گزارش پروژه فاز اول

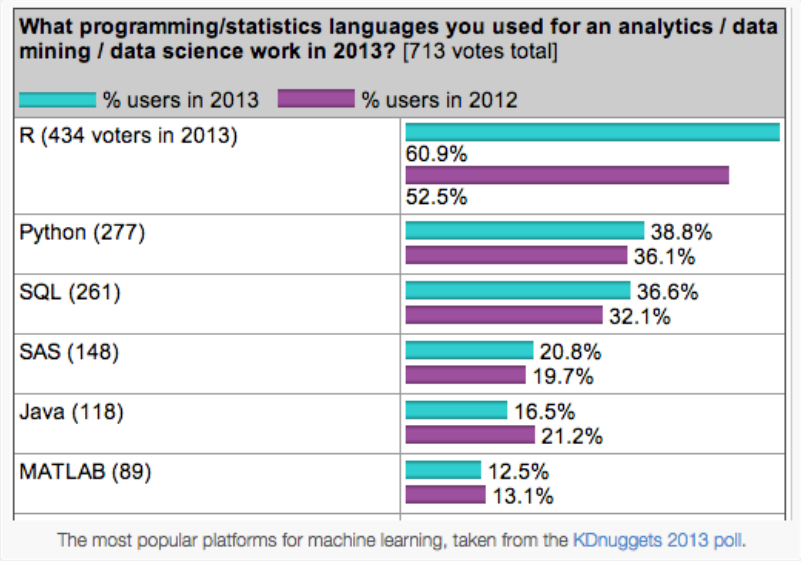
یادگیری ماشین

دکتر سلیمانی

تینا خواجه - ۹۳۲۱۰۷۶۱

انتخاب زبان برنامه نویسی و کتابخانه مورد استفاده :

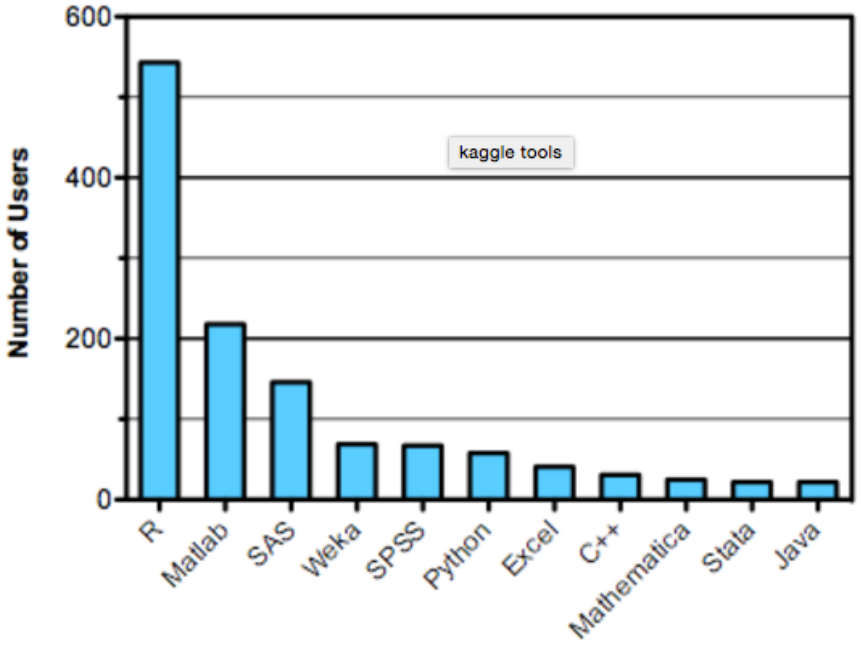
قبل از آنکه به انتخاب زبان برنامه نویسی بپردازیم به شناخت زبان های مختلفی که در این حیطه مورد استفاده است می پردازیم تا از میان آنها با توجه به نیاز خود یکی را برگزینیم. در یک نظر سنجی که با عنوان زبان برنامه نویسی مورد استفاده برای کار با داده ها که در سال ۲۰۱۳ برگزار گردید نتایج زیر بدست آمد.



The most popular platforms for machine learning, taken from the KDnuggets 2013 poll.

همانطور که در شکل مشخص است زبان های R,Python,Java و Matlab از جمله زبان های برنامه نویسی پرطرف دار برای کار با داده ها هستند.

علاوه بر ای نظر سنجی Kaggle نتایجی را در سال ۲۰۱۱ با عنوان Kaggler's Favorite tools منتظر کرد که در آن مشخص می کند در مسابقات یادگیری ماشین برگزار شده شرکت کنندگان برا یشرکت در مسابقات از چه زبان هایی استفاده کرده و کدامیک پر طرف دار تر هستند. این نتایج را در زیر مشاهده می نمایید:



همانطور که در این نتایج نیز مشخص است زبان برنامه نویسی R پر طرف دار ترین زبان می باشد. اما قبل از انتخاب زبان مورد نظر باید دقت داشته باشیم که برای پروژه پیش رو چه ابزار هایی لازم است و کدامیک از این زبان ها ما را به هدفمان نزدیک تر می کنند.

در این پروژه هدف ما یادگیری الگوریتم ها و جزییات دقیق پیاده سازی نیست و بیشتر به دنبال شناسایی رفتار هر یک از این روش ها بر روی داده ها هستیم.

ویژگی های هر یک از زبان هایی که کاندیدای انتخاب به عنوان استفاده در این پروژه بوده اند را در زیر می آوریم تا با بررسی هر یک در نهایت انتخاب درستی داشته باشیم:

MATLAB

یکی از خوبی های این زبان برنامه نویسی این است برای محاسبات برداری و ماتریسی طراحی شده و مشکلی با داده های برداری ندارد.

برای کاربرد هایی همچون پردازش سیگنال بسیار ارزشمند است و دارای کتابخانه های وسیعی می باشد.

متلب ساده ترین و دقیق ترین زبان برای کار با ماتریس است و برای هر مسئله ای که به صورت بردار ویژگی عددی ظاهر شود بسیار مناسب کار می کند.

خوبی دیگری که این زبان دارد این است که مانند زبانی همچون c درگیر کار های low level در هنگام کد زدن نمی شویم.

این زبان برای مواردی که قصد یادگیری الگوریتم و پیاده سازی آن با جزییات توسط خود فرد می باشد مناسب است اما ابزار قرت مندی به عنوان مرجعی از توابع و روش های آماده و مورد استفاده کاربران محسوب نمی شود.

R

مشابه با متلب این زبان نیز محاسبات برداری و ماتریسی را با سرعت بالایی انجام می دهد. R زبانی است که با هدف آنالیز های آماری و تمرکز بر روی این اصل بوجود آمد است. این زبان برای کار با داده ها و استخراج ویژگی های آماری موجود در آنها بسیار مناسب می باشد و تمام ابزار لازم برای این کار از جمله توابع، نمودار ها و... را تحت عنوان پکیج های مختلفی در اختیار کاربران خود قرار می دهد. هدف اصلی زبان کار آنالیز داده ها می باشد و بعد از آن این زبان در جهتی گسترش یافت که برای کاربرد های یادگیری ماشین نیز مورد استفاده قرار بگیرد. اما باید دقت داشته باشیم هرچند که این زبان دارای پکیج های مختلفی با هدف فراهم آوردن الگوریتم ها و ابزار های مورد استفاده در زمینه یادگیری ماشین است، اما این پکیج ها نقاط ضعفی همچون نداشتن داکیومنت ها با توضیحات دقیق، نداشتن انسجام بین پکیج ها و... را دارند که منجر می شود گاها کار بر در هنگام کار کردن با آنها دچار سر در گمی شود(همانطور که در ادامه و طبق تجربه اینجانب توضیح داده می شود) کار کردن با پکیج های مختلف R و رفتن از یک پکیج به پکیج دیگر معمولا کاربر را با interface ای متفاوت از پکیج قبلی مواجه می کند و با عث سردر گمی می شود.

برای کارهایی همچون feature extraction مناسب نمی باشد و بهتر است در کنار R از ابزاری دیگر همچون vowpal wabbit برای این هدف استفاده کنیم.

Python

به تازگی پایتون جایگاه خوبی در بین برندگان رقابت های یادگیری ماشین پیدا کرده است که با استفاده از کتابخانه scikit-learn در رقابت ها شرکت کرده بودند. کتابخانه scikit-learn یکی از بهترین کتابخانه های موجود برای روش ها و الگوریتم های یادگیری ماشین محسوب می شود که راهنمای استفاده بسیار قوی همراه با توضیحات و مثال هایی را برای کاربران خود فراهم می آورد.

یکی از دغدغه هایی که هنگام انتخاب زبانی مثل python وجود دارد این است که سرعت اجرا و محاسبات بسیار کند باشد. این کند بودن سرعت برای اجرای الگوریتم هایی همچون یادگیری ماشین که با داده های برداری کار می کنند در زبان پایتون کاملا مشاهده می شود. این کندی در سرعت اجرا را می توان از طریق تبدیل به زبان C حل کرد.

یکی دیگر از مزیت های کتابخانه scikit-learn در کنار جامع بود، کامل بود، توجه به جزییات مربوط به یادگیر ی ماشین و همچنین داشتن داکیومنتی کامل و دقیق به همراه مثال های کافی و نیز محبوبیت در بین متخصصان مختلف، توجه به مشکل این زبان برنامه نویسی برای کند بود اجرا است. این مشکل با تبدیل C حل شده و کاربران بدون داشتن دغدغه سرعت اجرای بسیار متفاوت با دیگر زبان ها می توانند از این کتابخانه استفاده نمایند.

در کنار توضیحات بالا که در باره مزیت های کتابخانه scikit-learn داده شد، این مزیت بزرگ را نیز باید در نظر بگیریم که زبان پایتون زبانی سطح بالا است و کاربران خود را از کار کردن با خیلی جزییات راحت می کند و تجربه خوبی از برنامه نویسی را در اختیار کاربران خود قرار می دهد.

در کنار کتابخانه scikit-learn کتابخانهnumpy کمک می کند که زبان پایتون تبدیل به زبان های مناسب برای یادگیری ماشین تبدیل شود.

Java

یکی از خوبی های این زبان برنامه نویسی سرعت خوب اجرای آن است. همچنین برای این زبان کتابخانه هایی همچون Mahout , Weka

جمع بندی و نتیجه گیری :

در بالا زبان های مختلف را از جنبه های مختلف کتابخانه های موجود ، سرعت اجرا ، راحتی در برنامه نویسی و... بررسی کردیم. با وجود توضیحات بالا انتخابی باقی نمی ماند و واضح است که به سراغ کدام زبان باید برویم :پایتون!

با وجود مشخص بودن این موضوع به دلیل مشکل در نصب کتابخانه بر روی سیستم اینجانب در مرحله بعد به سراغ زبان برنامه نویسی R رفته و سعی در انجام پروژه به زبان R داشتم که بعد از یک روز به نامناسب بودن پکیج های یادگیری ماشین و نکات ذکر شده در جهت سردر گمی کاربر پی برده و دوباره به سراغ زبان برنامه نویسی پایتون آمده و هر طور بود کتابخانه را نصب نمودم. گزارشات پیش رو حاصل پیاده سازی به زبان پایتون می باشد.

پیاده سازی و نتایج :

دوکلاسه

feature selection :

در ابتدا ویژگی های مناسب را انتخاب می کنیم :

روش های مختلفی برای این کار وجود دارند که از این جمله می توان به روش های :Removing features with low variance, Univariate feature selection,Recursive feature eliminationFeature selection using SelectFromModel اشاره کرد. روش هایی ک هر ویژگی را تنها با در نظر گرفتن خود داده انتخاب می کنند زیاد مورد علاقه نیستند و دوست داریم ویژگی ها را همه را با همدر نظر گرفته و تعیین کنیم که کدام ها خوب هستند و کدامها بد. برای انتخاب ویژگی های مناسب می توانیم از ماژول sklearn.feature_selection در کتابخانه scikit

learn استفاده نماییم. یکی از بهترین انواع انتخاب ویژگی Lassoاست که جوابی sparse دارد و در نهایت آن ویژگی هایی که ضرایب غیر صفر برای آنها بدست آمده را انتخاب می نماید.

پارامتر C موجود در این تابع مشخص کننده میزان sparsity می باشد. فرم به صورت زیر قابل مشاهده است :

class `sklearn.svm.`**LinearSVC**(*penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*) [source]

پارامتر های استفاده شده در این حالت عبارتند از :

C=0.2, penalty="l1"

در این حالت تعداد ویژگی ها از ۴۱۴ به ۴۳ ویژگی کاهش یافت.

۱) دسته بندی دو کلاسه : svm

ابتدا برای در نظر گرفتن اهمیت برابر برای تمام داده ها، با توجه به اینکه داده ها را نمی شناسیم که آیا یکی بر دیگری اهمیت دارد یا نه، داده های را scale می کنیم :

برای این کار از پکیج sklearn.preprocessing استفاده می کنیم و با استفاده از تابع MaxAbsScaler() داده ها را به بازه [-1,1] منتقل می نماییم. بدین ترتیب و از طریق یکسان کردن واریانس و رنج داده ها اثر همه در بدست آوردن مدل نهایی را یکسان می کنیم.

کد مربوط به این قسمت در برنامه نوشته شده عبارت است از:

```
import numpy as np
max_abs_scaler = preprocessing.MaxAbsScaler()
xtrainScaled = max_abs_scaler.fit_transform(xtrain)
```

همچنین علاوه بر داده های train داده های validation و همچنین test را نیز به همین صورت scale می نماییم.

برای اجرای الگوریتم svm بر روی داده ها به سراغ پکیج svm می ریم. برای استفاده از این الگوریتم از انواع مختلفی از تkظیمات استفاده می نماییم که در این حالت پارامتر های زیر را برای آن تغییر می دهیم :

kernel, C, gamma

در این حالت باید مدل های مختلف را بر روی داده train آموزش ببینیم و بهترین مدل را بر اساس کوچک ترین خطا بر روی داده های validation محاسبه نماییم. در صورتی که داده های train و validation جدا نبودند و می توانستیم از cross validation استفاده نماییم می توانستیم از تابع sklearn.grid_search.GridSearchCV برای پیدا کردن بهترین مقادیر پارامتر ها با یک دستور استفاده نماییم. اما به دلیل اینکه اینجا داده های train و validation جدا هستند باید ابتدا مدل را بر روی داده های train آموزش دهیم و بعد بر روی validation امتحان کنیم و مدل با پارامتر هایی را به عنوان بهترین مدل انتخاب کنیم که کمترین خدا بر اساس پعیار مورد بررسی را دارد.

کرنل های مختلفی که می توانیم استفاده کنیم اما به دلیل اینکه هیچ اطلاعی از داده ها نداریم که کدام کرنل برای آن مناسب تر است و همچنین اینکه اجرای تمام کرنل ها با توجه به زمان اجرا و اجرای برنامه بر روی لپ تاپ غیر ممکن می باشد به همین دلیل چون کرنل rbf حالت کلی تری دارد و برای وقتی که اطلاعی از داده ها نداریم معمولا جواب مناسبی می دهد به سراغ این کرنل رفته و به دنبال پیدا کردن بهترین نوع پارامتر ها برای آن می گردیم.

البته این قابلیت را با استفاده از این کتابخانه خواهیم داشت که کرنل توسط کاربر مشخص شود ولی از آن استفاده نمی کنیم.

پارامتر C نیز بیان می کند که چقدر missclassification را جریمه می کنید و این میزان بر روی margin اثر گذار است هرچه C بزرگ تر باشد margin کوچک تر می شود. برای مشخص کردن بهترین C مقادیر مختلف 10^3تا 10^-6 را چک می کنیم.و Cای که بهترین معیار را بر روی داده ها داشته باشد انتخاب می کنیم و سپس به سراغ پیدا کردن بهترین gamma با توجه به این c خاص می گردیم. برای پیدا کردن بهترین c نیز پارامتر گاما را برای تمام حالات ۰/۱ قرار می دهیم. نتایج دو معیار در جدول زیر آمده است :

C	10^-6	10^-5	10^-4	10^-3	10^-2	0.1	1	10	100	1000
Accuracy	0.6189999999999999	0.6189999999999999	0.6189999999999999	0.6189999999999999	0.7072000000000005	0.8042000000000003	0.8299999999999996	0.8461999999999995	0.8329999999999996	0.8145999999999999
F-Measure	0.7646695491043854	0.7646695491043854	0.7646695491043854	0.7646695491043854	0.8078235757416644	0.8534650501421943	0.8695518723143032	0.8463	0.8673550436854647	0.8522709163346145

همانطور که در جدول بالا نیز مشخص می باشد بهترین c عبارت است از ۱۰. حال با در نظر گرفتن این مقدار به عنوان c به دنبال مقدارgamma می رویم که بهترین دقت را به ما بدهد نتایج را در جدول زیر می توان مشاهده بنمایید:

gamma	10^-6	10^-5	10^-4	10^-3	10^-2	0.1	1	10
Accuracy	0.618999999999999999	0.618999999999999999	0.753199999999999998	0.792399999999999999	0.807200000000000003	0.846199999999999995	0.849999999999999998	0.618999999999999999
F-Measure	0.764669549104385545	0.764669549104385545	0.828611111111111108	0.841961023142509090	0.851326341764343080	0.879674542325144860	0.886122077133313040	0.764669549104385545

همانطور که در بالا قابل مشاهده است بهترین مقدار برای gamma عدد یک است که منجر به بدست آوردن بهترین معیار شده است. در نهایت بهترین دقتی که برای داده های validation به دست آمده است عبارت است از دو خانه رنگی شده.

همچنین cache size را که میزان RAM اختصاص داده شده به برنامه می باشد را 500 قرار می دهیم به این امید که سرعت اجرا بالا رود. که البته همچنان سرعت پایین است و برای یک بار اجرا و بدست آوردن نتایج بالا حدودا به یک ساعت زمان احتیاج است.

برای محاسبه معیار دقت از تابع sklearn.metrics.accuracy_score¶ استفاده می کنیم که prototype تابع به صورت زیر می باشد :

```
sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)
```

همچنین برای حالت دو کلاسه نیز از معیار f1 استفاده می کنیم که معادل تابع زیر می باشد :

```
sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)
```

حال زمان آن رسیده که با توجه به بهترین مدلی که بدست آورده ایم برچسب های داده های test را پیش بینی نماییم و نتایج را در فایل بنویسیم. پیش بینی مقادیر نهایی در فایل bin_test_svm برای این قسمت قرار داده شده

و ارسال گردیده است همانطور که در فایل کد این قسمت با نام bin_svm.py نیز آمده است برای خاوندن اطلاعات از دستور زیر استفاده نمایید.

```
import pickle
f = open('bin_test_svm', 'r')
yTestPredic = pickle.load(f)
```

KNN:

برای پیاده سازی این قسمت به سراغ تابع

class `sklearn.neighbors.`**`KNeighborsRegressor`**(*n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs*) [source]

می رویم که یک پیاده سازی از KNN است و در آن پیدا کردن تعدا همسایه ها مشکل به نظر می رسد. در ابتدا از روش استفاده شده در مرحله قبل برای feature extraction استفاده می کنیم.

پارامتر هایی که برای این مسئله تنظیم می کنیم عبارتند از :

n_neighbor = sqrt(number of features)

weights = distance

distance در واقع میزان اثر گذاری هر یک ار نمونه ها را مشخص می نماید که متناسب است با عکس فاصله.

همانطور که گفته شده پیدا کردن بهترین K کار ساده ای نیست و به دلیل اینکه هیچ اطلاعاتی در باره داده ها نداریم نمی توانیم یک بازه مشخص کرده و د رآن بازه به دنبال K بگردیم. یک قانون سر انگشتی وجود دارد که جذر تعداد نمونه ها در نظر بگیریم. مشکل این classifire هم همین است که انتخاب K مهم است و به شدت می تواند نتیجه را تحت تاثیر قرار دهد و قانون خاصی برای آن نداریم.

به منظور انتخاب درست مقدار K مقدار دو معیار خواسته شده را برای k های مختلف محاسبه نمودیم :

K	5	10	25	50	100	250
Accuracy	0.8397999999999999	0.8494000000000004	0.8489999999999998	0.8474000000000004	0.8376000000000001	0.8217999999999997
F-Measure	0.8736792304052882	0.8820673453406202	0.8840067598709479	0.8839190628328086	0.8776371308016877	0.8682537335501975

K	500	1000	2500	5000	10000
Accuracy	0.804000000 00000005	0.7792	0.728400000 00000005	0.660399999 99999999	0.618999999 99999999
F-Measure	0.858135495 07817033	0.844507042 25352113	0.817374932 75954798	0.783639143 73088672	0.764669549 10438545

بنابراین همانطور که از اطلاعات درج شده در جدول مشخص است بهترین مقدارk که البته دارای تفاوت اندکی در دو معیار محاسبه شده و مشخص شده در بالا است، می تواند مقداری برابر با ۲۰ در نظر گرفته شود.

پس مقادیر y را با این k تخمین می زنیم و در فایل bin_test_knn ثبت می نماییم. همانطور که در قسمت قبل نیز اشاره شد برای بدست آوردن اطلاعات درون فایل از دستورات ذکر شده در قسمت قبل استفاده نمایید.

Naive bayes :

این روش احتیاج به scaling ندارد وو این کار بر روی نتیجه اثری نخواهد داشت بنابراین این کار را بر روی داده های این قسمت انجام نمی دهیم. همچنین این روش با توجه به ماهیت خود که ویژگی ها را مستقل از هم دیگر در نظر می گیرد و در نتیجه دارای سرعت اجرای بالایی است احتیاج به انتخاب ویژگی ها ندارد.

برای اجرای الگوریتم از کلاس GaussianNB استفاده می کنیم که در آن الگوریتم GuassianNaive bayse را برای classification استفاده می کند. likelihood ویژگی ها فرض می شود که دارای فرم زیر می باشد.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

برای اینکه اطمینان حاصل شود که این روش بریا بهبود عملکرد خود احتیاج به کاهش ابعاد نداردو روش کاهش ابعاد را پیاده سازی کرده و معیار های بدست آمده را با داده های بدون کاهش مقایسه نمودیم. که در هر دو روش خود داده ها معیار های بهتری را گزارش می نمودند. که البته این موضوع از با کمی دقت بر روی روش naive bayse قابل فهم است.

معیار های دقت بدست آمده در این حالت نیز برابر است با :

Accuracy = 0.79500000000000004

F-Measure = 0.84472049689440998

Logistic Regression :

برای این قسمت از کلاس و توابع مربوط به آن با نام ¶sklearn.linear_model.LogisticRegression استفاده می نماییم. prototype تابع به ضورت زیر می باشد :

```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr',
verbose=0, warm_start=False, n_jobs=1) ¶                                                                    [source]
```

ماهیت روش logistic regression به گونه ای است که احتیاج به نرمال سازی داده ها ندارد و این کار اثری بر روی الگوریتم ندارد پس پیش پردازشی در این حالت نداریم.

پارامتر هایی که برای داده هایمان ست خواهیم کرد penalty و C می باشد که با توجه به معیار دقت بدست آمده بر روی ترکیب آنها تصمیم می گیریم کدام یک را برگزینیم. جدول زیر مقادیر مختلف C و penalty را بررسی نموده است که نتایج در آن آورده شده اند :

Penalty = L1

C	0.0001	0.001	0.01	0.1	1	10
Accuracy	0.38100000000000001	0.61899999999999999	0.72840000000000005	0.78839999999999999	0.82340000000000002	0.82340000000000002
F-Measure	0.0	0.76466954910438545	0.80677290836653381	0.83792892156862742	0.86175043056207923	0.86162043566839053

Penalty = 'L2'

C	0.0001	0.001	0.01	0.1	1	10
Accuracy	0.61899999999999999	0.61899999999999999	0.72640000000000005	0.7984	0.82099999999999995	0.82399999999999995
F-Measure	0.76466954910438545	0.76466954910438545	0.81573275862068972	0.84805547181187824	0.86143365846106212	0.86250000000000004

همانطور که در بالا مشاهده می کنیم بیشینه مقدار به ازای L2 و نیز c=10 بدست می آید.

در نهایت خروجی برای داده های test محاسبه می شود و در فایل bin_test_logistic قرار می گیرد. برای خواندن اطلاعات فایل همانطور که در قسمت اول نیز اشاره شد به صورت زیر عمل می کنیم

```
import pickle
f = open('bin_test_logistic', 'r')
yTestPredic = pickle.load(f)
```

۲) چند کلاسه :

SVM :

برای این حالت نیز در ابتدا به سراغ انتخاب ویژگی می رویم. مشابه حالت دو کلاسه در اینجا نیز از تابع LinearSVC با پارامتر C=0.02 استفاده می کنیم. که حاصل آن استخراج

همچنین برای این حالت نیز مشابه با حالت دو کلاسه ابتدا باید داده ها را scale نماییم تا نتیجه بهتری را بگیریم و وزن تمان ویژگی ها برابر باشد.

در ابتدا با استفاده از linearSVC پیاده سازی می کنیم که در واقع همان svm است با کرنل linear نتایج برای C های مختلف به صورت زیر می باشند :

Linear SVM(one vs rest)

C	0.0001	0.001	0.01	0.1	1	10
Accuracy	0.213799999 99999999	0.397600000 00000001	0.490599999 99999998	0.546799999 99999995	0.566999999 99999995	0.569400000 00000002
Macro_F1	0.094932583 822372629	0.252777750 60149371	0.351965567 79517379	0.437570010 72783108	0.473902302 77022316	0.478467962 91208266
Micro_F1	0.213799999 99999999	0.397600000 00000006	0.490599999 99999998	0.546799999 99999995	0.566999999 99999995	0.569400000 00000002

اما همانطور که مشخص است در بهترین حالت دقت ۰/۵ است برای معیار های مختلف که دقت مناسبی به نظر نمی رسد برای همین به سراغ svm غیر خطی می رویم.

مطابق با قبل به سراغ کرنل rbf می ریم. همچنان احتیاج به انتخاب مناسب دو پارامتر C و گاما داریم.

مثل قبل از میان مقادیر مختلف c که در جدول زیر قابل مشاهده می باشد و با قرار دادن متغیر گاما به صورت ثابت در این قسمت معیار های سنجش محاسبه می نماییم. و از بین آنها بهترین C را بر می گزینیم :

RBF SVM(choose C)

C	0.0001	0.001	0.01	0.1	1	10	100	1000
Accuracy	0.042200000000000001	0.042200000000000001	0.1158	0.5437999999999995	0.6049999999999998	0.6173999999999995	0.6044000000000005	0.6041999999999996
Macro_F1	0.00188331481258 33991	0.00188331481258 33991	0.0503620217120905 18	0.4333326136668986 3	0.5421058084894294 6	0.6175	0.5496576970798738 9	0.5493842871595568
Micro_F1	0.042200000000000001	0.042200000000000001	0.1158	0.5437999999999995	0.6049999999999998	0.6176	0.6044000000000005	0.6041999999999996

همانطور که در بالا مشخص است، مقداری از c که منجر به مقدار بهتر معیار ها می شود عبارت است از ۱۰. بنابراین با این مقدار به سراغ تعیین پارامتر گاما می ریم.

RBF SVM(choose gamma, C = 10)

gamma	0.00001	0.0001	0.001	0.01	0.1	1	10
Accuracy	0.042200000000000001	0.256000000000000001	0.559000000000000005	0.5887999999999999 9	0.6173999999999995	0.6191999999999997	0.042200000000000001
Macro_F1	0.00188331481258 33991	0.14782354604540124	0.46133360044644639	0.52801018343464723	0.56351328778504473	0.5568637650657 11	0.00188331481258 33991
Micro_F1	0.042200000000000001	0.256000000000000001	0.559000000000000005	0.5887999999999999 9	0.6173999999999995	0.6191999999999997	0.042200000000000001

همانطور که در بالا قابل مشاهده است به ازای gamma =1 معیار ها مقادیر مناسبی خواهند داشت.

بنابراین هر چند دقت به اندازه دلخواه بالا نیست اما پارامتر ها به صورت زیر تنظیم می شوند :

C=10, gamma=1

همانطور که می بینیم کرنل rbf بهتر از linear عمل کرده است پس برای پیش بینی به سراغ کرنل rbf می رویم.

مقادیر پیش بینی شده با این پارمتر ها در فایل با نام multiClass_test_svm ذخیره شده است. برای خواندن اطلاعات فایل از دستور زیر می توان استفاده کرد:

```
f = open('multiClass_test_svm', 'r')
yTestPredic = pickle.load(f)
```

KNN

برای این قسمت نیز ابتدا ویژگی های مورد احتیاج را انتخاب می کنیم.و سپس با استفاده از تابع NearestNeighbors استفاده شده در قسمت دو کلاسه دسته بندی را انجام می دهیم.

مثل قبل چالشی که با آن مواجه هستیم انتخاب درست k می باشد. از میان مقادیر مختلف k نتیجه معیار های مختلف را مشخص کرده و در زیر آورده ایم.

KNN(choose K)

K	5	10	20	25	50	100
Accuracy	0.5565999999999998	0.5814000000000003	0.5951999999999995	0.5948	0.5817999999999998	0.5719999999999995
Macro_F1	0.5565999999999998	0.5814000000000003	0.5953	0.5948	0.5817999999999998	0.5719999999999995
Micro_F1	0.4960858782795325	0.5103091516049758	0.5954	0.5225470738123155	0.5015781195567086	0.4753342471303635

KNN(choose K)-1

K	250	500	1000	2500	5000	10000
Accuracy	0.5444	0.5230000000000002	0.4988000000000002	0.4477999999999998	0.3910000000000001	0.2647999999999998
Macro_F1	0.5444	0.5230000000000002	0.4988000000000002	0.4477999999999998	0.3910000000000001	0.2647999999999998
Micro_F1	0.4366530932261694	0.4052145517826146	0.3699567663568282	0.3094742114277011	0.2511184918482067	0.1348913130103706

همانطور که در بالا می بینیم بیشینه مقدار معیار های مختلف به ازای K = 20 بدست می آید. به همین دلیل classifire را بر روی مجموعه داده های train و validation (به منظور بالا رفتن دقت از ترکیب داده های train و validation استفاده می کنیم) آموزش می دهیم و با استفاده از آن برچسب ها را پیش بینی می کنیم.

دلیل اینکه کلا میزان دقت برای داده های چند کلاسه پایین تر از دو کلاسه آمده است این است که با افزایش تعداد کلاس ها به 40 کلاس و ثابت باقی ماندن تعداد نمونه ها پیدا کردن الگوی درست در داده برای classificaion سخت تر می شود.

مقادیر پیش بینی شده با این پارمتر ها در فایل با نام multiClass_test_knn ذخیره شده است. برای خواندن اطلاعات فایل از دستور زیر می توان استفاده کرد:

```
f = open('multiClass_test_knn', 'r')
yTestPredic = pickle.load(f)
```

Logistic Regression :

در حالت چند کلاسه تابع logistic regression این کتابخانه از روش one vs rest استفاده می کند.

برای این کار مشابه با روش دو کلاسه عمل می کنیم و خطای validation را محاسبه می کنیم و با ترکیب داده های train و validation و به دست آوردن مجموعه بزرگ تری برای آموزش مدل را آموزش داده و برچسب ها را پیش بینی می نماییم.

معیار های دقت خواسته شده برای مقادیر مختلف C و penalti در زیر آورده شده است.

Penalty = 'L1'

C	0.001	0.01	0.1	1	10	100
Accuracy	0.042200000000000001	0.12479999999999999	0.47499999999999998	0.63300000000000001	0.65639999999999998	0.61539999999999995
Macro_F1	0.001883314812583399	0.032629437332303356	0.34961252099481911	0.55829325656513251	0.60358670444174840	0.56358442119169105
Micro_F1	0.042200000000000001	0.12479999999999999	0.47500000000000003	0.63300000000000001	0.65639999999999998	0.61539999999999995

Penalty = 'L2'-1

C	0.001	0.01	0.1	1	10	100
Accuracy	0.186	0.35320000000000001	0.53720000000000001	0.63500000000000001	0.6714	0.65600000000000003
Macro_F1	0.079932101744003753	0.21675951217294889	0.40554764594562093	0.55478591464972493	0.6714	0.60152769145099272
Micro_F1	0.186	0.35320000000000001	0.53720000000000001	0.63500000000000001	0.6714	0.65600000000000003

همانطور که در تصویر مشخص است به ازای Penalty =L2 و همچنین C =10 بهترین جواب را داریم. پس مقادیر خروجی test را با این پارامتر ها آموزش داده و نتیجه را در فایلی با نام multiClass_test_logistic ثبت می نماییم.

Naive bayes

در این قسمت مشابه با قسمت دو کلاسه عمل می کنیم. دقت های خواسته شده عبارتند از :

Accuracy = 0.59660000000000002

F1Macro = 0.56125307629578935

F1Micro = 0.59660000000000002

مقادیر خروجی test را با این پارامتر ها آموزش داده و نتیجه را در فایلی با نام multiClass_test_naive ثبت می نماییl.

Regression :

Linear Regression :

برای این قسمت پیاده سازی رگرسیون خطی را با regularization term از نوع ridge انجام میدهیم. در این حالت خطی و ساده باید پارامتر alpha که بیان کننده میزان جریمه ضرایب است را مشخص نماییم.

برای بدست آوردن معیار RMSE نیز از تابع mean_squared_error موجود در sklearn.metrics استفاده می کنیم و از آن جذر می گیریم.

MAE نیز مخفف mean absolute error است . که برا یاین معیار نیز از تابع mean_absolute_error استفاده می کنیم.

در ابتدا بدون transformation از فضای ویژگی استفاده می کنیم و به ازای alpha های مختلف میزان خطا را محاسبه نموده تا کمترین خطا را مشخص نماییم.

alpha	0.00001	0.0001	0.001	0.01	0.1	1	10	100	1000
RMSE	15.4219 8750412 214	15.4219 7920579 222	15.4218 9626587 9724	15.4210 7119233 6685	15.4132 4028802 4154	15.3670 1140550 973	15.6119 1486373 7745	17.7505 3275830 4594	21.9754 2223991 7446
MAE	10.7678 3465813 4829	10.7678 2364427 8733	10.7677 1352608 4265	10.7666 1910718 8298	10.7561 9234106 7393	10.6921 8737915 2946	10.8427 6764571 3504	12.5812 4779252 7186	15.3101 3049662 9936

میزان کمینه خطا به ازای alpha = 1 است.

همچنین رگرسیون را با regularization term از نوع lasso نیز پیاده سازی کردیم که دقت بالاتری نسبت به قبل دریافت ننمودیم.

پس سراغ polynomial transformation می رویم....!!!! اما به دلیل بالا بودن تعداد ویژگی ها و عدم توانایی لپ تاپ اینجانب و قفل کردن سیستم در حین اجرا امکان ثبت نتایج ممکن نبود. بنابراین با همین حالت ساده مقادیر خروجی را پیش بینی می کنیم.

KNN Regression

برای این مورد نیز از تابع نوشته شده در کتابخانه استفاده می کنیم. و میزان اثر گذاری هر همسایه را از طریق عکس میزان فاصله ای که با نقطه مورد نظر دارد دخیل می کنیم.

در این حالت نیز باید تعداد همسایه ها را به گونه ای مشخص نماییم که کمترین خطای پیش بینی را داشته باشم.

K	5	10	20	50	100
RMSE	13.831749033842394	13.431455388996383	13.283952052376959	13.574629330814451	13.976800911501243
MAE	7.320742799999997	7.484569400000007	7.690298099999997	8.207610240000011	8.713486460000004

همانطور که در جدول بالا مشاهده می کنیم مقدار کمینه دو خطا یکی نیست، برای RMSE تعداد ۲۰ همسایه مناسب است و برای MEA تعداد ۵ همسایه. حال باید کمی دقت کنیم که این تفاوت ناشی از چیست؟ این تفاوت ناشی از ماهیت این دو خطا است. RMSE وزن بیشتری به خطا های بزرگ می دهد در حالی که MAE برای تمام خطا ها وزن یکسانی را در نظر می گیرد. اینجا دوست نداریم که پیش بینی مان پرت باشد و خیلی تفاوت داشته باشیم بنابراین سعی می کنیم که خطا های بزرگ را کمینه کنیم، پس به سراغ RMSE می رویم و با تعدا ۲۰ همسایه پیش بینی را انجام می دهیم. نتایج در فایلی با نام reg_test_KNNReg قرار دارند.

توضیح درباره کد :

همانطور که در بالا مشخص است برای پیدا کردن پارامتر های مختلف بسته به نوع مسئله و گاهی احتیاج به چک کردن بیش از یک راه حل برای پیدا کردن جواب بهتر کد ها معمولا دارای بلاک های کامنت شده هستند که این قسمت های کامنت شده در واقع قسمت هایی از کد می باشند که برای بدست آوردن پارامتر های بهینه مورد استفاده بوده اند.