گزارش پروژه سوم درس داده‌کاوی

پرهام کاظمی ۹۴۳۶۱۱۰۴۳۰۱۸

علیرضا پرچمی ۹۴۳۶۱۸۱۱۳۰۰۴

۲۹ دی ۱۳۹۷

مقدمه

هدف از این پروژه، استفاده از الگوریتم‌های دسته‌بندی [1] برای داده‌کاوی و کشف حقایق در مجموعه‌داده‌ی Mushroom می‌باشد.

در ابتدا، با استفاده از الگوریتم K-Folds Cross-Validation، مجموعه‌داده، ۱۰ بار به مجموعه‌داده‌های آموزش و آزمایش قطعه‌بندی شده است. سپس، با تشکیل درخت‌های ID3 و CART، قوانین مورد استفاده برای دسته‌بندی استخراج و دقت و صحت درخت‌ها به کمک معیارهای F-Measure و Presicion و Recall محاسبه شده‌اند.

در نهایت، الگوریتم دسته‌بندی K نزدیک‌ترین همسایه [2] بر روی مجموعه‌داده‌های آموزش و آزمایش - که با روش hold out تقسیم‌بندی شده‌اند - اجرا شده و دقت دسته‌بند با توجه به معیارهای F-Measure و Presicion و Recall محاسبه شده است.

۱ ابزارهای استفاده شده

در پیاده‌سازی این پروژه، از کتاب‌خانه‌های زیر در زبان پایتون استفاده شده است:

jupyter برای پیاده‌سازی و استفاده از الگوریتم‌های موجود در کتاب‌خانه‌ها در محیطی مناسب.

scikit-learn شامل پیاده‌سازی الگوریتم‌های تولید درخت‌های تصمیم‌گیری و KNN و همین‌طور محاسبه‌ی معیارهای اندازه‌گیری دقت دسته‌بندهای به‌دست‌آمده.

[1] classification

[2] K-Nearest Neigbours

pandas جهت خواندن داده‌ها از فایل و آماده‌سازی و پیش‌پردازش آن‌ها.

graphviz برای نمایش گراف‌ها و درخت‌های تولید شده و ذخیره‌ی خروجی در فایل pdf.

۲ مجموعه‌داده

توضیحات دیتاست

۳ درخت تصمیم

درخت‌های تصمیم، نوعی از دسته‌بندها می‌باشند که با تقسیم‌بندی‌های متوالی مجموعه‌داده در هر گره و تصمیم در یال‌های درخت، کلاس هر نمونه‌ی ورودی را تعیین می‌کنند. در این پروژه، از دو روش استفاده از آنتروپی (درخت‌های ID3) و معیار GINI (در درخت CART)، دو دسته‌بند به دست آمده و از نظر دقت با هم مقایسه شده‌اند.

۱.۳ پیش‌پردازش داده‌ها

در این مجموعه‌داده، ستون ۱۱اُم که بیانگر ویژگی stalk-root است، دارای مقادیر گم‌شده می‌باشد. برای رفع این مشکل، در نمونه‌هایی که دارای مقدار ناقص برای این ویژگی می‌باشند، مقدار «؟» با مُد داده‌های موجود در این ستون جایگزین شده‌اند. دلیل این کار، اسمی بودن ویژگی‌های موجود می‌باشد. بدین منظور، قطعه‌کد زیر با استفاده از کتاب‌خانه pandas اجرا شده است:

```
 import pandas as pd
m11 = data.mode()['stalk-root'][0]
data.loc[data['stalk-root'] == '?', 'stalk-root'] = m11
```

در این قطعه‌کد، ابتدا فراوان‌ترین مقدار ویژگی مورد نظر در متغیر m11 ذخیره شده و سپس مقادیر مشخص شده با «؟»، توسط مُد به‌دست‌آمده جایگزین شده‌اند.

۲.۳ تقسیم‌بندی مجموعه‌داده

برای تقسیم‌بندی مجموعه‌داده به دو دسته‌ی آموزش و آزمایش، از روش K-Folds Cross-Validation استفاده شده است. در این روش، مجموعه‌داده در ابتدا به K مجموعه‌ی کوچک‌تر تقسیم شده و برای ایجاد هر مدل، یکی از زیرمجموعه‌ها به عنوان داده‌ی آزمایش و سایر داده‌های موجود، برای آموزش مدل مورد استفاده قرار می‌گیرند. برای تولید درخت‌های تصمیم‌گیری، پارامتر K برابر ۱۰ فرض شده است. الگوریتم K-Folds در کتاب‌خانه‌ی scikit-learn به صورت زیر استفاده می‌شود:

```
 from sklearn.model_selection import KFold
sets = KFold(n_splits=10)
```

۳.۳ ایجاد درخت ID3

در کتاب‌خانه‌ی scikit-learn، درخت‌های تصمیم‌گیری با استفاده از کلاس DecisionTreeClassifier تولید می‌شوند. در ورودی تابع سازنده این کلاس، پارامتر criterion نوع درخت مورد نظر را تعیین می‌کند. برای ایجاد درخت ID3، مقدار 'entropy' به این پارامتر تخصیص داده می‌شود. سپس با فراخوانی متدهای fit و predict، به ترتیب داده‌های آموزش و آزمایش را در اختیار الگوریتم قرار می‌دهیم:

```
 dt = DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
```

سپس با استفاده از کتاب‌خانه‌ی scikit-learn، دقت کلاس‌های به‌دست‌آمده برای مجموعه‌داده‌ی آزمایش (y_pred) محاسبه می‌شود:

```
 from sklearn.metrics import precision_recall_fscore_support
presicion, recall, f_measure, _ =
precision_recall_fscore_support(y_test, y_pred,
average='micro')
```

درخت حاصل و همین‌طور دقت‌های به‌دست‌آمده در فایل خروجی (decision_trees.html) قابل مشاهده می‌باشند.

۴.۳ ایجاد درخت CART

مشابه الگوریتم تولید درخت ID3، می‌توان با کلاس DecisionTreeClassifier(criterion='gini') در کتاب‌خانه‌ی sklearn، درخت CART را ایجاد کرد.

۵.۳ مقایسه‌ی درخت‌ها

۶.۳ استخراج قوانین از درخت تصمیم

برای استخراج قوانین از روی درخت‌های به‌دست‌آمده، می‌توان بر روی هر درخت پویش اول عمق انجام داد و پس از رسیدن به برگ‌ها، قوانین به‌دست‌آمده را با هم ترکیب کرد. برای این کار، یک تابع بازگشتی تعریف کرده و با فراخوانی‌های تودرتو، قوانین را به صورت رشته‌های IF ... THEN ... چاپ می‌کنیم. قطعه‌کد تولید

قانون‌ها، در فایل decision_trees.html (آخرین بلوک) قابل مشاهده می‌باشد. قوانین به‌دست‌آمده نیز در فایل RULES.txt ذخیره شده‌اند.

۴ الگوریتم KNN

الگوریتم K نزدیک‌ترین همسایه (KNN) روشی برای پیش‌بینی label داده‌ها است که جزو کلاس‌بندهای تنبل به‌شمار می‌آید. این الگوریتم داده‌های آموزشی را دریافت کرده و آن‌ها را با توجه به تعداد ویژگی‌ها در یک فضای چند بعدی قرار می‌دهد. برای مثال چنان‌چه داده‌های ما دارای ۴ ویژگی باشد، فضای ما ۴ بعدی می‌شود. به همین دلیل است که داده‌های مورد استفاده در این روش به صورت عددی هستند.

پیش‌بینی در این الگوریتم به این صورت است که داده آزمایشی را گرفته و فاصله آن را با همه داده‌های آموزشی که از قبل دریافت کرده بود محاسبه می‌کند. با توجه به عدد K که در ابتدا برای این الگوریتم مشخص کرده‌ایم، K داده‌ای که کمترین فاصله را با داده آزمایشی ما دارد را انتخاب کرده و سپس label آن‌ها را بررسی می‌کند. آن label که بیشترین تعداد را دارد، برای label داده آزمایشی ما انتخاب می‌شود. به این مرحله Voting نیز گفته می‌شود.

برای پیاده‌سازی کلاس‌بند به روش KNN، از کتابخانه‌های sklearn و pandas کمک گرفتیم. این دو کتابخانه در پیاده‌سازی برخی توابع و هم‌چنین خواندن فایل‌های مربوط به دیتابیس و کار با آنها کمک شایانی می‌کند.

۱.۴ پیش‌پردازش داده‌ها

در ابتدا، قطعه کد زیر جهت پیش‌پردازش دیتاست و داده‌های از دست رفته (MissingValue) اجرا می‌شود. در تابع fill_missing، مقادیری که با "؟" پر شده‌اند را در هر ستون پیدا کرده و سپس با مقدار مد جایگزین می‌شود.

```
 def fill_missing(data): for col in data.columns:
mod = data[col].mode()[0]
data[col] = data[col].replace('?', mod)
```

سپس با توجه به اینکه روش KNN بروی داده‌های عددی پیاده‌سازی می‌شود، داده‌های اسمی را به داده‌های عددی تبدیل می‌کنیم. این کار به دلیل محاسبه فاصله توسط الگوریتم KNN انجام می‌شود. در تابع nominal_to_numeric، نوع داده‌ها را از object به category تغییر داده و سپس آنها را به عدد تبدیل می‌کنیم. مقادیر هر ستون از عدد صفر شروع شده و به ترتیب مقداردهی می‌شوند.

```
 def nominal_to_numeric(data):
obj_cols = data.select_dtypes(['object']).columns
```

```
data[obj_cols] = data[obj_cols].astype('category')
cat_cols = data.select_dtypes(['category']).columns
data[cat_cols] = data[cat_cols].apply(lambda x: x.cat.codes)
```

۲.۴ تقسیم‌بندی داده‌ها

سپس داده‌ها به روش holdout به دو دسته training_set و testing_set تبدیل می‌شوند. در این تابع، ابتدا به صورت تصادفی تعداد یک سوم داده‌ها در test_set انتخاب شده و سپس بقیه داده‌ها به عنوان داده‌های آموزشی در train_set ریخته شده و بازگردانده می‌شوند.

```
 def hold_out(data):
test_set_size = math.floor(len(data.index) / 3)
test_set_indexes = sample(range(0,len(data.index)),
test_set_size)
test_set = pd.DataFrame(data, index=test_set_indexes)
train_set = data.drop(test_set_indexes)
return train_set, test_set
```

۳.۴ پیاده‌سازی و اجرای الگوریتم

سپس در تابع main، الگوریتم KNN پیاده‌سازی شده است. ابتدا دو مجموعه تولید کرده و یکی از آنها شامل ستون اول است که نتیجه‌های اصلی را شامل می‌شود (training_set_result) و دیگری شامل داده‌ها است (training_set_data). پس از آن با استفاده از الگوریتم KNN پیاده‌سازی شده در کتابخانه sklearn، مدل موردنظر خود را با n=3 می‌سازیم.

۴.۴ بررسی دقت و صحت

سپس برای بدست آوردن مقدار Precision، Recall و F_score ابتدا مجموعه testing_set را به دو مجموعه حاوی نتایج نهایی و داده‌ها تقسیم می‌کنیم و سپس داده‌های تست را به مدل ساخته شده می‌دهیم. نتایج ذخیره شده در متغیر predicted را با نتایج اصلی که در متغیر testing_set_result ریخته بودیم را به‌کار می‌گیریم تا مقدار precision و recall و سپس f_score را بدست آوریم.

```
 def hold_out(data):
test_set_size = math.floor(len(data.index) / 3)
test_set_indexes = sample(range(0,len(data.index)),
test_set_size)
```

```
test_set = pd.DataFrame(data, index=test_set_indexes)
train_set = data.drop(test_set_indexes)
return train_set, test_set
```