range(trials):
    lambd.value = lambda_vals[i]
    problem.solve(solver = cvx.ECOS)

    # store the error rate
    train_error_svm_1 += [np.mean(Y_train_svm != np.sign(X_train.dot(coeff.value) -
    test_error_svm_1 += [np.mean(Y_test_svm != np.sign(X_test.dot(coeff.value) - v.\
```

```
plt.figure(figsize=(12,6))
plt.plot(lambda_vals, train_error_svm_1, label="Train error svm - l_1 norm", linewic
plt.plot(lambda_vals, test_error_svm_1, label="Test error svm - l_1 norm", linewidtl
plt.xscale('log')
plt.legend(loc='upper left')
plt.xlabel(r"$\lambda$", fontsize=16)
plt.ylabel("Error rate", fontsize=16)
plt.show()
```



```
1
```

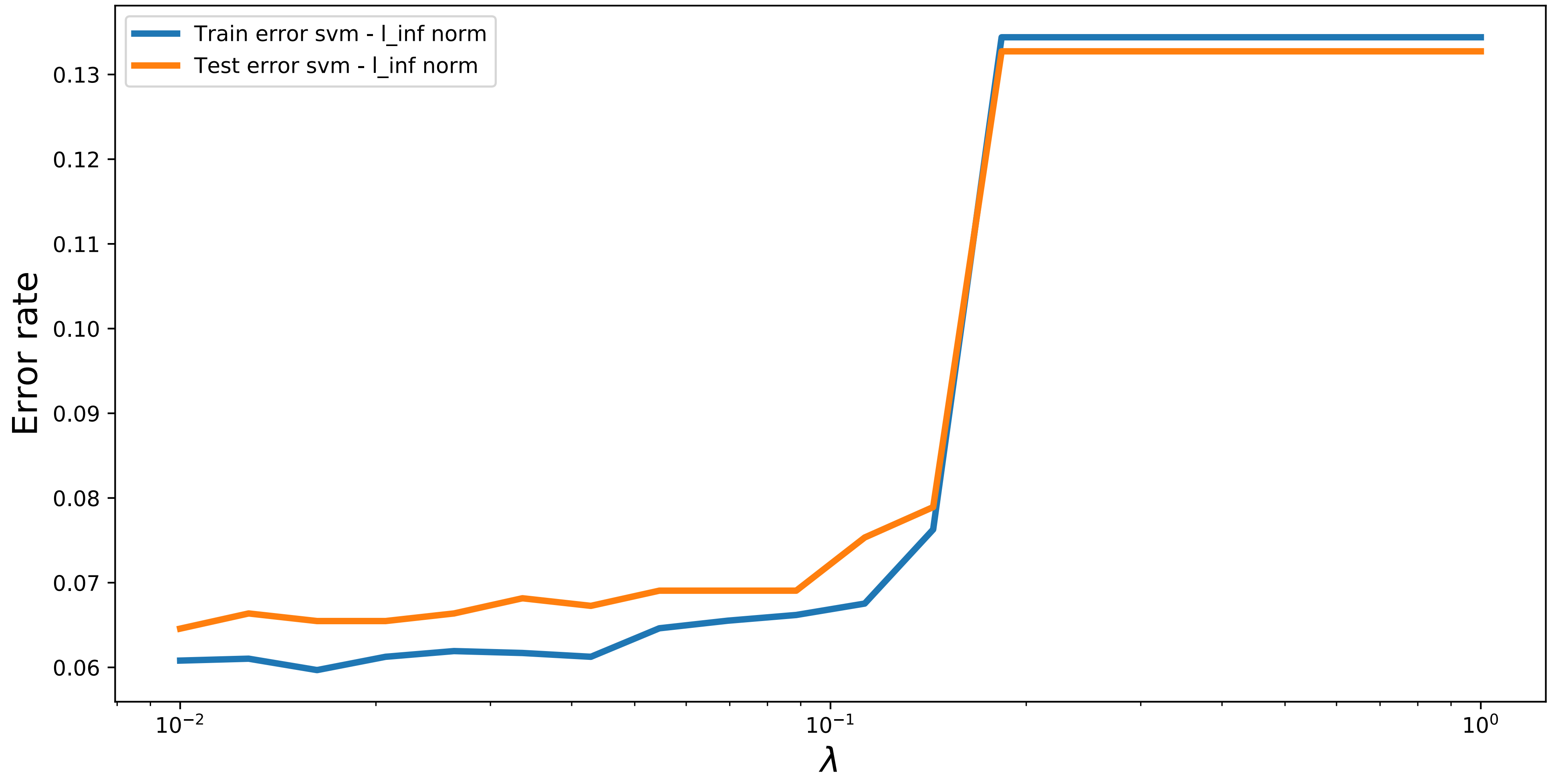
SVM linf

```
# define variable
coeff = cvx.Variable((n,1))
v = cvx.Variable()
```

```
slack_inf = cvx.Variable((1,1))

# cost function
cost = cvx.sum(cvx.pos(1 - cvx.multiply(Y_train_svm, X_train*coeff - v)))

# parameters
lambd = cvx.Parameter(nonneg=True)

# objective function
obj = cvx.Minimize(cost/m + lambd * slack_inf)

# constraint
con = [coeff >= -slack_inf@ones, coeff <= slack_inf@ones]

# define problem
problem = cvx.Problem(obj, con)
```

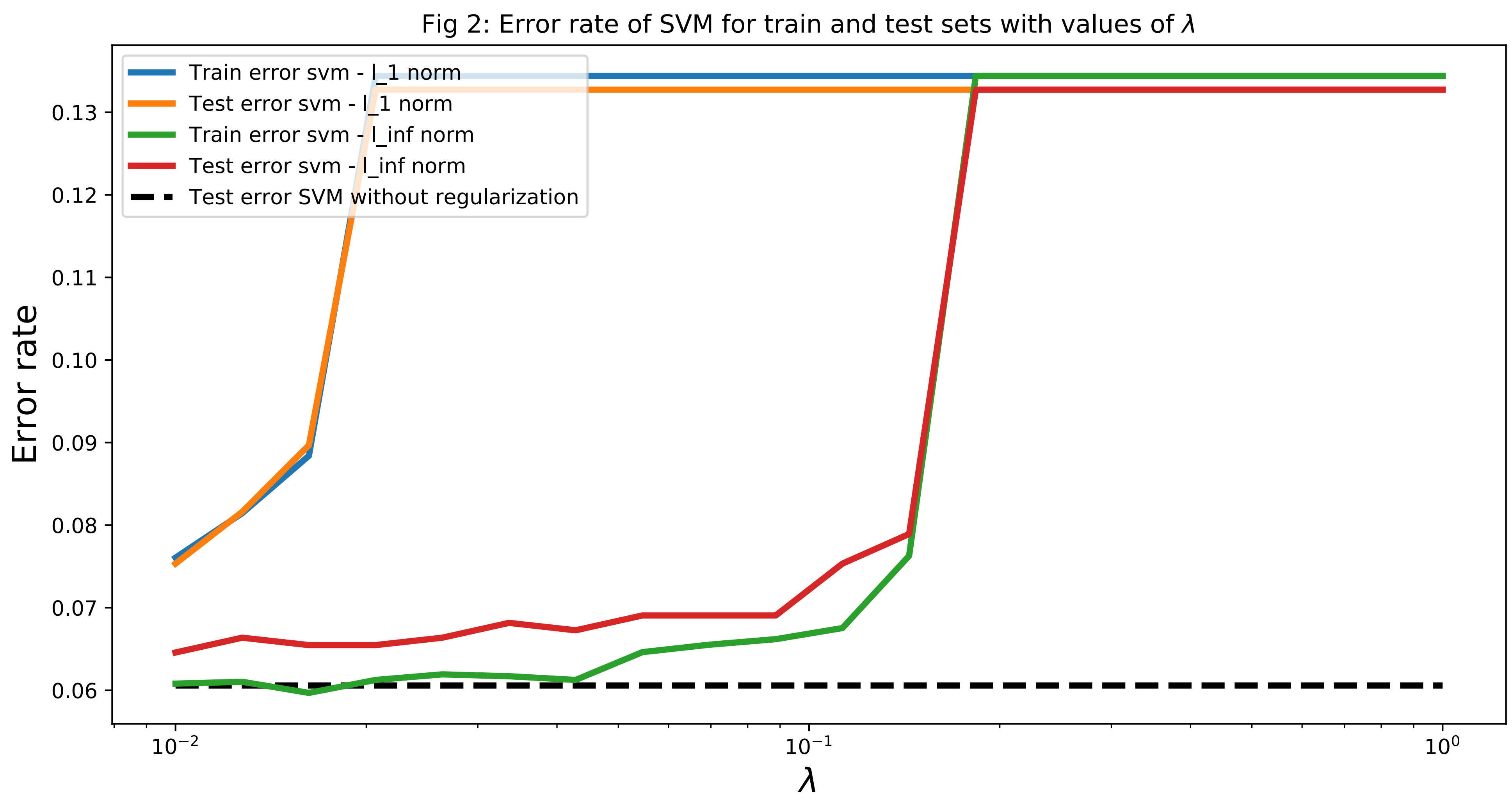
```
# number of values for lambd:
trials = 20

# to store the error rate
train_error_svm_inf = []
test_error_svm_inf = []

# space out the values by log of 10: to 10^-2 to 10^0
lambda_vals = np.logspace(-2, 0, trials)

# try out with different lambd values
for i in range(trials):
    lambd.value = lambda_vals[i]
    problem.solve(solver = cvx.ECOS)

    # store the error rate
    train_error_svm_inf += [np.mean(Y_train_svm != np.sign(X_train.dot(coeff.value)
    test_error_svm_inf += [np.mean(Y_test_svm != np.sign(X_test.dot(coeff.value) - v
```

```
plt.figure(figsize=(12,6))
plt.plot(lambda_vals, train_error_svm_inf, label="Train error svm - l_inf norm", lin
plt.plot(lambda_vals, test_error_svm_inf, label="Test error svm - l_inf norm", linew
plt.xscale('log')
plt.legend(loc='upper left')
plt.xlabel(r"$\lambda$", fontsize=16)
plt.ylabel("Error rate", fontsize=16)
plt.show()
```



SVM Summary

```
plt.figure(figsize=(12,6))
plt.plot(lambda_vals, train_error_svm_1, label="Train error svm - l_1 norm", linewic
plt.plot(lambda_vals, test_error_svm_1, label="Test error svm - l_1 norm", linewidtl
plt.plot(lambda_vals, train_error_svm_inf, label="Train error svm - l_inf norm", lir
plt.plot(lambda_vals, test_error_svm_inf, label="Test error svm - l_inf norm", linew
plt.hlines(test_error_svm, 0.01, 1, linestyle = "--", label="Test error SVM without
plt.xscale('log')
plt.legend(loc='upper left')
plt.xlabel(r"$\lambda$", fontsize=16)
plt.ylabel("Error rate", fontsize=16)
plt.title("Fig 2: Error rate of SVM for train and test sets with values of $\lambda$
plt.show()
```



```
1
```

Total results

```
plt.figure(figsize=(12,6))
plt.plot(lambda_vals, train_error_lgr_1, label="Train error lgr - l_1 norm", linewic
plt.plot(lambda_vals, test_error_lgr_1, label="Test error lgr - l_1 norm", linewidtl
plt.plot(lambda_vals, train_error_svm_1, label="Train error svm - l_1 norm", linewic
plt.plot(lambda_vals, test_error_svm_1, label="Test error svm - l_1 norm", linewidtl
plt.plot(lambda_vals, train_error_lgr_inf, label="Train error lgr - l_inf norm", lir
plt.plot(lambda_vals, test_error_lgr_inf, label="Test error lgr - l_inf norm", linew
plt.plot(lambda_vals, train_error_svm_inf, label="Train error svm - l_inf norm", lir
plt.plot(lambda_vals, test_error_svm_inf, label="Test error svm - l_inf norm", linew
plt.hlines(test_error_svm, 0.01, 1, linestyle = "--", label="Test error SVM without
plt.hlines(test_error_lgr, 0.01, 1, linestyle = "--", label="Test error LGR without
plt.xscale('log')
plt.legend(loc='upper left')
plt.xlabel(r"$\lambda$", fontsize=16)
plt.ylabel("Error rate", fontsize=16)
plt.title("Fig 3: Error rate of train and test sets with values of $\lambda$")
plt.show()
```

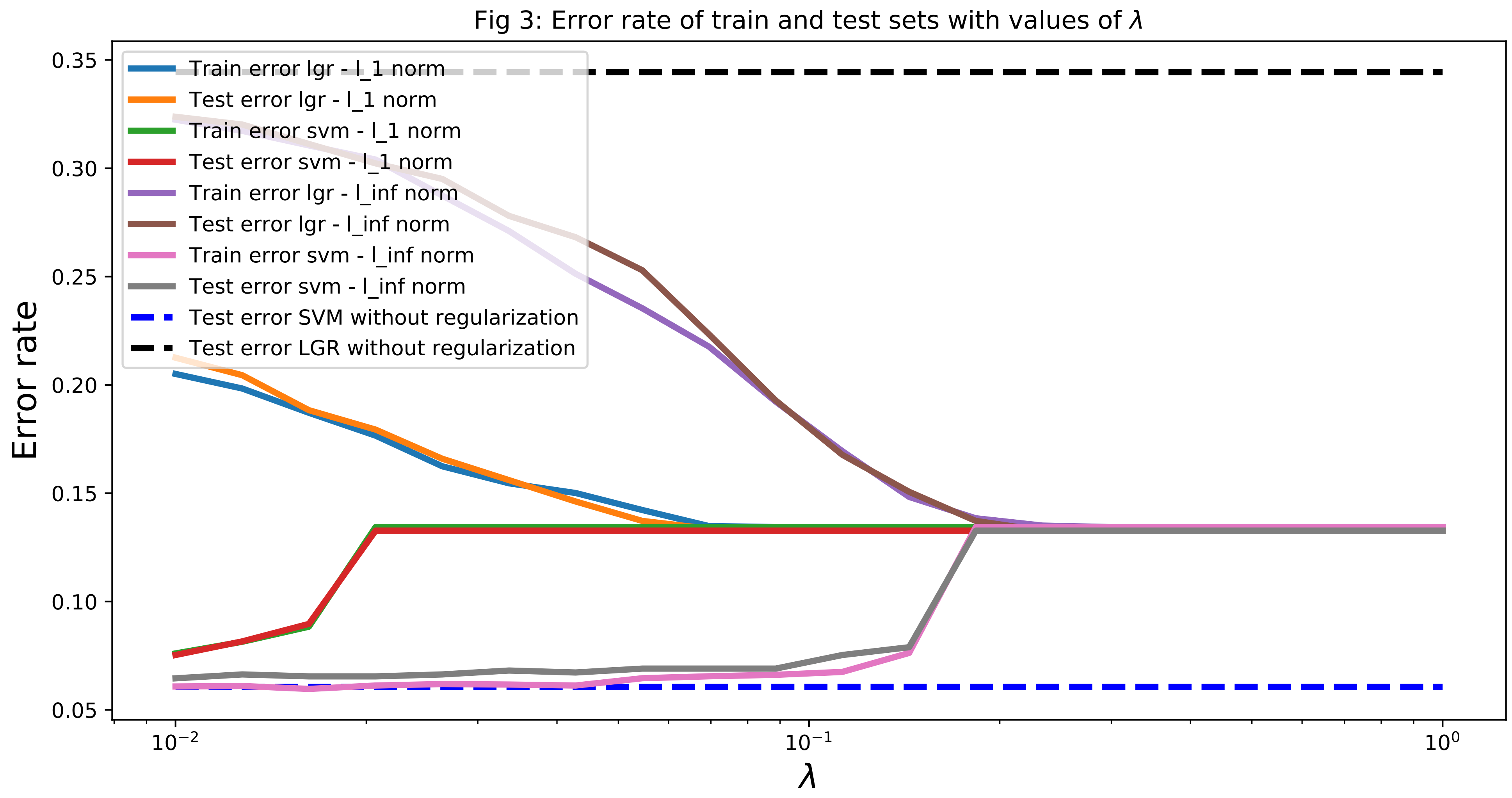


Fig 3: Error rate of train and test sets with values of λ
Train error lgr - l_1 norm
Test error lgr - l_1 norm
Train error svm - l_1 norm
Test error svm - l_1 norm
Train error lgr - l_inf norm
Test error lgr - l_inf norm
Train error svm - l_inf norm
Test error svm - l_inf norm
Test error SVM without regularization
Test error LGR without regularization
Error rate
0.35
0.30
0.25
0.20
0.15
0.10
0.05
10^{-2}
10^{-1}
10^{0}
λ

```
1
```