CS M146 Homework 2

Megan Pham - 505 313 300

Problem 1 Decision Trees

a) isPoisonous entropy:

$$-\frac{3}{8} \times \log\left(\frac{3}{8}\right) - \frac{5}{8}\log\left(\frac{5}{8}\right) = 0.53 + 0.423 = 0.954$$

b) isSmooth

c) from disc slides $I(X;Y) = H(Y) - H(Y|X)$

$$I(\text{isSmooth} | \text{isPoisonous}) = H(\text{poisonous}) - H(\text{Poisonous} | \text{smooth})$$

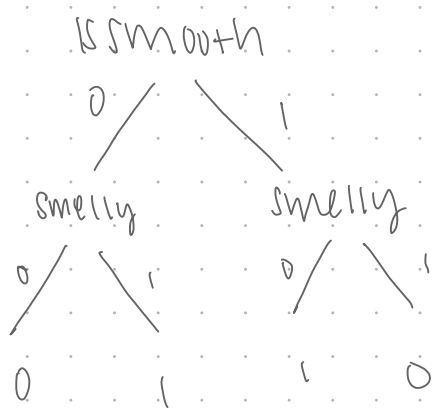
$$H(P|S) = \overset{(0)}{\frac{4}{8}}\left(-\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4}\right) + \overset{(1)}{\frac{4}{8}}\left(-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}\right)$$

$$= 0.5 + 0.4056$$

$$= 0.906$$

$$I(\text{is smooth}) = 0.954 - 0.906 = \boxed{0.0488}$$

d)



e) u: isSmooth = 1
isSmelly = 1
isPoisonous = 0
not poisonous

w: isSmooth = 0
isSmelly = 1
isPoisonous = 1
poisonous

v: isSmooth = 1
isSmelly = 1
isPoisonous = 0
not poisonous

Problem 2: entropy & information

entropy of Bernoulli

$$B(q) = -q\log q - (1-q)\log(1-q)$$

If $\frac{p_k}{p_k+n_k}$ is the same for all k, show that information gain of this attribute is 0

$$H(S) = B\left(\frac{p}{p+n}\right)$$

$$p = \sum p_k$$

$$n = \sum n_k$$

if $\frac{p_k}{p_k+n_k}$, then $\frac{p}{p+n}$

$$\text{gain} = H(S) - \sum_k \frac{S_k}{S} H(S_k)$$

$$= B\left(\frac{p}{p+n}\right) - B\left(\frac{p}{p+n}\right)\frac{1}{p+n}(p+n) = 0$$

Problem 3: k nearest neighbors

a) $k=0$ (assuming point can be its own neighbour) minimizes the training set error for this dataset because the training error is 0. the closest point to itself is itself.

b) point 1: − error = 0

point 2: + error = 0

point 3: + error = 0

point 4: − ← misclassified error = 1

training error = $\frac{1}{4}$

c) $k=1$ (as specified on campuswire) use cos sim and get MAX (not MIN)

point 1 $u = \langle 1, 5\rangle$ −
sim(u, <2,6>) = 0.992
sim(u, <2,7>) = 0.997 +
sim(u, <3,7>) = 0.979
sim(u, <3,8>) = 0.987

error = 1

point 2 $u = \langle 5, 1\rangle$ +
sim(u, <6,2>) = 0.992
sim(u, <7,2>) = 0.997 −
sim(u, <7,3>) = 0.979

error = 1

point 3 $u = \langle 9, 5\rangle$ +
sim(u, <8,4>) = 0.999 +
sim(u, <8,3>) = 0.989
sim(u, <7,3>) = 0.995
sim(u, <7,2>) = 0.974

error = 0

point 4 $u = \langle 4, 8\rangle$ −
sim(u, <3,8>) = 0.995
sim(u, <3,7>) = 0.998
sim(u, <9,9>) = 0.999 −
sim(u, <2,7>) = 0.983

error = 0

points 1 and 2 are misclassified

training error = $\frac{1}{2}$

Problem 4: kernel properties

valid kernel
$K(x, z) = (x^T z)^d$ where $x, z \in \mathbb{R}^n$, $d \in \mathbb{Z}^+$

a) prove $K_{new}(x,z) = \left(\sum_{i=1}^{n} \sqrt{x_i}\sqrt{z_i}\right)^d$ is valid

We can construct new kernels from K
let's use the hint.

If $K(x,z)$ is a valid kernel the $K_{new}(x,z) = K(g(x), g(z))$ is also a valid kernel

$$K_{new}(x,z) = \left(\sum_{i=1}^{n} \sqrt{x_i}\sqrt{z_i}\right)^d$$

From $K(x,z) = (x^T z)^d$

$$K_{new}(x,z) = ((g(x))^T (g(z)))^d$$

$g(x)^T = \sqrt{x}$
$g(z) = \sqrt{z}$

$g(x): \mathbb{R}^n \to \mathbb{R}^n$

K_{new} is a valid kernel because we were able to construct it from K and K is a valid kernel

b) $K_2(x,z) = \frac{K_0(x,z)}{\sqrt{K_0(x,x)}\sqrt{K_0(z,z)}}$ where K_0 is a valid kernel

To be valid, K_2 must be symmetric and positive definite.

We know K_0 is symmetric, so we can flip the inputs:

$$K_2(x,z) = \frac{K_0(x,z)}{\sqrt{K_0(x,x)}\sqrt{K_0(z,z)}} \iff K_2(z,x) = \frac{K_0(z,x)}{\sqrt{K_0(z,z)}\sqrt{K_0(x,x)}}$$

K_2 is symmetric

To be positive semi-definite, $u^T G u \geq 0 \quad \forall u \in \mathbb{R}^n$

since k_0 is valid, then k_0 must be positive semidefinite & symmetric, hence K_2 is also positive semidefinite.

K_2 is valid.

c) For $K(x,z) = K_0(\phi(x), \phi(z))$ to be a valid kernel, $\phi(x)$ and $\phi(z)$ must have the same input and output dimensions as they are the parameters for K_0. Therefore, $n = m$

Problem 5: Bias-variance Tradeoff

a) code
b) code
c) code
d) code
e) code

Problem 6: programming exercise - applying decision trees and k-nearest neighbors

a Ticket class is not that indicative of who survived, though higher ticket classes had more casualties

No labels for sex class, but one sex had more survivors than the other sex.

Age: More middle aged adults onboard, hence the higher number of casualties/survivors

sibsp: A lot of people with no siblings were onboard.

parch: There were a lot of single people on board, but it seems like families were prioritized

fare: People who paid more were more likely to survived.

embarked: people from cherbourg had more survivors than Queenstown and southampton.

b) code

c) error = 0.014

d) errors:
k=3 → 0.169
k=5 → 0.202
k=7 → 0.242

e) Majority vote

avg training error = 0.404
avg testing error = 0.407

Random

avg training error = 0.489
avg testing error = 0.497

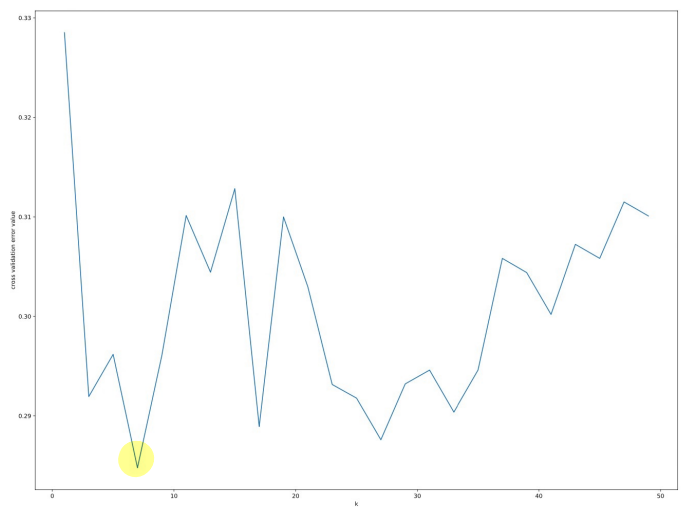
Decision tree

avg training error = 0.012
avg testing error = 0.241

K-nearest neighbors

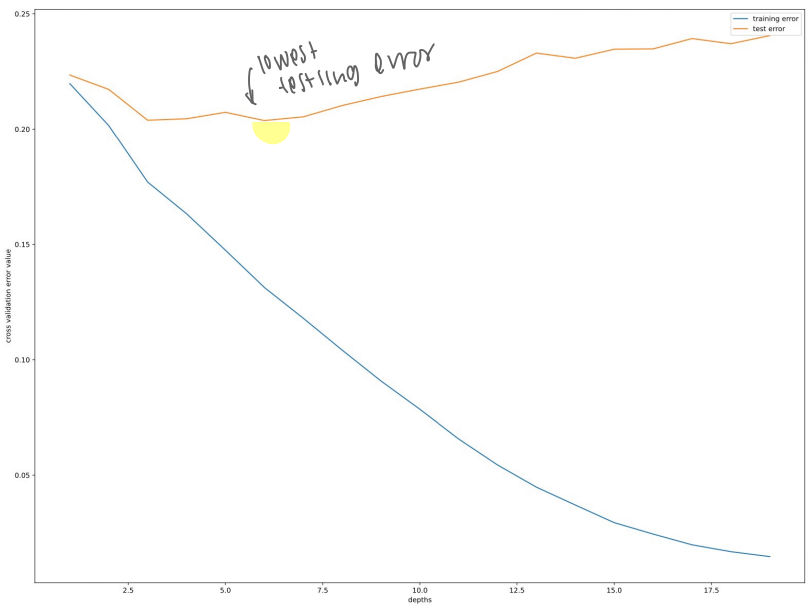
avg training error = 0.213
avg testing error = 0.315

f) The best value is k=7 with a cross validation error of 0.285



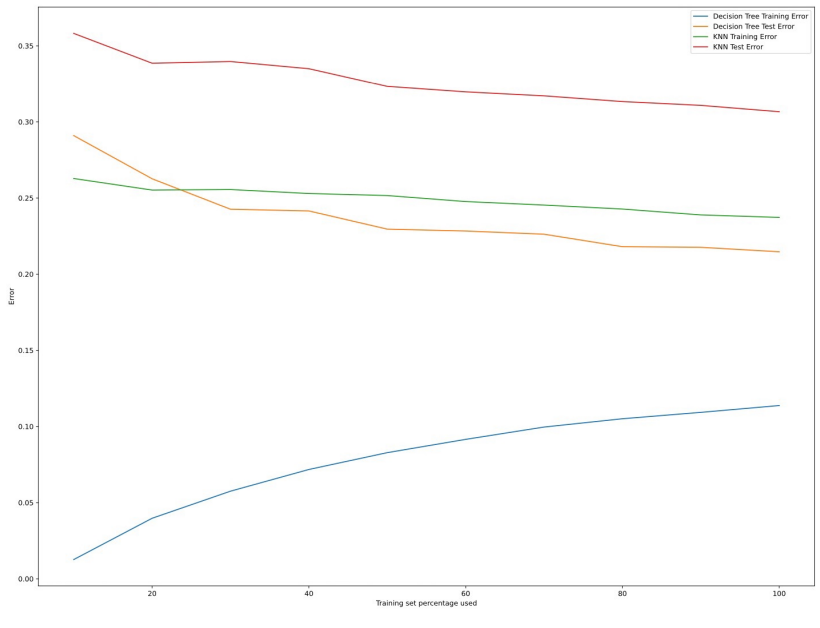
We see that cross validation error is high when k is too little or too big, which reminds us of underfitting/overfitting in the previous homework. The best k value is 7.

g) The best depth value is d=7 with a testing error of 0.209.



We see that there is overfitting when depths get higher for the testing error as the testing error increases and training error decreases.

h)



We see that the error decreases as we get more training data. However, my decision tree training error actually increases and then plateaus as we get more training data.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.datasets import load_breast_cancer
%matplotlib inline

breast_cancer = load_breast_cancer()
X, y = breast_cancer.data, breast_cancer.target

###
#IMPORT LOGISTIC REGRESSION MODEL USING sklearn
###
from sklearn.linear_model import LogisticRegression
###
# Define the model (SHOULD NOT BE MORE THAN 1 LINE OF CODE)
###
model= LogisticRegression(random_state=0)
train_size, train_score, validation_score =
learning_curve(estimator=model, X=X, y=y, cv=5)

train_score_m_LR = np.mean(train_score, axis=1)
validation_score_m_LR = np.mean(validation_score, axis=1)
```

```
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
```

```
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

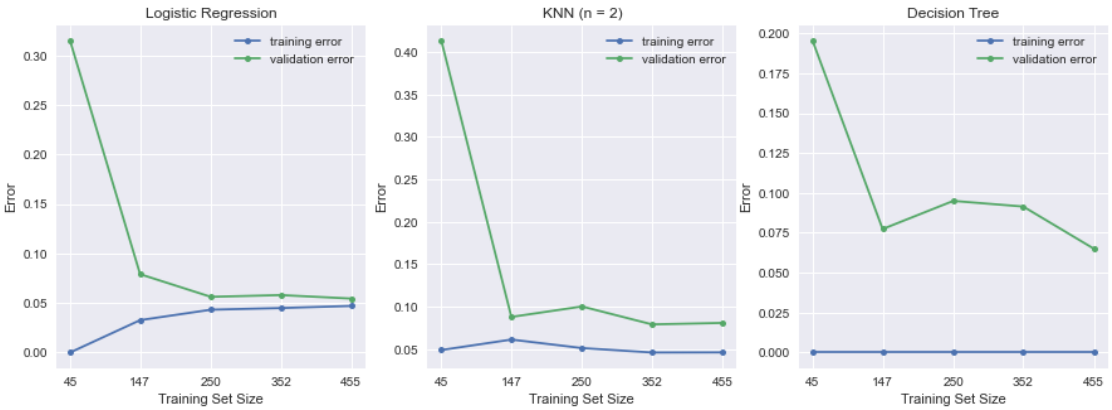
```
Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
```

```
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

```
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/meganpham/opt/anaconda3/lib/python3.9/site-packages/sklearn/
linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

###
#IMPORT KNN CLASSIFIER MODEL USING sklearn
###
from sklearn.neighbors import KNeighborsClassifier
###
# Define the 2-NN model (SHOULD NOT BE MORE THAN 1 LINE OF CODE)
###

model = KNeighborsClassifier(n_neighbors=3)
train_size, train_score, validation_score =
learning_curve(estimator=model, X=X, y=y, cv=5)

train_score_m_KNN2 = np.mean(train_score, axis=1)
validation_score_m_KNN2 = np.mean(validation_score, axis=1)

###
#IMPORT DECISION TREE CLASSIFIER USING sklearn
###
from sklearn.tree import DecisionTreeClassifier
###
# Define the DECISION TREE CLASSIFIER model (SHOULD NOT BE MORE THAN 1
LINE OF CODE)
###

model = DecisionTreeClassifier(criterion='entropy')
train_size, train_score, validation_score =
learning_curve(estimator=model, X=X, y=y, cv=5)

train_score_m_DT = np.mean(train_score, axis=1)
validation_score_m_DT = np.mean(validation_score, axis=1)

###
# Convert training and validation scores to training and validation
errors here.
# Run the below piece of code to get your final learning curves.
###

error_train_score_m_LR = 1 -train_score_m_LR
error_validation_score_m_LR = 1 -validation_score_m_LR

error_train_score_m_KNN2 = 1 - train_score_m_KNN2
error_validation_score_m_KNN2 = 1 -validation_score_m_KNN2

error_train_score_m_DT = 1 -train_score_m_DT
```

```
error_validation_score_m_DT = 1 -validation_score_m_DT


plt.style.use('seaborn')

f, ((ax11, ax12, ax13)) = plt.subplots(1, 3, figsize=(15,5))
X_axis = train_size
# Y_axis = [0, 0.1, 0.2, 0.3, 0.4, 0.5]

ax11.set_xlabel('Training Set Size')
ax11.set_ylabel('Error')
ax11.set_xticks(X_axis)
# ax11.set_yticks(Y_axis)
ax11.set_title('Logistic Regression')
ax11.plot(X_axis, error_train_score_m_LR, 'o-', markersize=5,
label="training error")
ax11.plot(X_axis, error_validation_score_m_LR, 'o-', markersize=5,
label="validation error")
ax11.legend()

ax12.set_xlabel('Training Set Size')
ax12.set_ylabel('Error')
ax12.set_xticks(X_axis)
# ax12.set_yticks(Y_axis)
ax12.set_title('KNN (n = 2)')
ax12.plot(X_axis, error_train_score_m_KNN2, 'o-', markersize=5,
label="training error")
ax12.plot(X_axis, error_validation_score_m_KNN2, 'o-', markersize=5,
label="validation error")
ax12.legend()

ax13.set_xlabel('Training Set Size')
ax13.set_ylabel('Error')
ax13.set_title('Decision Tree')
ax13.set_xticks(X_axis)
# ax13.set_yticks(Y_axis)
ax13.plot(X_axis, error_train_score_m_DT, 'o-', markersize=5,
label="training error")
ax13.plot(X_axis, error_validation_score_m_DT, 'o-', markersize=5,
label="validation error")
ax13.legend()

plt.show()
```


Logistic Regression
training error
validation error
Error
Training Set Size
KNN (n = 2)
training error
validation error
Error
Training Set Size
Decision Tree
training error
validation error
Error
Training Set Size

```
"""
Author      : Yi-Chieh Wu, Sriram Sankararaman
Description : Titanic
"""

# Use only the provided packages!
import math
import csv
from util import *
from collections import Counter


from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn import metrics

######################################################################
# classes
######################################################################

class Classifier(object) :
    """
    Classifier interface.
    """

    def fit(self, X, y):
        raise NotImplementedError()

    def predict(self, X):
        raise NotImplementedError()


class MajorityVoteClassifier(Classifier) :

    def __init__(self) :
        """
        A classifier that always predicts the majority class.

        Attributes
        --------------------
            prediction_ -- majority class
        """
        self.prediction_ = None

    def fit(self, X, y) :
        """
        Build a majority vote classifier from the training set (X, y).

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples
            y    -- numpy array of shape (n,), target classes

        Returns
        --------------------
            self -- an instance of self
        """
        majority_val = Counter(y).most_common(1)[0][0]
        self.prediction_ = majority_val
        return self

    def predict(self, X) :
        """
        Predict class values.
```

```
        Parameters
        ----------
           X    -- numpy array of shape (n,d), samples

        Returns
        ----------
           y    -- numpy array of shape (n,), predicted classes
        """
        if self.prediction_ is None :
           raise Exception("Classifier not initialized. Perform a fit first.")

        n,d = X.shape
        y = [self.prediction_] * n
        return y


class RandomClassifier(Classifier) :

    def __init__(self) :
        """
        A classifier that predicts according to the distribution of the classes.

        Attributes
        ----------
           probabilities_ -- class distribution dict (key = class, val = probability of class)
        """
        self.probabilities_ = None

    def fit(self, X, y) :
        """
        Build a random classifier from the training set (X, y).

        Parameters
        ----------
           X    -- numpy array of shape (n,d), samples
           y    -- numpy array of shape (n,), target classes

        Returns
        ----------
           self -- an instance of self
        """

        ### ========== TODO : START ========== ###
        # part b: set self.probabilities_ according to the training set
        random_val = Counter(y)
        self.probabilities_ = [random_val[0]/float(y.shape[0]) , random_val[1]/float(y.shape[0])]

        ### ========== TODO : END ========== ###

        return self

    def predict(self, X, seed=1234) :
        """
        Predict class values.

        Parameters
        ----------
           X    -- numpy array of shape (n,d), samples
           seed -- integer, random seed

        Returns
        ----------
           y    -- numpy array of shape (n,), predicted classes
        """
        if self.probabilities_ is None :
```

```
            raise Exception("Classifier not initialized. Perform a fit first.")
        np.random.seed(seed)

        ### ========== TODO : START ========== ###
        # part b: predict the class for each test example
        # hint: use np.random.choice (be careful of the parameters)

        y = np.random.choice([0,1], size=X.shape[0], replace=True, p=self.probabilities_)

        ### ========== TODO : END ========== ###

        return y


##############################################################################
# functions
##############################################################################
def plot_histograms(X, y, Xnames, yname) :
    n,d = X.shape  # n = number of examples, d =  number of features
    fig = plt.figure(figsize=(20,15))
    nrow = 3; ncol = 3
    for i in range(d) :
        fig.add_subplot (3,3,i+1)
        data, bins, align, labels = plot_histogram(X[:,i], y, Xname=Xnames[i], yname=yname, show = False)
        n, bins, patches = plt.hist(data, bins=bins, align=align, alpha=0.5, label=labels)
        plt.xlabel(Xnames[i])
        plt.ylabel('Frequency')
        plt.legend() #plt.legend(loc='upper left')

    plt.savefig ('histograms.pdf')



def plot_histogram(X, y, Xname, yname, show = True) :
    """
    Plots histogram of values in X grouped by y.

    Parameters
    ----------
        X     -- numpy array of shape (n,d), feature values
        y     -- numpy array of shape (n,), target classes
        Xname -- string, name of feature
        yname -- string, name of target
    """

    # set up data for plotting
    targets = sorted(set(y))
    data = []; labels = []
    for target in targets :
        features = [X[i] for i in range(len(y)) if y[i] == target]
        data.append(features)
        labels.append('%s = %s' % (yname, target))

    # set up histogram bins
    features = set(X)
    nfeatures = len(features)
    test_range = list(range(int(math.floor(min(features))), int(math.ceil(max(features)))+1))
    if nfeatures < 10 and sorted(features) == test_range:
        bins = test_range + [test_range[-1] + 1] # add last bin
        align = 'left'
    else :
        bins = 10
        align = 'mid'

    # plot
    if show == True:
```

```
        plt.figure()
        n, bins, patches = plt.hist(data, bins=bins, align=align, alpha=0.5, label=labels)
        plt.xlabel(Xname)
        plt.ylabel('Frequency')
        plt.legend() #plt.legend(loc='upper left')
        plt.show()

    return data, bins, align, labels


def error(clf, X, y, ntrials=100, test_size=0.2) :
    """
    Computes the classifier error over a random split of the data,
    averaged over ntrials runs.

    Parameters
    --------------------
        clf         -- classifier
        X           -- numpy array of shape (n,d), features values
        y           -- numpy array of shape (n,), target classes
        ntrials     -- integer, number of trials

    Returns
    --------------------
        train_error -- float, training error
        test_error  -- float, test error
    """

    ### ========== TODO : START ========== ###
    # compute cross-validation error over ntrials
    # hint: use train_test_split (be careful of the parameters)

    train_error = 0
    test_error = 0

    for i in range(0, ntrials):
        X_train, X_test, y_train, y_test = train_test_split (X, y, test_size= test_size, random_state=i )
        clf.fit(X_train, y_train)
        y_train_pred = clf.predict(X_train)
        train_error += 1-metrics.accuracy_score(y_train, y_train_pred, normalize=True)
        y_test_pred = clf.predict(X_test)
        test_error += 1-metrics.accuracy_score(y_test, y_test_pred, normalize=True)

    train_error = train_error/ntrials
    test_error = test_error/ntrials

    ### ========== TODO : END ========== ###

    return train_error, test_error

def error_h(clf, X, y, ntrials=100, test_size=0.2, inc=10) :
    """
    Computes the classifier error over a random split of the data,
    averaged over ntrials runs.

    Parameters
    --------------------
        clf         -- classifier
        X           -- numpy array of shape (n,d), features values
        y           -- numpy array of shape (n,), target classes
        ntrials     -- integer, number of trials

    Returns
    --------------------
        train_error -- float, training error
        test_error  -- float, test error
    """
```

```
    ### ========== TODO : START ========== ###
    # compute cross-validation error over ntrials
    # hint: use train_test_split (be careful of the parameters)

    train_error = 0
    test_error = 0
    inc = inc/10.0

    for i in range(0, ntrials):
        X_train, X_test, y_train, y_test = train_test_split (X, y, test_size= test_size, random_state=i )
        X_train = X_train[:int(len(X_train)*inc)]
        y_train = y_train[:int(len(y_train)*inc)]
        clf.fit(X_train, y_train)
        y_train_pred = clf.predict(X_train)
        train_error += 1-metrics.accuracy_score(y_train, y_train_pred, normalize=True)
        y_test_pred = clf.predict(X_test)
        test_error += 1-metrics.accuracy_score(y_test, y_test_pred, normalize=True)

    train_error = train_error/ntrials
    test_error = test_error/ntrials

    ### ========== TODO : END ========== ###

    return train_error, test_error


def write_predictions(y_pred, filename, yname=None) :
    """Write out predictions to csv file."""
    out = open(filename, 'wb')
    f = csv.writer(out)
    if yname :
        f.writerow([yname])
    f.writerows(list(zip(y_pred)))
    out.close()


###########################################################################
# main
###########################################################################

def main():
    # load Titanic dataset
    titanic = load_data("../data/titanic_train.csv", header=1, predict_col=0)
    X = titanic.X; Xnames = titanic.Xnames
    y = titanic.y; yname = titanic.yname
    n,d = X.shape  # n = number of examples, d =  number of features



    #========================================
    # part a: plot histograms of each feature
    print('Plotting...')
    plot_histograms(X, y, Xnames=Xnames, yname=yname)


    #========================================
    # train Majority Vote classifier on data
    print('Classifying using Majority Vote...')
    mclf = MajorityVoteClassifier() # create MajorityVote classifier, which includes all model parameters
    mclf.fit(X, y)                  # fit training data using the classifier
    my_pred = mclf.predict(X)       # take the classifier and run it on the training data
    train_error = 1 - metrics.accuracy_score(y, my_pred, normalize=True)
    print('\t-- training error: %.3f' % train_error)
```

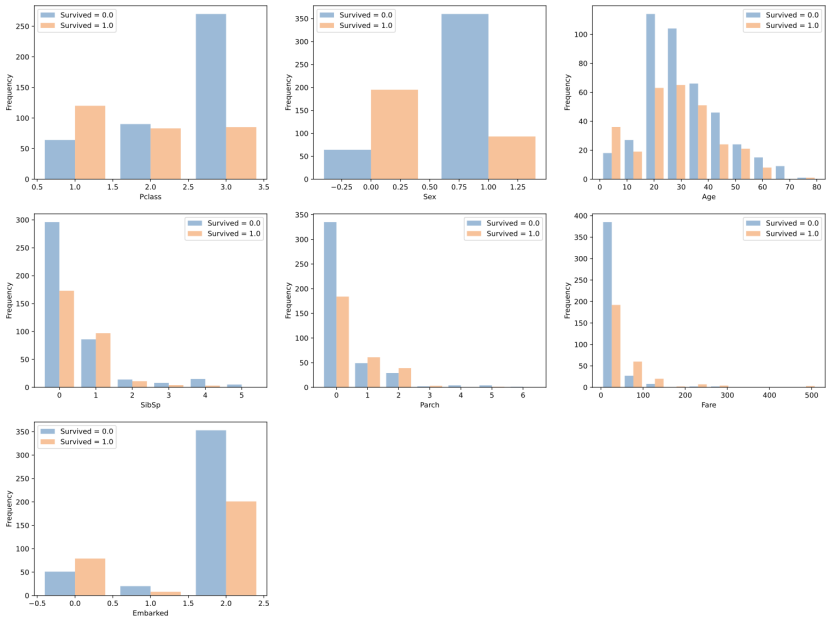
```
### ========== TODO : START ========== ###
# part b: evaluate training error of Random classifier
print('Classifying using Random...')
rclf = RandomClassifier() # create MajorityVote classifier, which includes all model parameters
rclf.fit(X, y)                # fit training data using the classifier
ry_pred = rclf.predict(X)       # take the classifier and run it on the training data
train_error = 1 - metrics.accuracy_score(y, ry_pred, normalize=True)
print('\t-- training error: %.3f' % train_error)

### ========== TODO : END ========== ###



### ========== TODO : START ========== ###
# part c: evaluate training error of Decision Tree classifier
# use criterion of "entropy" for Information gain
print('Classifying using Decision Tree...')
dclf = DecisionTreeClassifier(criterion="entropy") # create MajorityVote classifier, which includes all model parameters
dclf.fit(X, y)                # fit training data using the classifier
dy_pred = dclf.predict(X)       # take the classifier and run it on the training data
train_error = 1 - metrics.accuracy_score(y, dy_pred, normalize=True)
print('\t-- training error: %.3f' % train_error)

### ========== TODO : END ========== ###



# note: uncomment out the following lines to output the Decision Tree graph

# save the classifier -- requires GraphViz and pydot
"""
import StringIO, pydot
from sklearn import tree
dot_data = StringIO.StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                     feature_names=Xnames)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("dtree.pdf")
"""



### ========== TODO : START ========== ###
# part d: evaluate training error of k-Nearest Neighbors classifier
# use k = 3, 5, 7 for n_neighbors
print('Classifying using k-Nearest Neighbors...')
kclf = KNeighborsClassifier(n_neighbors=3) # create MajorityVote classifier, which includes all model parameters
kclf.fit(X, y)                # fit training data using the classifier
ky_pred = kclf.predict(X)       # take the classifier and run it on the training data
train_error = 1 - metrics.accuracy_score(y, ky_pred, normalize=True)
print('\t-- training error for k=3: %.3f' % train_error)

k5clf = KNeighborsClassifier(n_neighbors=5) # create MajorityVote classifier, which includes all model parameters
k5clf.fit(X, y)                # fit training data using the classifier
k5y_pred = k5clf.predict(X)       # take the classifier and run it on the training data
train_error = 1 - metrics.accuracy_score(y, k5y_pred, normalize=True)
print('\t-- training error for k=5: %.3f' % train_error)

kclf = KNeighborsClassifier(n_neighbors=7) # create MajorityVote classifier, which includes all model parameters
kclf.fit(X, y)                # fit training data using the classifier
ky_pred = kclf.predict(X)       # take the classifier and run it on the training data
train_error = 1 - metrics.accuracy_score(y, ky_pred, normalize=True)
print('\t-- training error for k=7: %.3f' % train_error)

### ========== TODO : END ========== ###
```

```
### ========== TODO : START ========== ###
# part e: use cross-validation to compute average training and test error of classifiers
print('Investigating various classifiers...')

mclf_error = error(mclf, X, y)
print('Majority Vote for training and test error...')
print('\t-- average training error: %.3f' % mclf_error[0])
print('\t-- average testing error: %.3f' % mclf_error[1])

rclf_error = error(rclf, X, y)
print('Random for training and test error...')
print('\t-- average training error: %.3f' % rclf_error[0])
print('\t-- average testing error: %.3f' % rclf_error[1])

dclf_error = error(dclf, X, y)
print('Decision Tree for training and test error...')
print('\t-- average training error: %.3f' % dclf_error[0])
print('\t-- average testing error: %.3f' % dclf_error[1])

k5clf_error = error(k5clf, X, y)
print('k-Nearest Neighbors (k=5) for training and test error...')
print('\t-- average training error: %.3f' % k5clf_error[0])
print('\t-- average testing error: %.3f' % k5clf_error[1])



### ========== TODO : END ========== ###



### ========== TODO : START ========== ###
# part f: use 10-fold cross-validation to find the best value of k for k-Nearest Neighbors classifier
print('Finding the best k for KNeighbors classifier...')
plt.clf()
k_val = []
scores_err = []

for i in range(1, 51, 2):
    knclf = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score (knclf, X, y, cv=10)
    print('\t-- cross val score for k=%d: %.3f' % (i, 1-np.mean(scores)))
    scores_err.append(1-np.mean(scores))
    k_val.append(i)

plt.xlabel("k")
plt.ylabel('cross validation error value')
plt.plot(k_val, scores_err)

plt.savefig ('kneighbors_cross_val.pdf')



### ========== TODO : END ========== ###



### ========== TODO : START ========== ###
# part g: investigate decision tree classifier with various depths
print('Investigating depths...')
plt.clf()
d_val = []
train_err = []
test_err = []
```

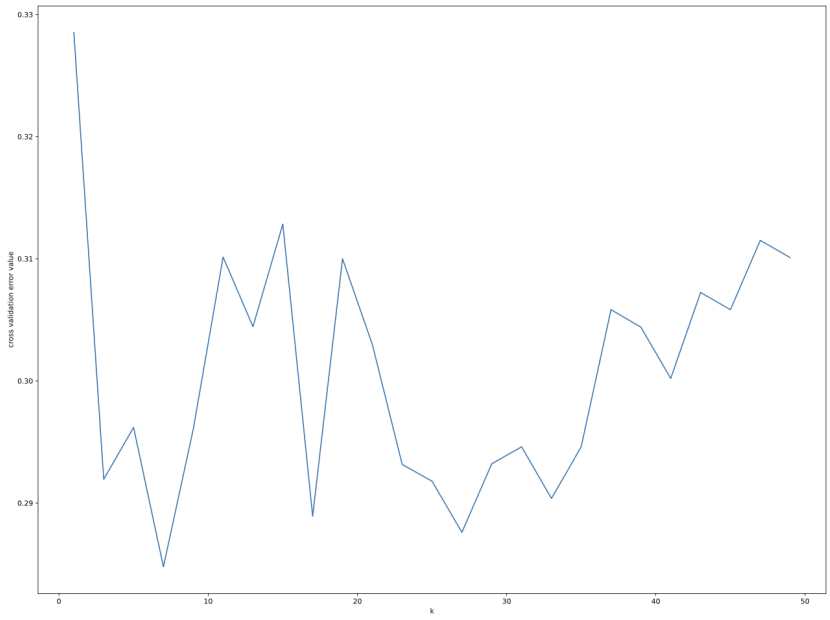
```
for i in range(1,21):
    ddclf = DecisionTreeClassifier(criterion='entropy', max_depth=i)
    train_error, test_error = error(ddclf, X, y)

    train_err.append(train_error)
    test_err.append(test_error)
    print('\t-- cross val score for k=%d: %.3f' % (i, np.mean(test_err)))
    d_val.append(i)

plt.xlabel("depths")
plt.ylabel('cross validation error value')
plt.plot(d_val, train_err, label ='training error')
plt.plot(d_val, test_err, label ='test error')
plt.legend()

plt.savefig ('decisiontree_cross_val.pdf')



### ========== TODO : END ========== ###



### ========== TODO : START ========== ###
# part h: investigate Decision Tree and k-Nearest Neighbors classifier with various training set sizes
print('Investigating training set sizes...')
plt.clf()
hknclf = KNeighborsClassifier(n_neighbors=7)
hdclf = DecisionTreeClassifier(max_depth=7)

hd_training_error = []
hd_test_error = []
hkn_training_error = []
hkn_test_error = []

x_vals = []
for i in range(1, 11, 1):
    hd_err = error_h(hdclf, X, y, 100, 0.1, i)
    hkn_err = error_h(hknclf, X, y, 100, 0.1, i)

    hd_training_error.append(hd_err[0])
    hd_test_error.append(hd_err[1])
    hkn_training_error.append(hkn_err[0])
    hkn_test_error.append(hkn_err[1])
    x_vals.append(i*10)


    print('    Using %g percent of training data...' % (i*10))
    print('\t decision tree...')
    print ('\t\t training error: %.3f  and testing error: %.3f' % (hd_err[0], hd_err[1]))
    print('\t k neighbors...')
    print ('\t\t training error: %.3f and testing error: %.3f' % (hkn_err[0], hkn_err[1]))

plt.xlabel ("Training set percentage used")
plt.ylabel('Error')
plt.plot(x_vals, hd_training_error, label='Decision Tree Training Error' )
plt.plot(x_vals, hd_test_error, label='Decision Tree Test Error' )

plt.plot(x_vals, hkn_training_error, label='KNN Training Error' )
plt.plot(x_vals, hkn_test_error, label='KNN Test Error' )
plt.legend()

plt.savefig("part_h.pdf")
```

```
    ### ========== TODO : END ========== ###


    print('Done')


if __name__ == "__main__":
    main()
```

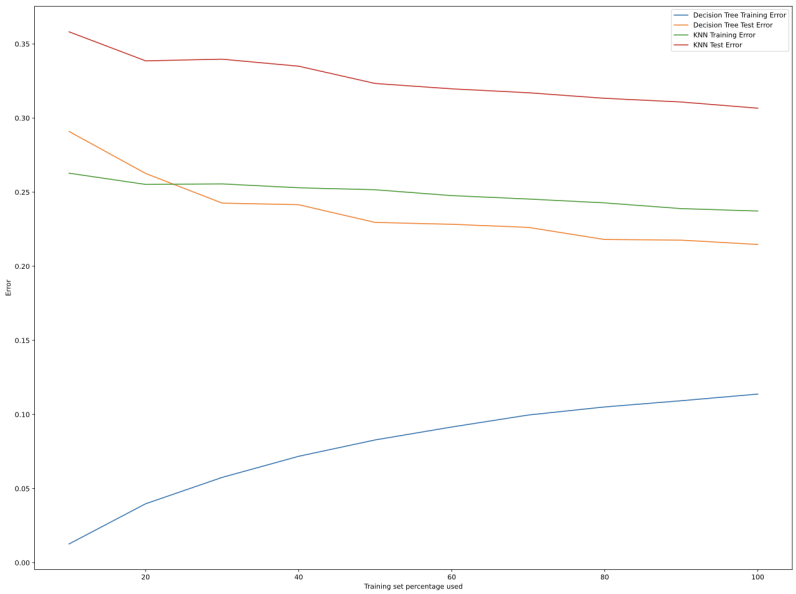
histograms.pdf
1 page



kneighbors_cross_val.pdf
1 page



part_h.pdf
1 page



decisiontree_cross_val.pdf
1 page
Search