دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

پایان‌نامه کارشناسی‌ارشد
گرایش ریاضی

هوش‌مصنوعی - گزارش ۰۷ - پیاده‌سازی و مقایسه‌ی الگوریتم‌های طبقه‌بندی

پایان‌نامه

نگارش
آترین حجت

استاد راهنما
نام کامل استاد راهنما

استاد مشاور
نام کامل استاد مشاور

فروردین ۱۴۰۰


بسم الله الرحمن الرحيم

صفحه فرم ارزیابی و تصویب پایان نامه- فرم تأیید اعضاء کمیته دفاع

در این صفحه فرم دفاع یا تایید و تصویب پایان نامه موسوم به فرم کمیته دفاع- موجود در پرونده آموزشی- را قرار دهید.

نکات مهم:

- نگارش پایان نامه/رساله باید به زبان فارسی و بر اساس آخرین نسخه دستورالعمل و راهنمای تدوین پایان نامه های دانشگاه صنعتی امیرکبیر باشد.(دستورالعمل و راهنمای حاضر)
- رنگ جلد پایان نامه/رساله چاپی کارشناسی، کارشناسی ارشد و دکترا باید به ترتیب مشکی، طوسی و سفید رنگ باشد.
- چاپ و صحافی پایان نامه/رساله بصورت پشت و رو(دورو) بلامانع است و انجام آن توصیه می شود.

دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

به نام خدا

تعهدنامه اصالت اثر

تاریخ: فروردین ۱۴۰۰

اینجانب **آترین حجت** متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

آترین حجت

امضا

فهرست مطالب

عنوان صفحه

فهرست اشکال

شکل صفحه

فهرست جداول

جدول صفحه

فهرست نمادها

نماد	مفهوم
$n(G)$	تعداد رئوس گراف G
$deg_G(v)$	درجه‌ی راس v در گراف G
$nei(v)$	همسایه‌های راس v در گراف

فصل اول

شرح مسئله و روند کار

۱-۱ مقدمه

در این گزارش عمکرد روش‌های مختلف طبقه‌بندی را روی یک دیتاست مقایسه می‌کنیم. پیاده سازی و کد در اینجا قابل مشاهده می‌باشد. برای پیاده سازی از Python و کتاب‌خانه‌های numpy, sklearn, matplotlib, pandas استفاده شده.

۱-۲ روش اجرا

برای اجرای برنامه نیاز به Python با حداقل ورژن 3.8 دارید.

برای اجرای برنامه ابتدا نیاز به راه اندازی یک Virtual environment برای Python است. به این منظور در فولدر Report 07/codes دستورات زیر را اجرا کنید.

```
cd Report\\ 06/Codes
python3 -m venv venv
```

برای activate کردن با توجه به سیتم عامل دستورات اینجا [۱] را اجرا کنید.

برای نصب پیشنیاز ها دستورات زیر را اجرا کنید.

```
python3 -m pip install -r requirements.txt
```

برای اجرای کد نیاز به دانلود دیتاست دارید . مکان از دیتاست را یا باشد variable environment با نام DATASET_FILE و یا در هنگام اجرای برنامه وارد کنید.

با اجرای فایل Loader.py می‌توانید تستّهای برنامه را اجرا کنید.

```
python3 Loader.py
```

[۱] برای جزئیات بیشتر به اینجا مراجعه کنید

فصل دوم

دیتاست

هدف این دیتاست تشخیص قارچ‌های خوراکی از سمی می‌باشد. اطلاعات آن شامل شکل، جنس، بو، و... قارچ‌ها می‌باشد. [1] برچست قارچ‌های خوراکی e و قارچ‌های سمی p می‌باشد. برچسب‌های دیگز اطلاعات:

cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s

cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s

cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y

bruises: bruises=t,no=f

odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s

gill-attachment: attached=a,descending=d,free=f,notched=n

gill-spacing: close=c,crowded=w,distant=d

gill-size: broad=b,narrow=n

gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=

stalk-shape: enlarging=e,tapering=t

stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?

stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s

stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s

stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

veil-type: partial=p,universal=u

veil-color: brown=n,orange=o,white=w,yellow=y

ring-number: none=n,one=o,two=t

ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z

spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y

population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y

habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

[1] Dataset

۲-۰-۱ تبدیل دسته‌بندی‌ها به اعداد

برای تبدیل دسته‌های دوتایی به اعداد صرفا یک مقدار را برابر ۰ و دیگری را برابر ۱ قرار دادیم. راه‌حل مشابهی برای دسته‌های بزرگ‌تر از ۲ امکان پذیر نمی‌باشد زیرا به دسته‌ها اطلاعات نامربوط اضافه می‌شود. مثلا اگر برای دسته بندی سه رنگ قرمز، آبی و زرد اعداد ۰ تا ۲ را استفاده کنیم، ممکن است نتایجی مانند : قرمز بزرگ‌تر از آبی است داشته باشیم. به منظور رفع این مشکل به ازای هر دسته یک ستون جدید اضافه می‌کنیم.

سپس باید label ها را جدا کرده و تعدادی از ورودی‌ها را به عنوان تست انتخاب کنیم.

این بخش در کد با کمک Pandas و SKLearn انجام می‌شود.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import os



def LoadMushrooms(filename=None):
    if filename is None:
        filename =os.environ.get('DATASET_FILE', None)

    if filename is None:
        filename =input("Enter dataset file location: ")

    mushrooms_raw =pd.read_csv(filename)

    encoder =OneHotEncoder(drop="if_binary")
    mushrooms =encoder.fit_transform(mushrooms_raw).toarray()

    categorized_columns =[]

    for i in range(len(mushrooms_raw.columns)):
        column_name =mushrooms_raw.columns[i]
```

```
    if len(encoder.categories_[i]) >2:
        for val in encoder.categories_[i]:
            categorized_columns.append(f"{column_name}_{val}")
    else:
        categorized_columns.append(f"{column_name}")
print(encoder.categories_)

mushrooms =pd.DataFrame(data=mushrooms, columns=categorized_columns)

input_data =mushrooms.drop(['class'], axis=1)
output_data =mushrooms['class']


X_train, X_test, y_train, y_test =train_test_split(input_data,
                                        output_data,
        test_size=0.2, random_state=1)

return X_train, X_test, y_train, y_test, mushrooms, input_data,
                                        output_data
```

فصل سوم

الگوریتم‌ها

برای پیاده‌سازی الگوریتم‌ها از کتابخانه‌ی SKLearn استفاده‌شده.

۳-۱ Model

کلاس Model یک Interface برای پیاده‌سازی الگوریتم‌هاست که دو تابع train و predict را معرفی‌میکند. همه‌ی Classifier ها از این کلاس ارث‌بری می‌کنند. برای بررسی دقت و صحت از $sklearn.metrics.classification\_report$ استفاده شده است.

۳-۲ Decision Tree

برای Decision Tree از sklearn.tree.DecisionTree استفاده شده. بغیر از توابع __init__ ، train و predict یک تابع برای نمایش دادن درخت خروجی نیز پیاده‌سازی شده.

```
class DecisionTree(Model):
   def __init__(self, X, y, name, feature_names, criterion="gini",
                                          splitter="best",
         max_depth=2, min_samples_leaf=1):
      self.X =X
      self.y =y
      self.model =DecisionTreeClassifier(criterion=criterion,
                                             splitter=splitter,
            max_depth=max_depth, min_samples_leaf=min_samples_leaf)
      self.name =name
      self.feature_names =feature_names

   def train(self):
      self.model.fit(self.X, self.y)

   def predict(self, data):
      return self.model.predict(data)
```

تابع save_output درخت نهایی را با استفاده از sklearn.tree.plot_tree در یک

figure با matplotlib نمایش می‌دهد و خروجی را در یک فایم در پوشه‌ی Report 07/code/output با فرمت jpg زخیره می‌کند. اگر مقدار Environment variable با نام LATEX_OUTPUT برابر یک باشد، بجای نمایش‌دادن یک فایل pfg خروجی می‌دهد.

```
def save_output(self):
    plt.figure()
    fig, axes =plt.subplots(nrows=1, ncols=1,
          figsize=(10, 10), dpi=900)
    dec_tree =plot_tree(decision_tree=self.model,
                                          feature_names=self.feature_names,
                filled=True , precision=4, rounded=True, ax=axes)
    if not os.path.exists(os.path.dirname(os.path.join("./output",
       f"{self.name}.jpg"))):
       try: os.makedirs(os.path.dirname(os.path.join("./output",
          f"{self.name}.jpg")))
       except OSError as exc: # Guard against race condition
          if exc.errno !=errno.EEXIST:
             raise
    plt.savefig(os.path.join("./output", f"{self.name}.jpg"))
    if os.environ.get('LATEX_OUTPUT', '0') =='1':
       matplotlib.rcParams.update({
          "pgf.texsystem": "xelatex",
          'text.usetex': True,
          'pgf.rcfonts': False,
          "font.family": "mononoki Nerd Font Mono",
          "font.serif": [],
          # "font.cursive": ["mononoki Nerd Font", "mononoki Nerd Font
                                                Mono"],
       })
       plt.savefig(os.path.join("./output", f"{self.name}.pgf"))
    plt.show()
```

۳-۲-۱ خروجی

در شکل بخش ۳-۲-۱ یک نمونه‌ی خروجی `Decision Tree` با حداکثر عمق ۶ نمایش داده شده. دقت و صحت این الگوریتم در جدول جدول ۳-۱ ، جدول ۳-۲ و جدول ۳-۳ نشان داده شده.

Table 3-1: Decision tree accuracy measures with max depth 6

	precision	recall	f1-score	support
e	1.00	1.00	1.00	820
p	1.00	1.00	1.00	805
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Table 3-2: Decision tree accuracy measures with max depth 3

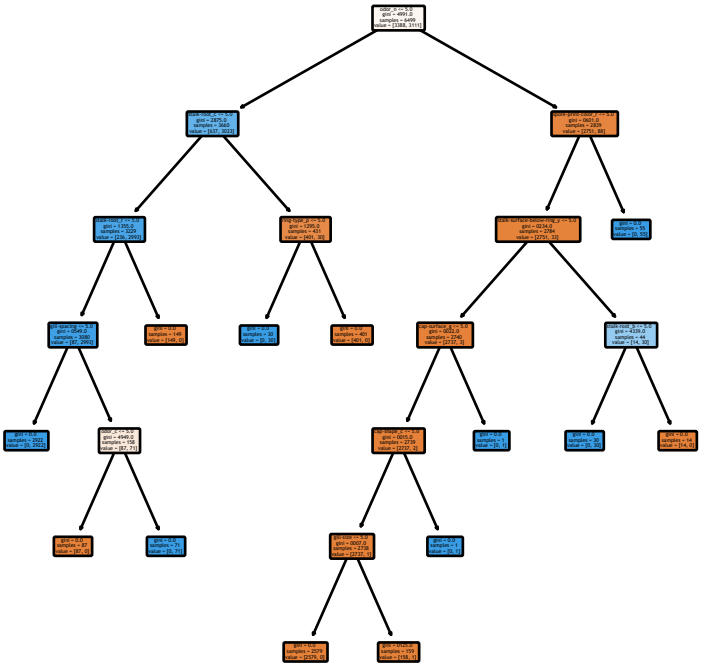
	precision	recall	f1-score	support
e	0.99	0.99	0.99	820
p	0.99	0.99	0.99	805
accuracy			0.99	1625
macro avg	0.99	0.99	0.99	1625
weighted avg	0.99	0.99	0.99	1625

Table 3-3: Decision tree accuracy measures with max depth 2

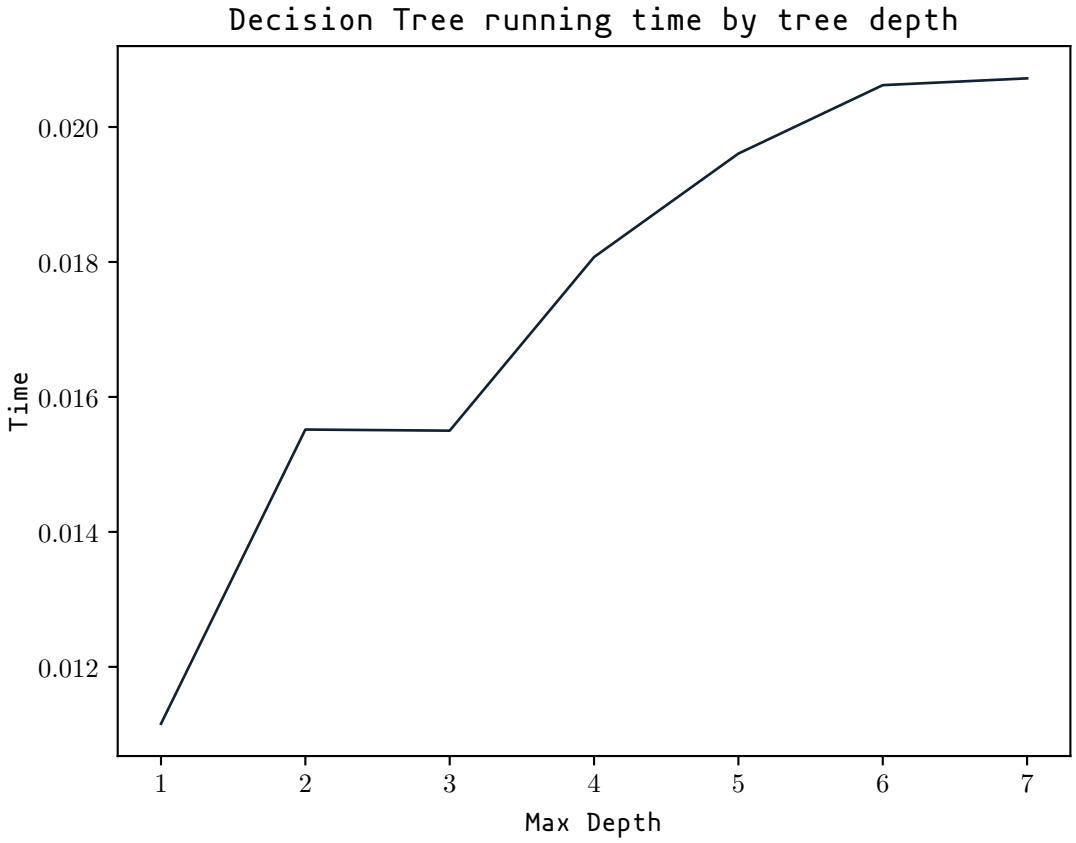
	precision	recall	f1-score	support
e	0.97	0.94	0.95	820
p	0.94	0.97	0.96	805
accuracy			0.96	1625
macro avg	0.96	0.96	0.96	1625
weighted avg	0.96	0.96	0.96	1625

۳-۲-۲ بررسی الگوریتم

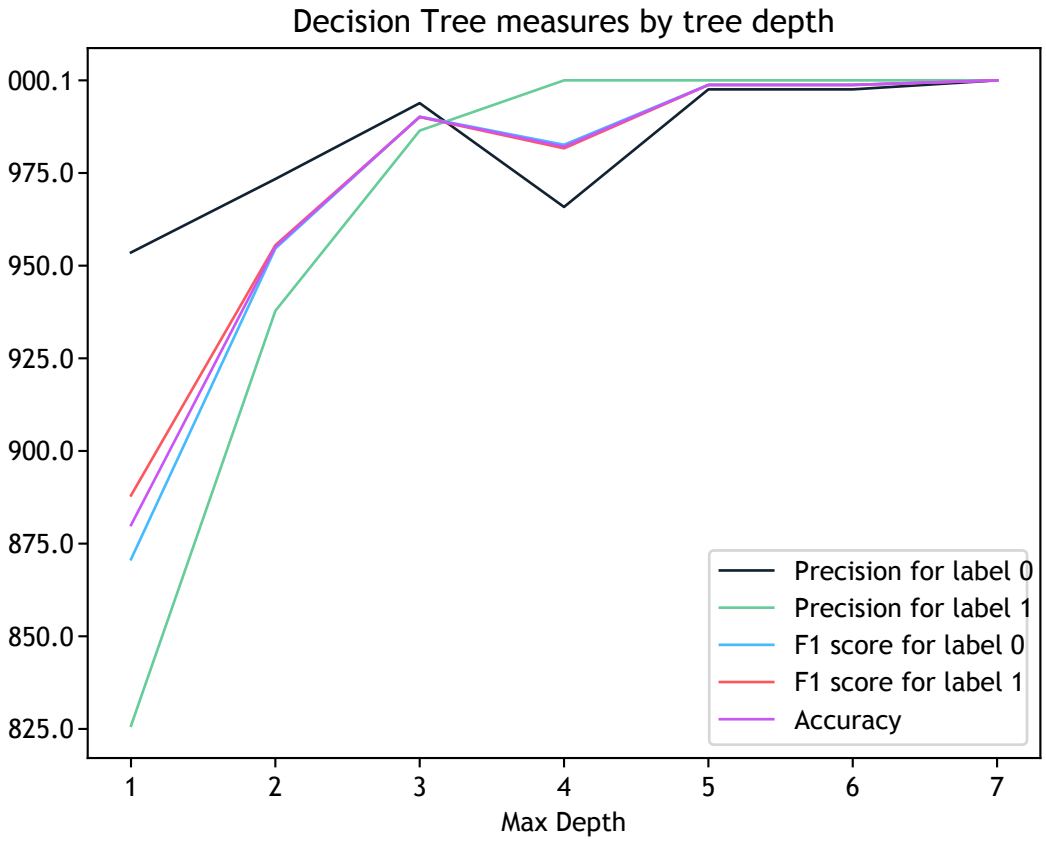
در این الگوریتم، با افزایش حداکثر عمق تا ۵ دقت افزایش می‌یابد و پس از آن تقریبا ثابت می‌ماند. زمان اجرا نیز با افزایش حداکثر عمق اکیدا صعودی است. (بخش ۳-۲-۲ بخش ۳-۲-۲)



شکل ۳-۱: Sample decision tree ouput



شکل ۳-۲: Decision Tree running time by max depth



شکل ۳-۳: Decision Tree Accuracy measures by max depth

۳-۳ Random Forest

پیاده سازی این الگوریتم نیز مانند پیاده‌سازی Tree Decision می‌باشد با این تفاوت که برای نمایش دادن آن فقط ۱۵ درخت اول را نمایش می‌دهیم. در شکل بخش ۳-۳ یک نمونه‌ی خروجی و در جدول‌های جدول ۳-۴ جدول ۳-۵ دقت و صحت مدل را مشاهده می‌کنید.

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
import os
import matplotlib
import matplotlib.pyplot as plt
from models.Model import Model

class RandomForest(Model):
   def __init__(self, X, y, name, feature_names, n_estimators=100,
          criterion="gini", min_samples_split=2,
          max_depth=2, min_samples_leaf=1, PLOT_nrows=3, PLOT_ncols=5):
      self.X =X
      self.y =y
      self.model =RandomForestClassifier(n_estimators=n_estimators,
                                          criterion=criterion,
            min_samples_split=min_samples_split,
            max_depth=max_depth, min_samples_leaf=min_samples_leaf,
                                                n_jobs=4)
      self.name =name
      self.feature_names =feature_names
      self.PLOT_ncols =PLOT_ncols
      self.PLOT_nrows =PLOT_nrows

   def train(self):
      self.model.fit(self.X, self.y)
```

```
def predict(self, data):
    return self.model.predict(data)

def save_output(self):
    plt.figure()
    fig, axes =plt.subplots(nrows=self.PLOT_nrows, ncols=self.PLOT_ncols,
          figsize=(7, 6), dpi=900)

    for i in range(15):
        dec_tree =plot_tree(decision_tree=self.model.estimators_[i],
              feature_names=self.feature_names,
               filled=True , precision=4, rounded=True,
               ax=axes[int(i /self.PLOT_ncols), i % self.PLOT_ncols])
        axes[int(i /self.PLOT_ncols), i %
                                              self.PLOT_ncols].set_title('Estimator:
                                              ' +str(i), fontsize =11)
    if not os.path.exists(os.path.dirname(os.path.join("./output",
        f"{self.name}.jpg"))):
        try: os.makedirs(os.path.dirname(os.path.join("./output",
            f"{self.name}.jpg")))
        except OSError as exc: # Guard against race condition
            if exc.errno !=errno.EEXIST:
                raise
    plt.savefig(os.path.join("./output", f"{self.name}.jpg"))
    if os.environ.get('LATEX_OUTPUT', '0') =='1':
        matplotlib.rcParams.update({
            "pgf.texsystem": "xelatex",
            'text.usetex': True,
            'pgf.rcfonts': False,
            "font.family": "mononoki Nerd Font Mono",
            "font.serif": [],
            # "font.cursive": ["mononoki Nerd Font", "mononoki Nerd Font
```

```
                                                Mono"],
        })
        plt.savefig(os.path.join("./output", f"{self.name}.pgf"))
    plt.show()
```

Table 3-4: Random Forest accuracy measures with max depth 5

	precision	recall	f1-score	support
e	0.97	1.00	0.99	820
p	1.00	0.97	0.99	805
accuracy			0.99	1625
macro avg	0.99	0.99	0.99	1625
weighted avg	0.99	0.99	0.99	1625

Table 3-5: Random Forest accuracy measures with max depth 2

	precision	recall	f1-score	support
e	0.87	1.00	0.93	820
p	1.00	0.85	0.92	805
accuracy			0.92	1625
macro avg	0.93	0.92	0.92	1625
weighted avg	0.93	0.92	0.92	1625

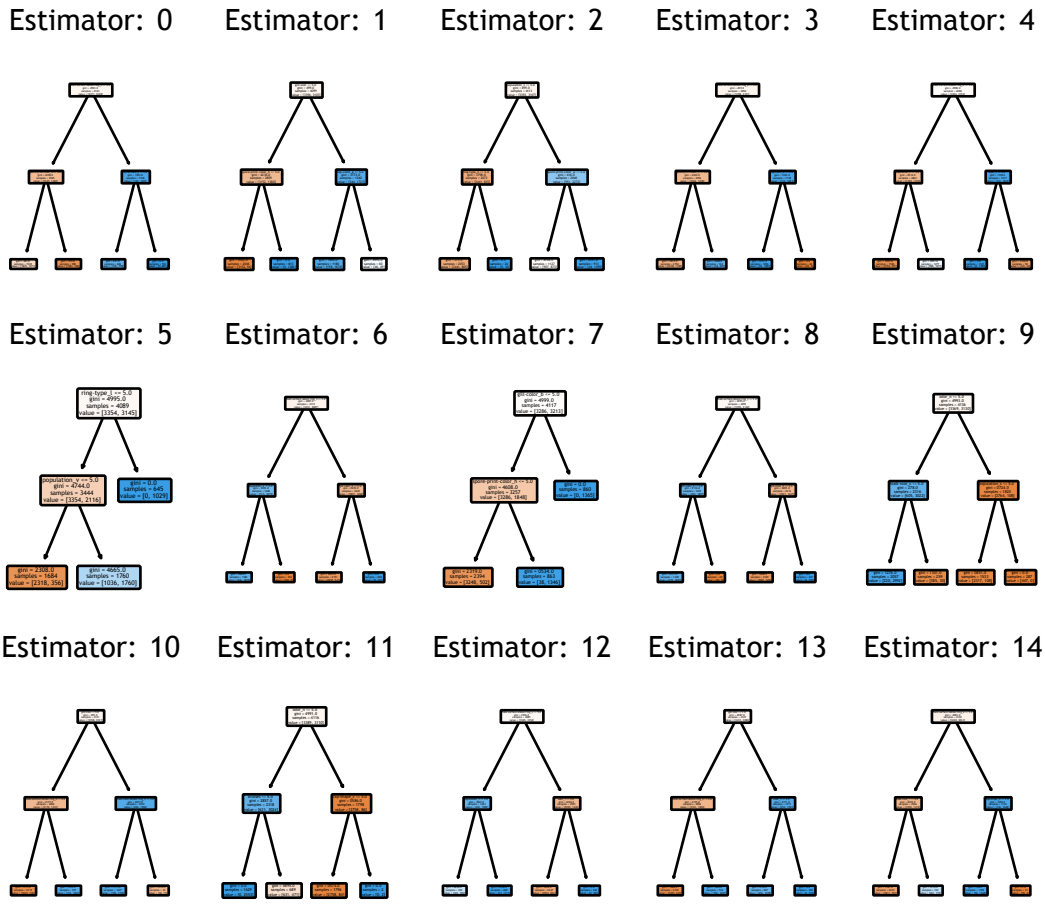
۳-۳-۱ بررسی الگوریتم

با افزایش حداکثر عمق دقت و صحت داعما افزایش می‌یابند اما زمان اجرا تغییر چندانی نمی‌کند. (بخش ۳-۳-۱ بخش ۳-۳-۱) با تغییر تعداد درخت‌ها تا ۱۰۰ دقت افزایش نمی‌یابد اما زمان اجرا ۴ تا ۵ برابر می‌شود. (بخش ۳-۳-۱ بخش ۳-۳-۱)

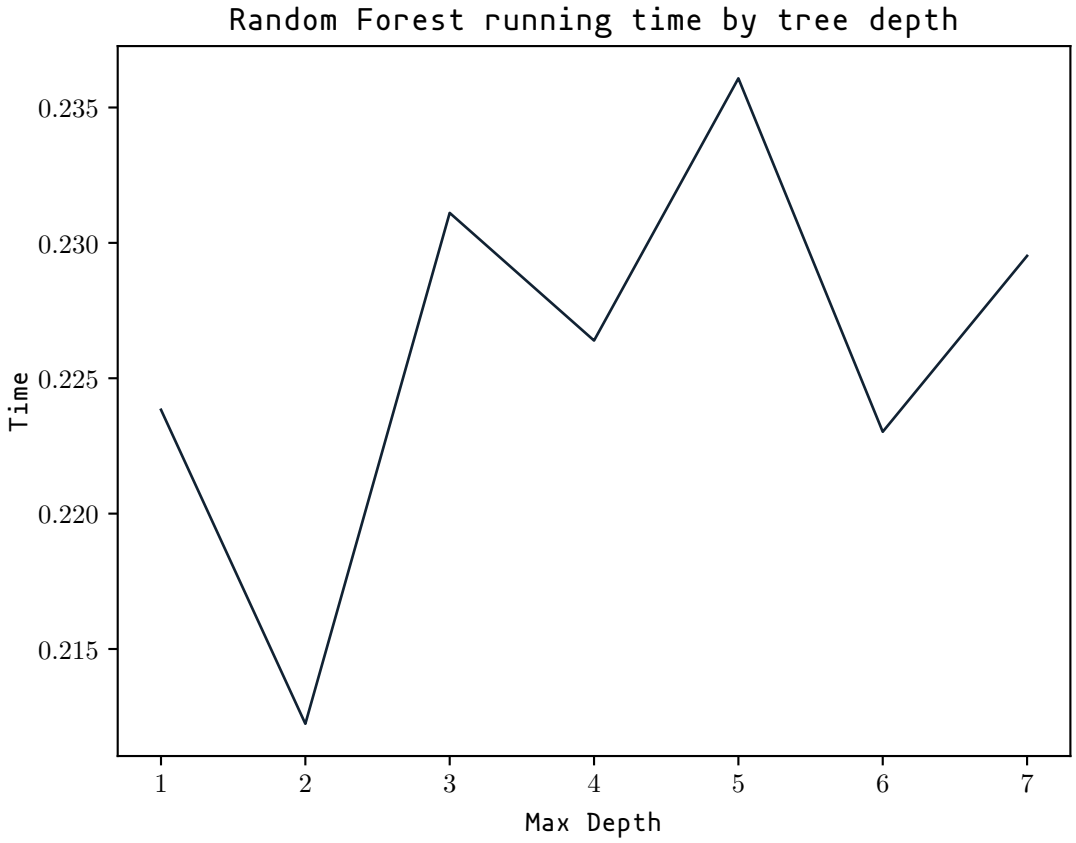
۳-۴ Perceptron and KNN

پیاده سازی این دو شبکه مانند شبکه‌های دیگر اما بدون نمایش انجام شده. این دو با کمترین زمان به دقت و صحت ٪۱۰۰ رسیدند.

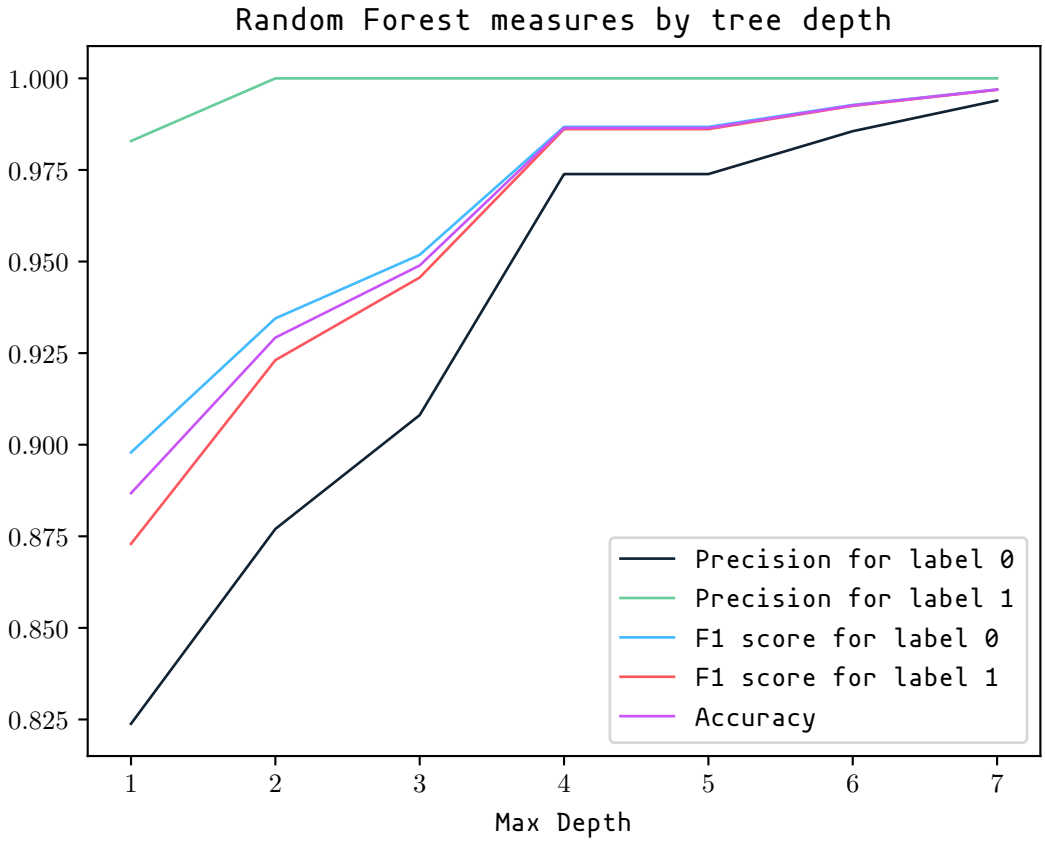
```
import pandas as pd
```



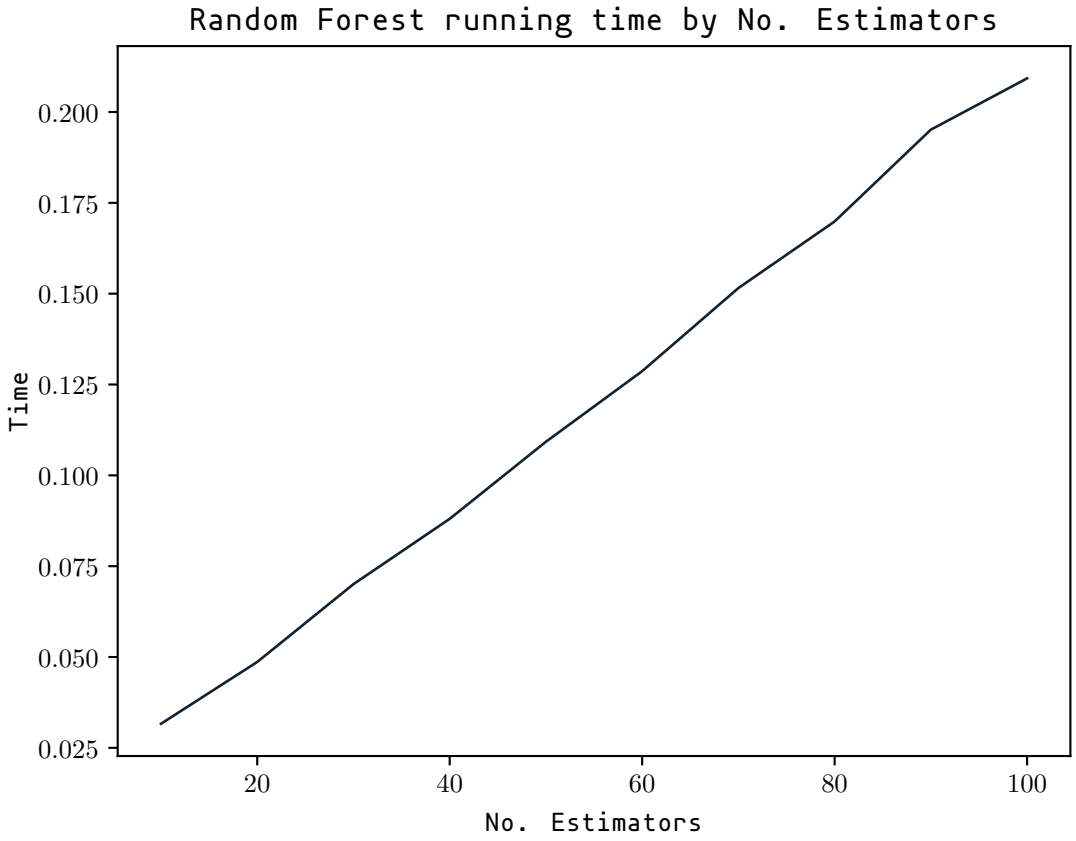
شکل ۳-۴: Sample decision tree ouput



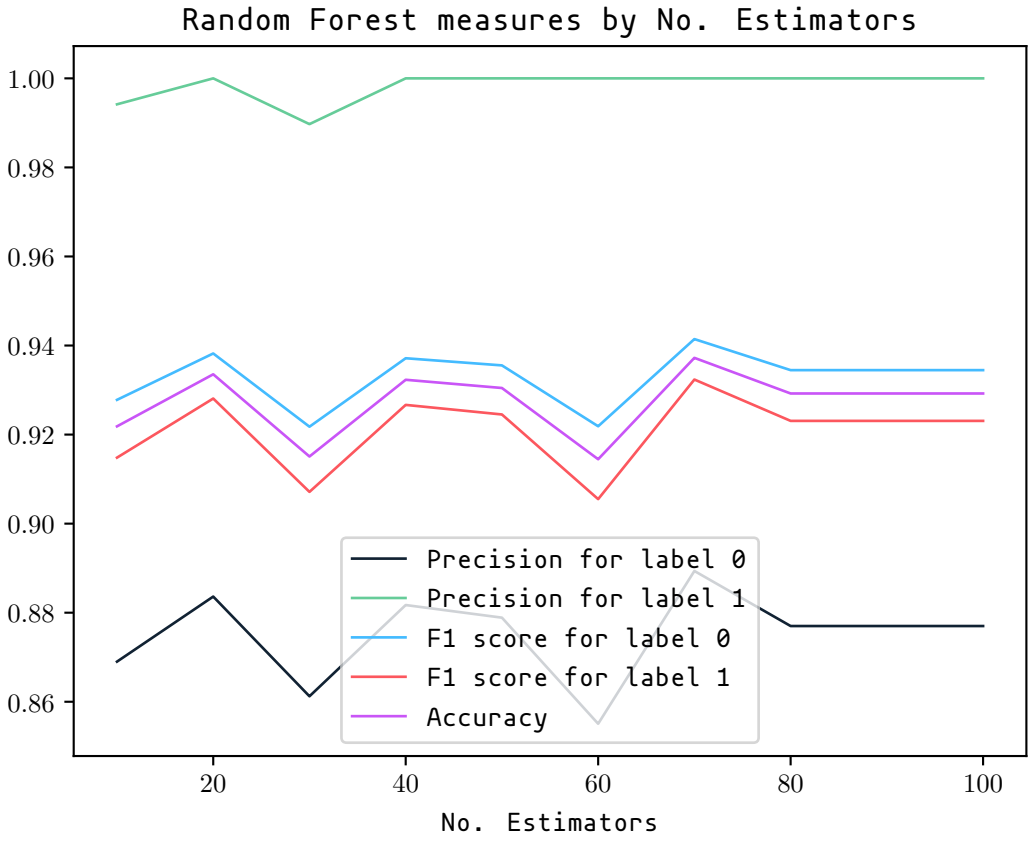
شکل ۳-۵: Random forest running time by max depth



شکل ۳-۶: Random forest accuracy by max depth



شکل ۳-۷: Random forest running time by no. estimators



شکل ۳-۸: Random forest accuracy by no. estimators

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import Perceptron as PerceptronClassifier
import os
import matplotlib.pyplot as plt
from models.Model import Model


class Perceptron(Model):
   def __init__(self, X, y, name, feature_names, max_iter=1000,
                                          tolerance=1e-3,
         eta0=1):
      self.X =X
      self.y =y
      self.model =PerceptronClassifier(max_iter=max_iter, tol=tolerance,
            eta0=eta0, n_jobs=4)
      self.name =name
      self.feature_names =feature_names

   def train(self):
      self.model.fit(self.X, self.y)

   def predict(self, data):
      return self.model.predict(data)
```

```
import pandas as pd
from sklearn.neighbors import KNeighbors
import os
import matplotlib.pyplot as plt
from models.Model import Model


class KNearestNeighbors(Model):
```

```
def __init__(self, X, y, name, feature_names, n_neighbors=6):
    self.X =X
    self.y =y
    self.model =KNeighbors(n_neighbors=n_neighbors, n_jobs=4)
    self.name =name
    self.feature_names =feature_names

def train(self):
    self.model.fit(self.X, self.y)

def predict(self, data):
    return self.model.predict(data)
```

۳-۵ خروجی

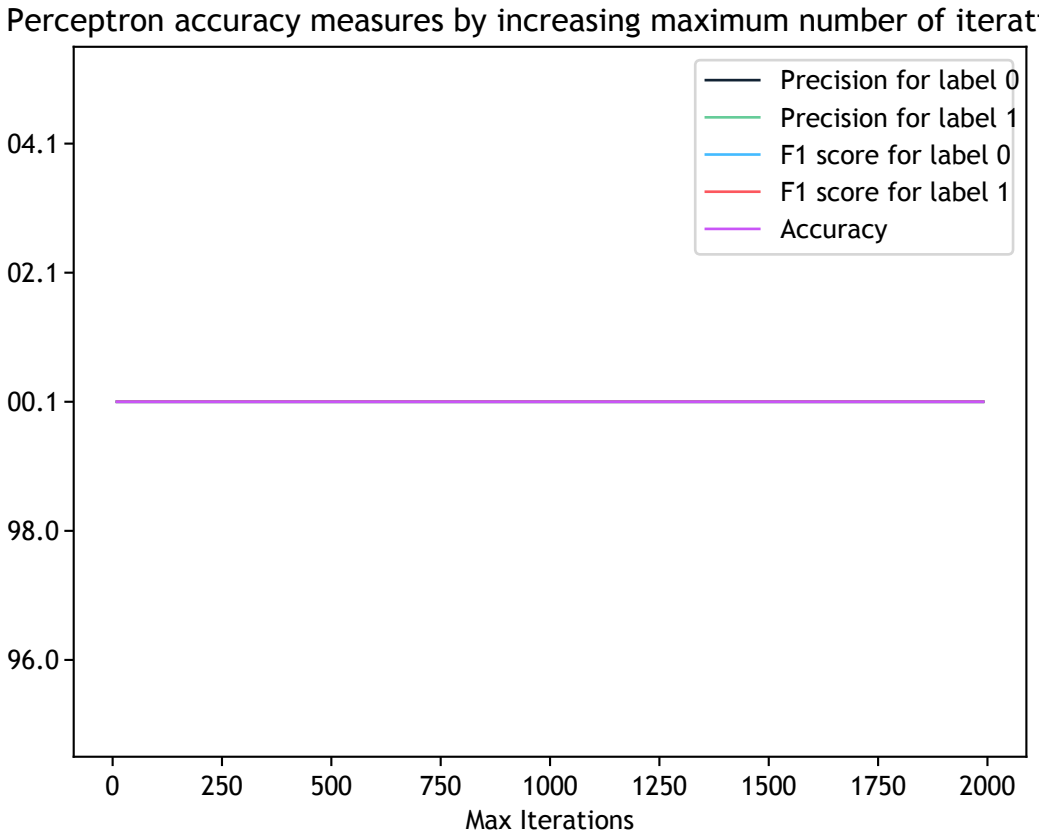
در جدول‌های جدول ۳-۶ و جدول ۳-۷ دقت Perceptron و NearestK- Neighbors نمایش داده شده.

Table 3-6: Perceptron Results

	precision	recall	f1-score	support
e	1.00	1.00	1.00	820
p	1.00	1.00	1.00	805
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625

Table 3-7: K-Nearest Neighbors Results

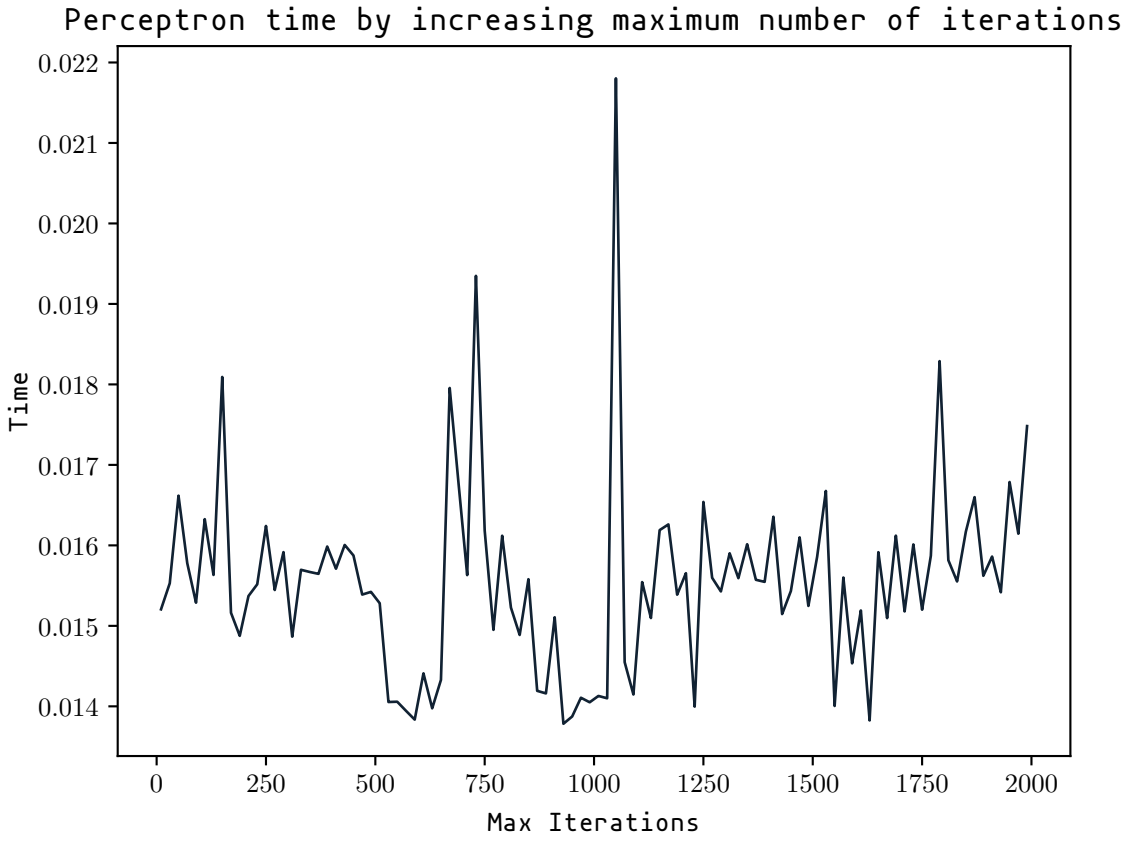
	precision	recall	f1-score	support
e	1.00	1.00	1.00	820
p	1.00	1.00	1.00	805
accuracy			1.00	1625
macro avg	1.00	1.00	1.00	1625
weighted avg	1.00	1.00	1.00	1625



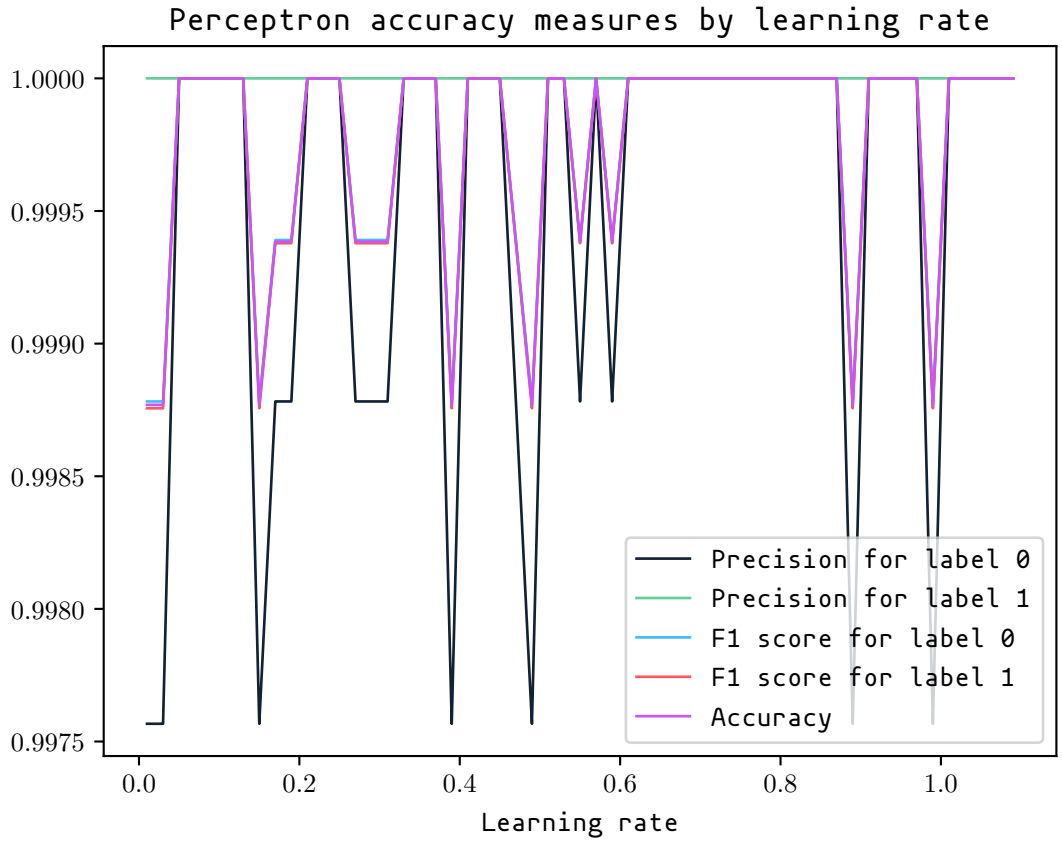
شکل ۳-۹: Perceptron Accuracy mearsures by maximum number of iterations

۳-۵-۱ بررسی الگوریتم

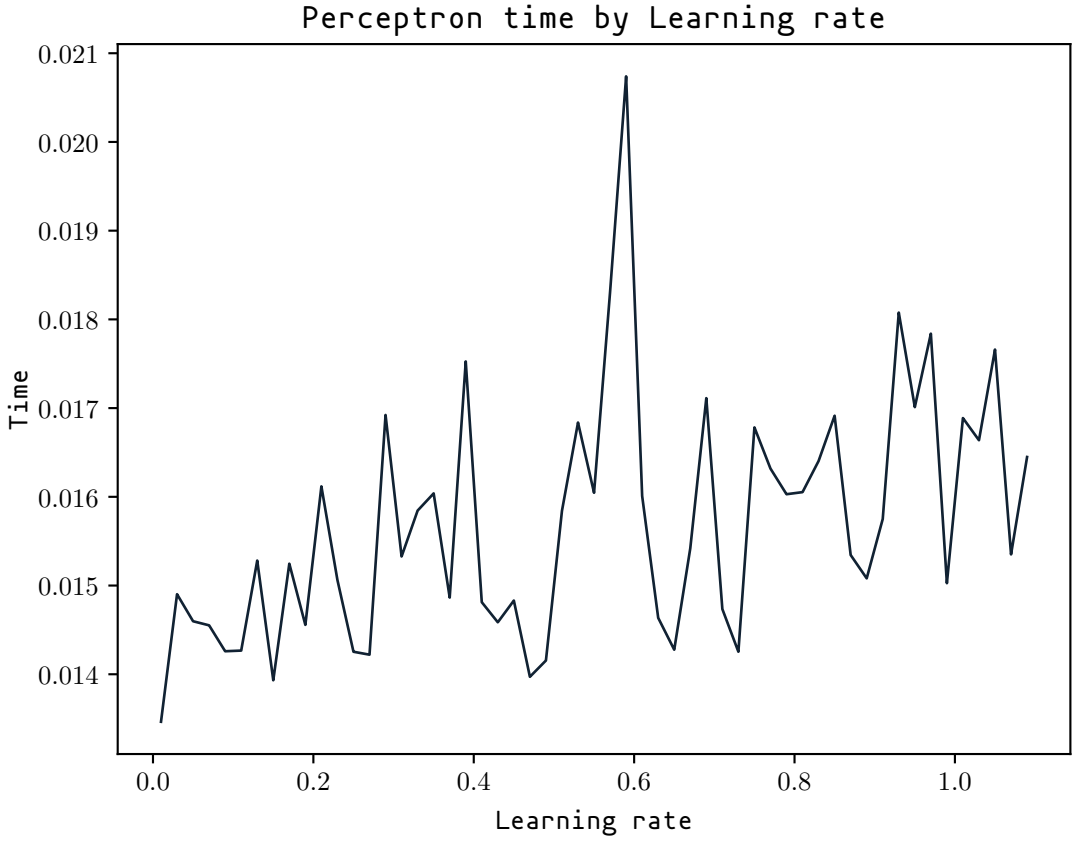
با تغییر `Tolerance` و تعداد `Iteraction` از دقت شبکه `Perceptron` کم نمی‌شود اما با تغییر `Rate Learning` دقت و صحت آن در بعضی موارد کم می‌شود. (بخش ۳-۵-۱ بخش ۳-۵-۱ بخش ۳-۵-۱) تغیرات زمان اجرای `Perceptron` در محدوده‌ی ۵ تا ۱۰ میلی‌ثانیه و قابل چشم‌پوشی می‌باشد. (بخش ۳-۵-۱ بخش ۳-۵-۱ بخش ۳-۵-۱)



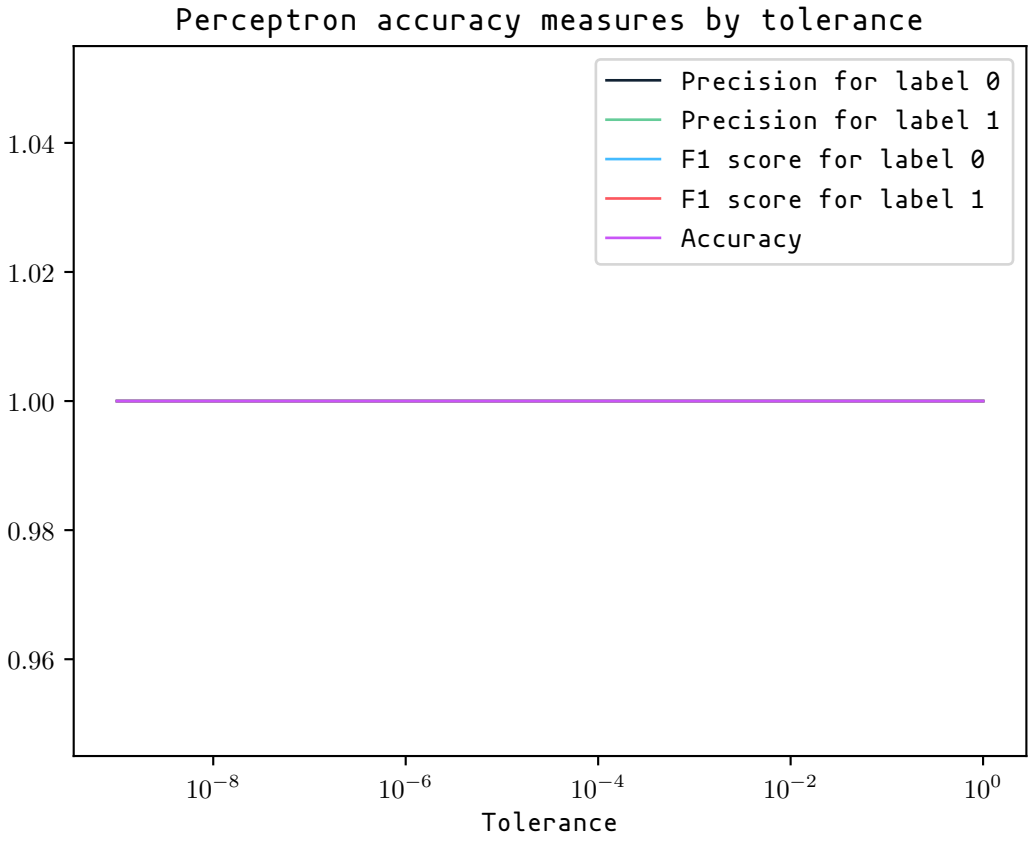
شکل ۳-۱۰: Perceptron time by maximum number of iterations



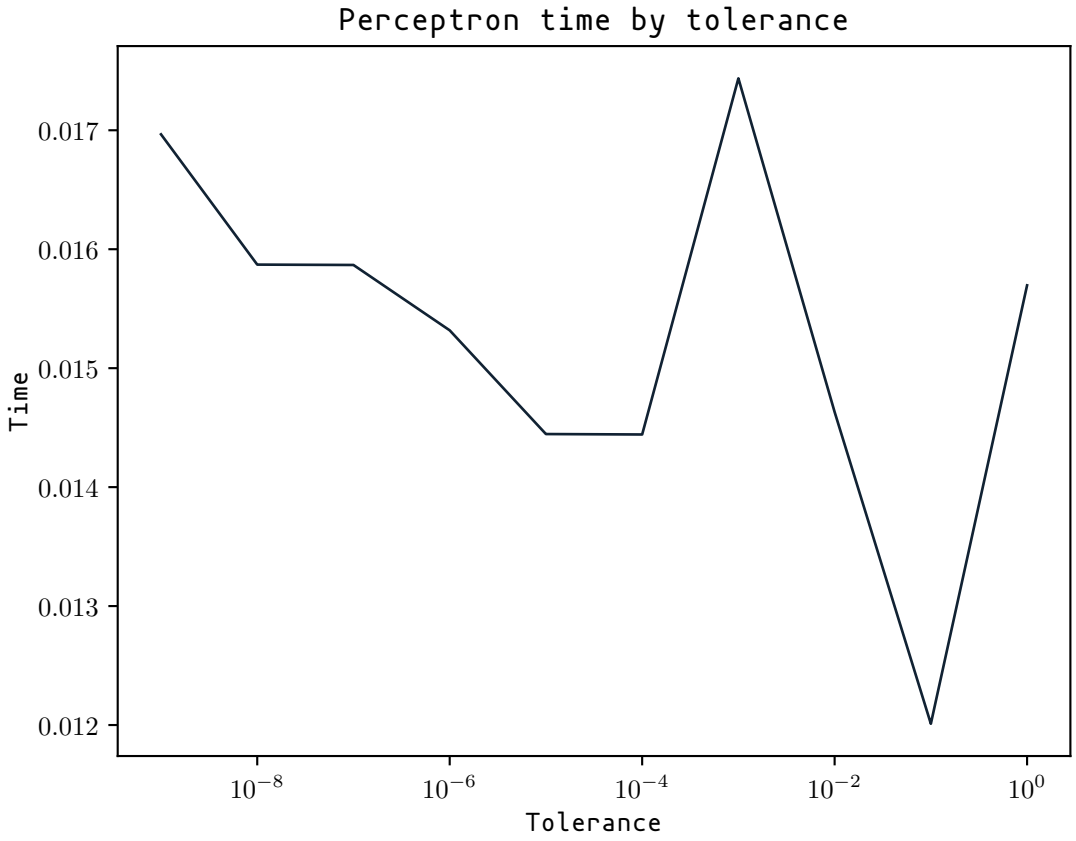
شکل ۳-۱۱: Perceptron accuracy measures by Learning rate



شکل ۳-۱۲: Perceptron time by Learning rate



شکل ۳-۱۳: Perceptron accuracy measures by tolerance



شکل ۳-۱۴: Perceptron time by tolerance

فصل چهارم

مقایسه‌ی الگوریتم‌ها

الگوریتم‌های Perceptron و KNN همواره بدون خطا بودند. بیشترین دقت بعد از آنها Tree Decision و در آخر Forest Random بود. (جدول ۴-۱) به لحاظ زمان Perceptron کمترین و Forest Random بیشترین زمان اجرا را داشتند. KNN و Tree Decision با اختلاف کم به ترتیب در جایگاهدوم و سوم بودند.

Table 4-1: Results

	Accuracy	Edible F1-Score	Piosonous F1-Score	Time
KNN	1.00	1.00	1.00	0.02 ± 0.005
Perceptron	1.00	1.00	1.00	0.015 ± 0.001
Decision Tree D2	0.95	0.96	0.96	0.019
Decision Tree D3	0.99	0.99	0.99	0.019
Decision Tree D6	1.00	1.00	1.00	0.036
Random Forest D2	0.93	0.92	0.93	0.237
Random Forest D3	0.97	0.96	0.96	0.237
Random Forest D6	1.00	1.00	1.00	0.252

منابع و مراجع

پیوست