به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس داده کاوی

پروژه پایانی

نام و نام خانوادگی: حسین سیفی

شماره دانشجویی : ۸۱۰۱۰۰۳۸۶

خرداد ۱۴۰۱

فهرست

هدف پروژه

در این پروژه قصد داریم به کمک مجموعه داده خسارات ناشی از زلزله، مدلی مناسب جهت پیش‌بینی خسارات با توجه به ویژگی‌های ساختمان مورد نظر ایجاد کنیم. برای ایجاد این مدل از روش‌های یادگیری ماشین و شبکه‌های عصبی مصنوعی استفاده می‌کنیم و تلاش می‌کنیم با تغییر پارامترهای مربوط به هر کدام از مدل‌ها، به بهترین دقت ممکن از پیش‌بینی نتایج برسیم.

مرحله اول: آماده‌سازی ویژگی‌ها

برای استفاده از یک مجموعه داده در فرآیند یادگیری ماشین و آموزش و تست مدل به کمک آن مجموعه داده، نیاز داریم تا پیش پردازش‌های مناسب را روی آن انجام دهیم. این پیش پردازش می‌تواند شامل نرمال‌سازی مقادیر، حذف مقادیر نامعتبر و پر کردن فضاهای خالی در هر ویژگی به روش مناسب باشد (روش مناسب با توجه به نوع داده‌های ویژگی، مفهوم آن ویژگی در مجموعه داده و مقادیر سطرهای دارای معتبر انتخاب می‌شود.). همچنین به این دلیل که الگوریتم‌های یادگیری ماشین بر روی داده‌های عددی کار می‌کنند نیاز داریم تا ویژگی‌هایی که مقدار غیر عددی مانند تاریخ و رشته دارند را با استراتژی مناسب به مقادیر عددی نگاشت کنیم. در ادامه ابتدا به پیش پردازش داده‌ها و سپس انتخاب تاثیرگذارترین ویژگی‌ها برای به‌دست آوردن بهترین مقادیر معیارهای ارزیابی اعم از دقت و فراخوانی با روش‌های متفاوت می‌پردازیم.

در ابتدا از بین ویژگی‌های موجود در فایل‌های Labels و Values مجموعه داده‌ی موجود، ستون building_id را حذف کردیم چرا که این ستون نشان‌دهنده ویژگی نیست و یک شناسه یکتا ست که نه تنها تأثیر مثبتی در نتیجه نهایی ندارد بلکه می‌تواند باعث افت عملکرد سیستم شود.

تبدیل ویژگی‌های غیر عددی به عددی

در مجموعه داده موجود تعداد ۸ ویژگی دارای مقادیر غیر عددی شامل حروف هستند که باید به اعداد تبدیل شوند. این ویژگی‌ها به شرح زیر می‌باشند:

1. Land_surface_condition
2. Foundation_type
3. Roof_type
4. Ground_floor_type
5. Other_floor_type
6. Position
7. Plan_configuration
8. Legal_ownership_status

استراتژی مورد نظر بدین صورت است که ابتدا لیستی از تمام مقادیر موجود یک ویژگی ایجاد می‌شود و سپس مقادیر غیر عددی با اندیس خود در لیست ایجاد شده جایگزین می‌شوند. عمل ذکر شده به کمک قطعه کد زیر انجام می‌شود:

```
land_condition_encode = list(set(values['land_surface_condition']))
values['land_surface_condition'] = values.land_surface_condition.map(la
mbda x: land_condition_encode.index(x))
```

کد فوق جایگزینی مقادیر غیر عددی با عددی را برای ویژگی land_condition نشان می‌دهد. در این مرحله مشکلی که ایجاد می‌شود این است که این ویژگی‌ها تعداد کمی مقدار ممکن دارند و ویژگی‌هایی مانند geo_level_3_id که مقادیر بسیار گسترده‌تر و بیشتری دارند تأثیر بیشتری روی مدل‌هایی که در آینده آموزش خواهیم داد می‌گذارند و در نتیجه می‌توانند ما را از نتایج بهتر دور کنند. بنابراین نیاز داریم تا از روشی برای نرمال‌سازی مقادیر ویژگی‌ها استفاده کنیم تا هیچ ویژگی خاصی به‌دلیل بازه مقادیر ممکن تأثیر بیشتری از دیگر ویژگی‌ها نداشته باشد. در بخش بعد روش‌های برای حل این مشکل معرفی و پیاده‌سازی خواهیم کرد.

نرمال‌سازی مقادیر ویژگی‌ها

همانطور که گفته شد وجود ویژگی‌هایی با بازه‌های متفاوت می‌توانند بر نتیجه نهایی یک مدل یادگیری ماشین تأثیر منفی بگذارند و مقادیر ویژگی‌ها باید به روش‌های مناسب نرمال شوند. سه روش Minmax و Zscore روش‌هایی برای نرمال‌سازی داده‌ها هستند که در ادامه معرفی و پیاده‌سازی می‌شوند:

۱. Minmax: در این روش بیشترین مقدار موجود به مقدار ۱ و کمترین مقدار موجود به عدد ۰ نگاشت می‌شود و سایر مقادیر به‌صورت خطی به مقادیری بین ۰ تا ۱ نگاشت می‌شوند. مزیت این روش سادگی و قابل درک بودن و سادگی در پیاده‌سازی آن است و ایراد آن همان نگاشت خطی مقادیر است که می‌تواند باعث از بین رفتن توزیع واقعی مقادیر شود برای مثال مقادیری با توزیع نرمال پس از اعمال Minmax دیگر توزیع نرمال نخواهند داشت. در ادامه نحوه نرمال‌سازی به این روش به کمک کتابخانه sklearn نمایش داده شده است. همانطور که مشاهده می‌شود این نرمال‌سازی و نرمال‌سازی Zscore که در ادامه معرفی می‌شود روی تمامی داده‌ها اعمال می‌شود و در صورت اعمال به‌صورت جداگانه روی داده‌های آموزش و ارزیابی به نتایج درستی منجر نمی‌شود.

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
values_minmax = min_max_scaler.fit_transform(values)
```

۲. Zscore: این روش نرمال‌سازی به کمک فرمول $Zscore = \frac{x-\mu}{\sigma}$ محاسبه می‌شود که σ نشان‌دهنده انحراف معیار و μ نشان‌دهنده میانگین مقادیر یک ویژگی است. نرمال‌سازی به این روش برخلاف روش Minmax توزیع مقادیر را حفظ می‌کند و می‌تواند انتخاب مناسب‌تری نسبت به روش قبلی باشد. پیاده‌سازی در زبان پایتون به شکل زیر می‌باشد و همانطور که مشخص است برای محاسبه مقادیر میانگین و انحراف معیار از توابع آماده موجود در کتابخانه Numpy استفاده شده است.

```
norm_values = (values-np.mean(values))/np.std(values)
```

اگرچه روش‌های دیگری نیز برای نرمال‌سازی داده‌ها وجود دارد اما به پیاده‌سازی و اعمال دو روش فوق قناعت می‌کنیم و با توجه به تعریف مناسب‌تر روش Zscore با این روش در بخش‌های بعدی ادامه می‌دهیم.

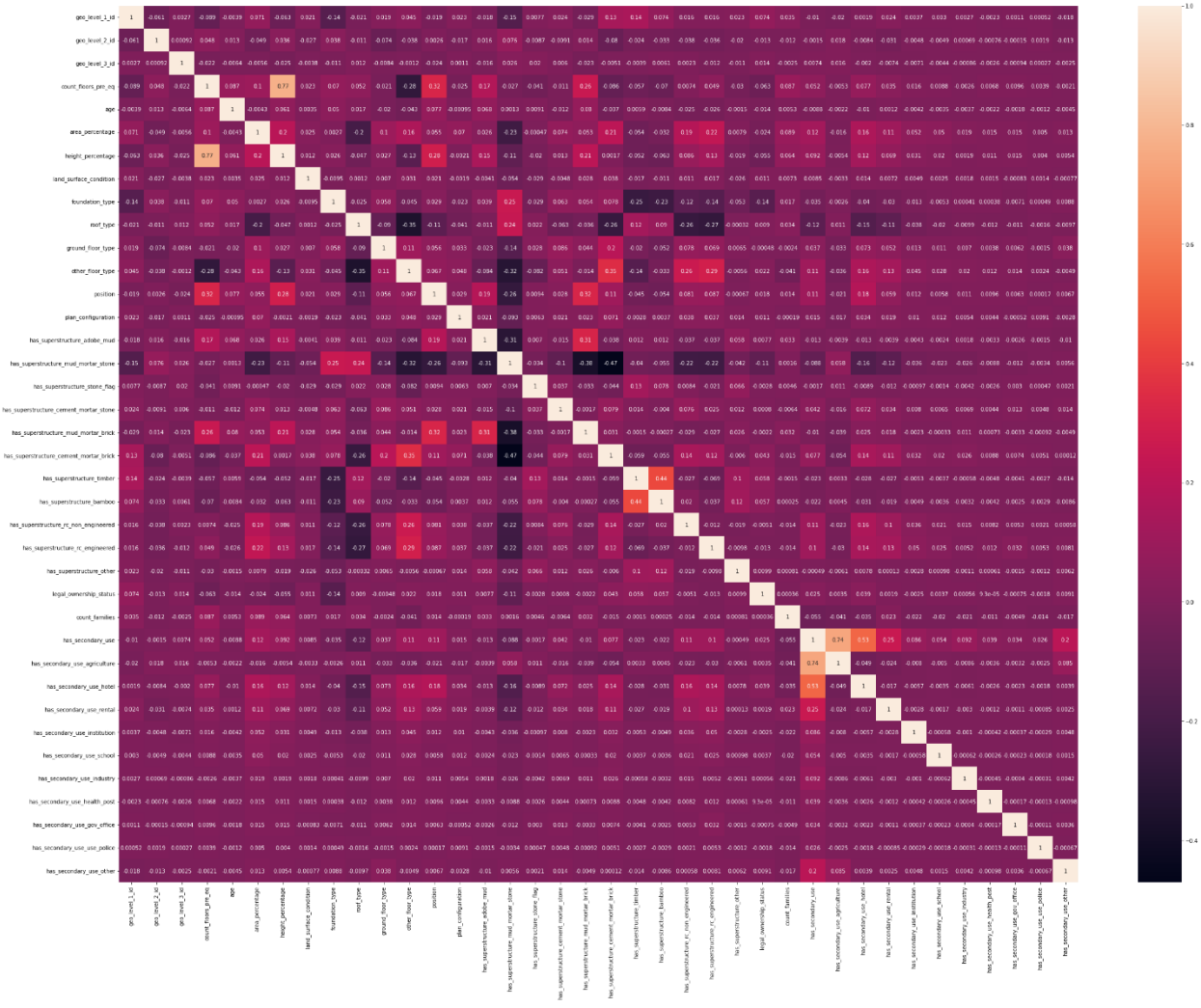
کاهش ویژگی‌ها

روش‌های متعددی برای کاهش ویژگی از جمله Sequential Forward Feature Selection و Sequential Backward Feature Elimination را می‌توان روی مجموعه داده اعمال کرد و نتایج هر کدام را بررسی کرد اما در ابتدا می‌توان با محاسبه وابستگی (Correlation) بین ویژگی‌ها، تعدادی از ویژگی‌هایی که با یکدیگر اکیداً وابسته هستند (Strongly correlated) را از مجموعه داده حذف کرد.

وابستگی با فرمول $r = \frac{\sum(x_i-\mu_x)\,(y_i-\mu_y)}{\sqrt{\sum(x_i-\mu_x)^2\,\sum(y_i-\mu_y)^2}}$ محاسبه می‌شود و نشان می‌دهد دو ویژگی چقدر در صعود و نزول مقادیر خود به یکدیگر شباهت دارند. این ضریب برای هر زوج از ویژگی‌ها به کمک قطعه کد زیر محاسبه می‌شود و سپس در یک نمودار این مقادیر نمایش داده می‌شوند.

```
cor = values.corr()
plt.figure(figsize=(40,30))
sb.heatmap(cor, annot=True)
```

نتیجه این کد به شکل زیر می‌باشد:



نتایج خروجی وابستگی را می‌توان بدین صورت تفسیر کرد که ویژگی‌های با مقدار r بیشتر از ۰.۵ اکیدا وابسته هستند و با توجه به مفهوم آن ویژگی‌ها در مجموعه داده می‌توان یکی از آن‌ها را حذف کرد. با توجه به نمودار فوق ویژگی has_secondry_use و ویژگی has_secondry_use_agriculture اکیدا وابسته هستند. از طرفی دیگر ویژگی has_secondry_use با تعدادی از ویژگی‌های دیگر نیز وابستگی بالایی دارد(اگرچه اکیدا وابسته نیستند) و می‌توان این ویژگی را برای حذف از مجموعه داده انتخاب

کرد. همچنین ویژگی‌های height_percentage و count_floors_pre_eq نیز اکیدا وابسته هستند که حذف هر کدام از آن‌ها با دیگری تفاوتی ندارند و به انتخاب یکی از آن‌ها را می‌توان حذف کرد. دو ستون انتخاب شده به کمک دستور زیر حذف می‌شوند که یکی از دستورات پیش‌فرض کتابخانه pandas است.

```
norm_values = norm_values.drop(columns=['has_secondary_use', 'height_pe
rcentage'], axis=1)
```

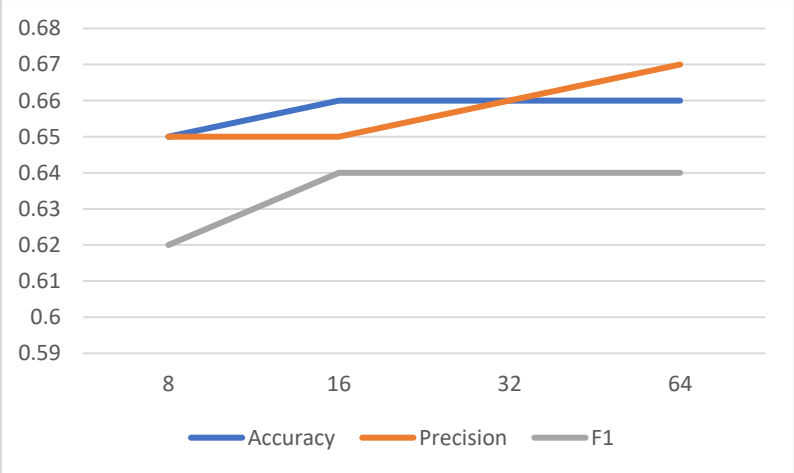
پس از حذف چندین ویژگی مرتبط با هم، می‌توانیم از الگوریتم‌های انتخاب ویژگی برای استخراج ویژگی‌های تاثیرگذارتر مانند Sequential forward feature selection یا Sequential backward feature elimination استفاده کنیم. در ادامه این دو روش به صورت جزئی معرفی می‌شوند:

۱. Sequential forward feature selection: در این روش یک معیار ارزیابی به عنوان نمایانگر کیفیت مدل و یک مدل برای یافتن بهترین مجموعه ویژگی‌ها روی این مدل انتخاب می‌شوند. ابتدا یک مجموعه تهی به عنوان ویژگی‌های انتخاب شده ایجاد می‌شود و در هر مرحله یک ویژگی که بیشترین مقدار را برای معیار ارزیابی ایجاد می‌کند به مجموعه ویژگی‌ها اضافه می‌شود. تعداد ویژگی‌های انتخاب شده به عنوان مقداری پیش از شروع الگوریتم انتخاب می‌شود. پیاده‌سازی این روش در زبان پایتون با استفاده از کتابخانه sklearn به شکل زیر می‌باشد.

```
sfs = SequentialFeatureSelector(model, n_features_to_select=17, sc
oring='accuracy')

sfs.fit(norm_values, labels['damage_grade'])
```

۲. Sequential Backward feature elimination: در این روش نیز همانند روش قبلی یک مدل و معیار ارزیابی وجود دارد. برای یافتن بهترین مجموعه ویژگی‌ها این روش ابتدا مجموعه تمامی ویژگی‌ها را در نظر می‌گیرد و هر بار ویژگی را از مجموعه حذف می‌کند که باعث بیشترین کاهش مقدار معیار ارزیابی شده است و این کار را تا رسیدن به تعداد ویژگی‌های مورد نظر انجام می‌دهد.



روش اول با مقادیر متفاوت برای تعداد ویژگی‌ها اجرا و بررسی شد اما متاسفانه هر بار با کاهش مقدار معیارهای ارزیابی مانند Accuracy، Precision و F-measure نسبت به حالتی که این روش‌ها استفاده نشده‌اند مشاهده شد بنابراین به نظر می‌رسد که حذف معیارهای ارزیابی مشابه(همانطور که در جدول وابستگی‌ها توضیح داده شد) کافی است و انجام هر فعالیت اضافی در رابطه با انتخاب ویژگی‌ها منجر به کاهش کیفیت مدل می‌شود. دقت‌های به دست آمده از مدل MLP به کمک روش اول در نمودار فوق قابل مشاهده است.

Class Imbalance

همچنین در داده‌های موجود مشکل نامتعادل بودن کلاس‌ها نیز وجود دارد به صورتی که در کلاس ۱، ۲ و ۳ به ترتیب ۲۵۱۲۴، ۱۴۸۲۵۹ و ۸۷۲۱۸ سطر داده وجود دارد. برای این مشکل می‌توان از روش‌هایی مانند SMOTE استفاده کرد تا اندازه هر کلاس برابر با بزرگترین کلاس موجود شود یا از Sampling برای کاهش اندازه کلاس‌ها به اندازه کوچکترین کلاس استفاده شود. دو روش فوق در ادامه معرفی می‌شوند.

۱. SMOTE: این روش یک تکنیک Oversampling است که برای کلاسی با کوچکترین سایز انجام می‌شود. ساده‌ترین روش با هدف Oversampling تکرار سطرهای داده در کلاس‌های کوچکتر است تا اندازه کلاس‌ها با هم برابر شود اگرچه این روش اطلاعات جدیدی را به مجموعه داده اضافه نمی‌کند. روش‌های دیگری نیز برای ایجاد نمونه‌های جدید از روی نمونه‌های موجود وجود دارد که در همین دسته قرار می‌گیرند. در قطعه کد زیر این روش با استفاده از کتابخانه‌های موجود روی مجموعه داده اعمال شده است. پس از اجرای این قطعه کد، سایز تمام کلاس‌ها به ۱۴۸۲۵۹ تغییر می‌کند.

```
from imblearn.over_sampling import SMOTE

oversample = SMOTE()
X, y = oversample.fit_resample(norm_values, labels['damage_grade'
])
```

۲. Sampling: در این تکنیک تعدادی از داده‌های هر کلاس به منظور یکسان‌سازی اندازه کلاس‌ها انتخاب می‌شود. این تعداد انتخاب شده می‌تواند به اندازه کوچکترین کلاس باشد تا نیازی به کپی کردن سطرهای داده یا ایجاد سطر جدید برای کلاس کوچکتر نیازی نباشد. انتخاب سطرها می‌تواند به صورت تصادفی باشد یا با محاسبه شباهت بین سطرهای موجود در هر کلاس سطرهایی را انتخاب کرد که کمترین شباهت را دارند و در نتیجه باعث افزایش دقت در مدل نهایی آموزش دیده شوند.

متاسفانه هیچ یک از روش‌های فوق نیز باعث پیشرفت دقت و دیگر معیارهای ارزیابی در مدل نهایی نشدند و باعث افت کردن این معیارها نسبت به مدلی که از تمامی سطرها(اگر چه نامتقارن) استفاده می‌کند شدند بنابراین ترجیح بر این است که از این دست روش‌ها نیز استفاده نشود و تنها با تبدیل داده‌های غیرعددی به عددی و نرمال‌سازی داده‌ها مرحله پیش پردازش را به اتمام برسانیم و به مرحله آموزش و تست مدل برسیم.

مرحله دوم: یادگیری و انتخاب مدل مناسب

در این مرحله به ایجاد مدل‌های مختلف و یافتن بهترین پارامترهای هر مدل می‌پردازیم.

ایجاد مدل‌ها

در ابتدا با استفاده از کتابخانه‌های مناسب برای هر یک از موارد خواسته شده یک مدل خام تعریف می‌کنیم.

```
Model = MLPClassifier(alpha=a, hidden_layer_sizes = size
, validation_fraction=0.5, activation='tanh', max_iter=500, learning_ra
te_init=lr)
```

به وسیله دستور فوق مدل Multi-Layer Perceptron ایجاد می‌شود با اندازه لایه نهان size، مقدار نرخ یادگیری lr، مقدار آلفا a و تابع فعال‌ساز Tanh ایجاد می‌شود.

```
Model = svm.SVC(kernel=k)
```

و به وسیله دستور فوق مدل Svm با کرنل k ایجاد می‌شود.

بهترین مقادیر هایپر پارامترها

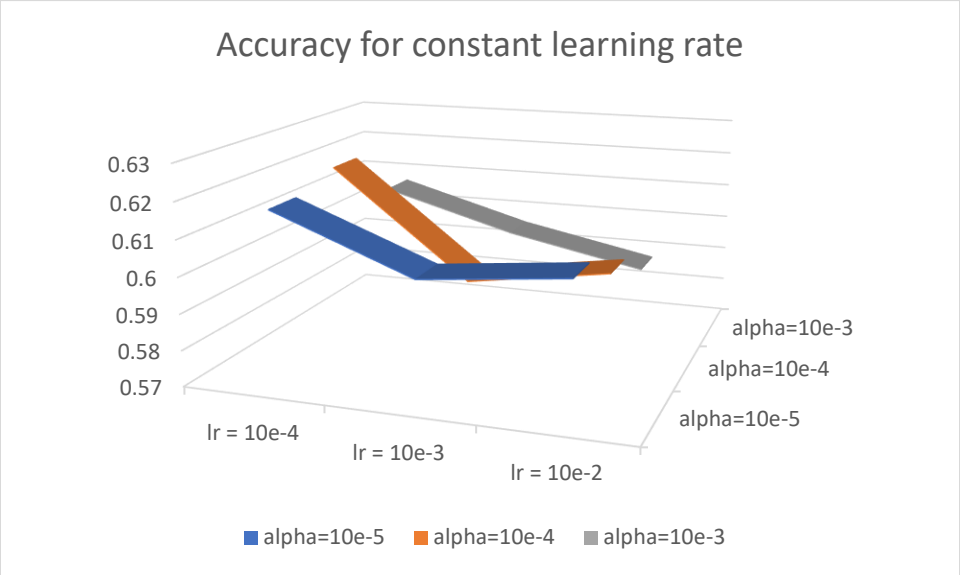
همانطور که میدانیم نمی‌توان مقادیر مناسبی برای هایپر پارامترهایی مانند نرخ یادگیری و مقدار آلفا در مراحل مختلف الگوریتم‌های شبکه عصبی و یادگیری ماشین یافت و باید مستقیما مقدار دهی شوند. برای تعیین پارامترهای موجود در MLP می‌توان از Grid search استفاده کرد. در این روش مقادیر مختلف را برای پارامترهای مورد نظر مدل ایجاد شده مشخص می‌کنیم و با این روش جستجو با بهینه سازی‌های مورد نیاز مدل‌هایی با مقادیر پارامترهای متفاوت ایجاد می‌شوند و مقدار معیار ارزیابی مشخص شده برای هر یک از مدل‌ها گزارش می‌شود تا بهترین مقادیر انتخاب شوند. لازم به ذکر است که توابع Grid Search موجود در کتابخانه‌های پایتون از k-fold cross validation نیز استفاده می‌کنند که مقدار k قابل تعیین کردن است و با این روش می‌توان بهترین نتیجه از تقسیمات مختلف داده به آموزش و تست را به دست آورد. این عمل به کمک دستورات زیر قابل انجام است.

```
parameters = {'alpha':(1e-5, 1e-4, 1e-3), 'learning_rate_init':(1e-
4, 1e-3, 1e-2), 'learning_rate':('constant', 'adaptive')}

model = MLPClassifier(activation='tanh', validation_fraction=0.25, hidd
en_layer_sizes=32, max_iter=500)
clf = GridSearchCV(model, parameters, verbose=3, scoring="accuracy", cv
=5)
clf.fit(norm_values[:10000], labels[:10000]['damage_grade'])
```

همانطور که در دستورات فوق قابل مشاهده است مدلی با اندازه لایه نهان ۳۲ ایجاد شد و مقادیر بر روی آن بررسی شدند تا در مرحله بعد با مقادیر به دست آمده برای هایپر پارامترهای متفاوت مدل‌هایی با سایز لایه‌های نهان ۸ ، ۱۶ و ۶۴ نیز بررسی شوند. همچنین برای سه هایپر پارامتر alpha، learning_rate_init و learning_rate مقادیر متفاوتی در نظر گرفته شده‌اند تا به کمک grid search بهترین مقدار آن انتخاب شود. برای alpha و learning_rate_init مقادیر پیش فرض به مقدار ده برابر و

۰.۱ برابر آن‌ها را برای بررسی انتخاب کرده‌ایم و برای learning_rate دو مقدار مناسب‌تر بین سه حالت متفاوت انتخاب شده‌اند. در نهایت به کمک ۱۰۰۰۰ سطر از داده‌های موجود نتایج مختلف برای این مقادیر بررسی می‌شوند. برای قطعه کد فوق در مجموع ۹۰ مدل آموزش داده شد و نتایج گزارش شدند که این نتایج برای مقدار نرخ یادگیری ثابت به شرح زیر می‌باشند:



همانطور که از نمودار فوق مشخص است بهترین مقدار برای دقت در مدل‌های MLP با مقدار آلفا و نرخ یادگیری هر کدام برابر ۰.۰۰۰۱ به دست می‌آید. این مقدار نرخ یادگیری نشان می‌دهد که برای دست‌یابی به بهترین مقادیر معیارهای ارزیابی نیاز به گام‌هایی کوچک‌تر از مقدار مرسوم در جهت یافتن مقدار بهینه داریم. این نکته می‌تواند به معنی نویزی بودن بیش از اندازه نمودار تابع مقادیر باشد.

همچنین به دلیل اینکه در تمامی حالات مقادیر به دست آمده به وسیله نرخ یادگیری adaptive کمتر از مقادیر نمودار فوق هستند از رسم نمودار برای نرخ یادگیری adaptive صرف نظر کردیم.

آموزش مدل

در این مرحله به آموزش مدل‌ها می‌پردازیم. در ابتدا داده‌های مجموعه داده را به مجموعه‌های آموزش+اعتبار سنجی و تست تقسیم می‌کنیم که این مقادیر به ترتیب برابر ۴۰ درصد و ۶۰ درصد داده‌ها هستند. این وظیفه به کمک دستور زیر انجام می‌شود. همچنین این داده‌ها بر اساس برچسب که همان ستون damage_grade است نیز به صورت stratified نمونه گیری می‌شوند.

```
X_train, X_test, y_train, y_test = train_test_split(norm_values, labels
['damage_grade'], stratify=labels['damage_grade'], test_size=0.6)
```

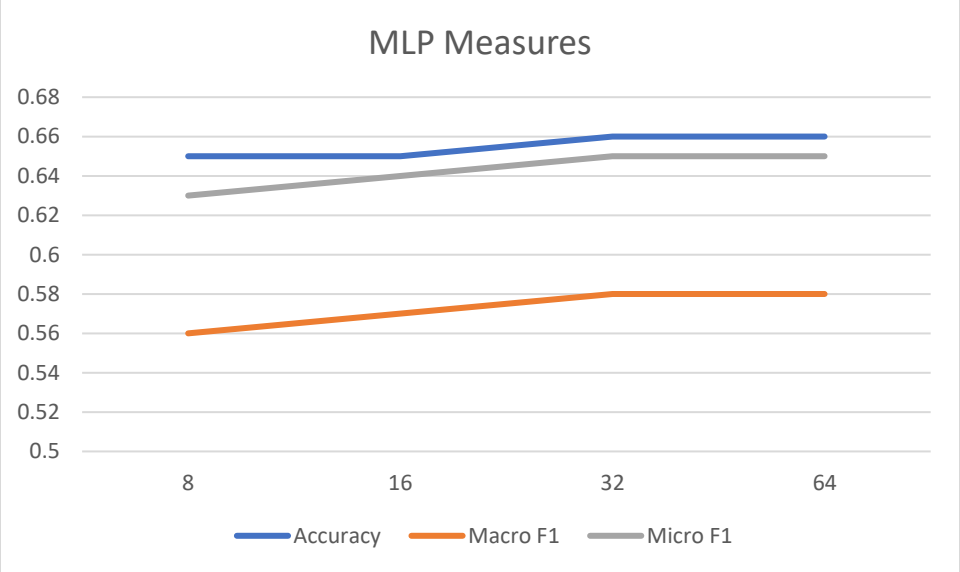
سپس مدل‌ها به کمک داده‌های تقسیم شده آموزش داده می‌شوند. مدلهای MLP به کمک دستورات زیر آموزش داده می‌شوند و پیش بینی هر مدل از داده‌های تست در یک دیکشنری ذخیره می‌شود. متاسفانه این مدل با مقادیر هایپر پارامترهای به دست آمده از روش Grid search به خوبی عمل نکرد و در نتیجه مقادیر به مقدار پیش فرض تغییر کردند.

```
for size in [8, 16, 32, 64]:
      pred = MLPClassifier(alpha=1e-
5, hidden_layer_sizes=size, validation_fraction=0.25, activation='tanh'
, max_iter=500, learning_rate_init=1e-
3).fit(X_train, y_train).predict(X_test)
      clf_pred[size] = pred
      acc = accuracy_score(y_test, pred)
      clf_acc[size] = acc
```

مدل‌های فوق را می‌توان به کمک دستور زیر از کتابخانه scikit-learn ارزیابی کرد. این دستور تمامی معیارهای شناخته شده از جمله accuracy، precision و recall و به دو صورت میانگین گیری Micro و Macro محاسبه می‌کند.

```
classification_report(y_test, clf_pred[size])
```

نتایج ارزیابی این مدل به شکل زیر خواهد بود:



همانطور که مشاهده می‌شود بهترین مقادیر هر سه معیار ارزیابی به ازای اندازه لایه نهان ۳۲ و ۶۴ به دست می‌آید. در نمودار فوق در تمامی نقاط معیار Macro F1 مقدار کمتری از Micro F1 ارائه می‌دهد که مشخصا به دلیل پایین‌تر بودن مقدار F1 برای کلاس کوچکتر یعنی کلاس ۱ اتفاق می‌افتد در حالی که با میانگین گیری وزن دار روی مقدار F1 هر کلاس یا به عبارت دیگر محاسبه Micro F1 تاثیر کلاس‌های کوچک‌تر به نسبت تعداد داده‌های آنان بر کل داده‌ها اعمال می‌شود و برای مدل‌هایی نظیر چهار مدل آموزش دیده در این بخش که مقدار F1 برای کلاس بزرگ‌تر مقدار بالاتری است، مقادیر بهتری ارائه می‌دهد. با توجه به اینکه تعداد ویژگی‌ها ۳۶ عدد است و بسیار نزدیک به تعداد نورون‌های مدل ۳۲ تایی است و مدل ۶۴ نورونی نیز دقت بیشتری ارائه نمی‌دهد می‌توان پیش‌بینی کرد که هر چه اندازه لایه از ۳۲ بیشتر شود باز هم دقت به مقدار چشم‌گیری تغییر نمی‌کند.

از آنجا که مدل SVM مدت زمان آموزش بسیار طولانی تری نسبت به MLP دارد، برای آموزش این مدل‌ها تنها از ۴ درصد داده‌ها استفاده می‌کنیم که به صورت Stratified نمونه برداری شده‌اند. این عمل با کد زیر انجام می‌گیرد:

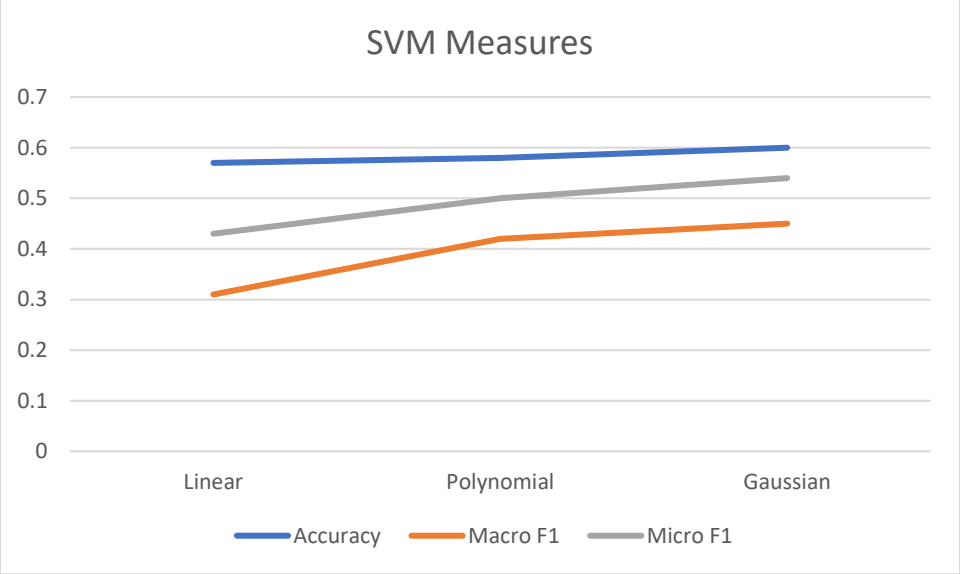
```
svm_train_set, extra, svm_train_label, extra =  train_test_split(X_trai
n, y_train, stratify=y_train, test_size=0.9)
```

همچنین به وسیله دستوری مشابه فقط بخشی از مجموعه تست را برای ارزیابی مدل انتخاب می‌کنیم.

برای آموزش مدل‌های svm با کرنل‌های متفاوت و دریافت پیش‌بینی می‌توان از حلقه زیر استفاده کرد:

```
for k in ["linear", "poly", "rbf"]:
  pred = SVC(kernel=k).fit(svm_train_set, svm_train_label).predict(svm_
test_set)
  svm_pred[k] = pred
  acc = accuracy_score(svm_test_label, pred)
  svm_acc[k] = acc
```

و در نهایت به کمک دستوری مشابه با دستور استفاده شده برای مدل‌های MLP، مدل‌های Svm را ارزیابی می‌کنیم. نتایج به شکل زیر خواهند بود:



با توجه به نمودار فوق بهترین نتیجه‌های ممکن را مدل با کرنل Gaussian ارائه می‌دهد. در این مدل‌های نیز توجیح یکسانی با مدل‌های قبل برای اختلاف معیارهای Macro-F1 و Micro-F1 وجود دارد. اگرچه مدل Gaussian دقت بالایی را ارائه می‌دهد اما این مدل زمان اجرا و آموزش بسیار طولانی تری را دارا می‌باشد و زمان بسیار زیادی را برای داده‌های حجیم نسبت به دو کرنل دیگر یعنی چندجمله‌ای(Polynomial) و خطی(Linear) دارد. در واقع با توجه به داده‌های موجود در مجموعه داده هم انتظار می‌رفت مدل Gaussian بهترین عملکرد را روی این داده‌های خاص داشته باشد چرا که این مدل یک مدل عمومی است و در مواقعی مناسب است که از دیتاست موجود دانش پیشین زیادی در دسترس نیست و این ویژگی‌ها با مجموعه داده در دسترس ما همخوانی زیادی دارد. اگرچه در نهایت بهترین نتیجه ممکن که از داده‌های موجود با کمک مدل‌های svm وجود دارد از ضعیف ترین مدل MLP نیز ضعیف‌تر است.

جمع بندی و نتیجه گیری

برای این پروژه یک دیتاست نسبتا بزرگ با تعداد داده‌های زیاد و ۳۸ ویژگی وجود داشت که پس از تبدیل تمامی ویژگی‌ها به مقادیر عددی و نرمال‌سازی مقادیر موجود برای یکسان کردن دامنه و میانگین ویژگی‌ها تلاش کردیم تا با تکنیک‌های متفاوت اعم از SFS و Correlation اقدام به کاهش ابعاد کنیم. پس از آن مرحله با تعیین مقادیر مناسب برای پارامترهای متفاوت ۷ مدل متفاوت را آموزش دادیم و نتایج مربوط به هر کدام را گزارش کردیم. در نهایت مشاهده شده که مدل Multi layer perceptron با اندازه لایه نهان ۳۲ و ۶۴ بهترین مقادیر ممکن را ارائه می‌کنند که با توجه به تعداد ویژگی‌های شباهت این دو مدل تا حد زیادی قابل توجیح است. در نتیجه بهترین و بهینه‌ترین حالت ممکن برای ایجاد مدل پیش بینی خسارت ساختمان‌ها استفاده از مدل با معماری Multi layer perceptron تک لایه با اندازه لایه نهان ۳۲ و مقدار نرخ یادگیری ۰.۰۰۱ می‌باشد. اگرچه مقادیر به دست آمده به تنهایی قابل اتکا نیستند و با انجام عمل K-fold cross validation و بررسی هر بخش در یک مرحله به عنوان داده آموزش می‌توانستیم اتکا پذیری این روش‌ها را افزایش دهیم اما به دلیل اتلاف وقت بسیار زیاد این روش‌ها به خصوص روی تکنیک‌های کندی مثل Svm از این کار صرف نظر کردیم.