

Inteligencia Artificial

Cargar json

- Aquí se cargo el json con los datos de entrenamiento y se obtuvo un score de 0.8858570849655545 utilizando el método de árbol de decisión y se le puso un tamaño de testeo de 30%

```
data_features = rental_listing.drop(['interest_level'], axis=1).values
data_labels = rental_listing['interest_level'].values

X_train, X_test, Y_train, Y_test = train_test_split(data_features, data_labels, test_size=0.3)

model = DecisionTreeClassifier()
model.fit(X_train, Y_train)

score = model.score(X_test, Y_test)

print(score)
```

Datos vacios

- Aqui se buscaron datos vacios en el datasheet y no se encontraron, para esto se urilizo el un método de pd para encontrar la cantidad de datos vacios de cada clase y se saca el porcentaje se obtuvo 0 en todas. Despues de esto se saco que tan únicos eran los datos para ver la funcionalidad de las clases.

```
print(pd.DataFrame({'percent_missing': rental_listing.isnull().sum() * 100 / len(rental_listing)}))

print(pd.DataFrame({'percent_unique': rental_listing.apply(lambda x: x.unique().size/x.size*100)}))
```

Remove outliers

- Aquí se buscaron outliers con el apoyo de boxplot y scatter plot, después de esto se removieron simplemente con la opción de pandas, se guardo esto en un nuevo data_frame específico sin outliers. Score = 0.8758544087491456

```
# Delete outliers
rental_listing_no_out = rental_listing
rental_listing_no_out = rental_listing_no_out[(rental_listing_no_out["latitude"] > 40.50) & (rental_listing_no_out["latitude"] < 40.95)]
rental_listing_no_out = rental_listing_no_out[(rental_listing_no_out["longitude"] > -74.25) & (rental_listing_no_out["longitude"] < -73.65)]
rental_listing_no_out = rental_listing_no_out[(rental_listing_no_out['bathrooms'] == 1)]
rental_listing_no_out = rental_listing_no_out[(rental_listing_no_out['bedrooms'] <=3)]
plotBox(rental_listing_no_out['bathrooms'], 'bathrooms without outliers')
plotBox(rental_listing_no_out['bedrooms'], 'bedrooms without outliers')
```

- Al remover los outliers me di cuenta que el algoritmo mejoró muchísimo, comencé a jugar con las gráficas para que quedaran tanto la plotbox como la scatterplot sin outliers

Selección de datos

- Buscando los datos del sklearn encontré un método fácil de hacer el feature selection, con el árbol de decisión

```
X_train, X_test, Y_train, Y_test = train_test_split(data_features, data_labels, test_size=0.25)

model = DecisionTreeClassifier()
model.fit(X_train, Y_train)

print(dict(zip(rental_listing_no_out.columns, model.feature_importances_)))
```

```
{
    "bathrooms": 0.0,
    "bedrooms": 0.04537160587416816,
    "building_id": 0.026155609061964274,
    "created": 0.0,
    "description": 0.0025355711538119284,
    "display_address": 0.0018767862160205394,
    "latitude": 0.19194831945520255,
    "listing_id": 0.32564746947938333,
    "longitude": 0.18937543722344619,
    "manager_id": 0.0,
    "price": 0.21708920153600292,
    "street_address": 0.0
}
```

- Despues buscando mas opciones descubri como utilizar selectkbest que me parecio mas adecuada para el problema score = 0.894565960355434

```
test = SelectKBest(score_func=f_classif,k=5)
fit = test.fit(data_features, data_labels)
print('')
print(json.dumps(dict(zip(rental_listing_no_out.columns, fit.scores_)), indent=2, sort_keys=True))
features = fit.transform(data_features)
print('')
print(features[0:10])
```

```
{
    "bathrooms": NaN,
    "bedrooms": 56.67807302320977,
    "building_id": 422.94668171668457,
    "created": 10.54429514189353,
    "description": 48.98305930559811,
    "display_address": 7.818869906375938,
    "latitude": 11.039080488156864,
    "listing_id": 8.78594614853764,
    "longitude": 295.9417030825513,
    "manager_id": 1.0437421075507993,
    "price": 3.4280433045050223,
    "street_address": 0.48559690170251213
}
```

- Para el feature selection comencé con un problema, debido a que me había faltado quitar las clases que tenían valores únicos, por lo que me estaba mostrando scores altísimos.
- Después de borrarlos, me mostro scores mas normales por lo que puse que agarrara solo las mejores 5 clases

Min max scaler

- Se utilizo el atributo `fit_transform` para normalizar los datos, dándole un rango de 0,1 y nos dio un score de 0.8911483253588517
- Tambien se borraron columnas con valores únicos y se le dio un `max_depth` de 5 al árbol de decisión haciendo que agarre menos características mejorando el score 0.9163533834586466 con el `maxdepth=7` 0.9174641148325359

```
'''Scaler'''
# MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
# print(features)
features = scaler.fit_transform(features)
# print('')
# print(features)
```

```
model = DecisionTreeClassifier(max_depth=5)
model.fit(X_train, Y_train)

# print(json.dumps(dict(zip(rental_listing_no_outliers, model.predict(X_test)))))

score = model.score(X_test, Y_test)

print(score)
```

Metodos ensamble

- Se agragaron métodos de redes neuronales MLP, adaBoost, Bagging y random forest obteniendo estos reultados, jugando un poco con las opciones de cada uno para mejorar el score

```
names = ['Decision Tree Classifier', 'MLP Classifier',
         'Random Forest Classifier', 'AdaBoost',
         'Bagging Classifier']

classifiers = [
    DecisionTreeClassifier(max_depth=5),
    MLPClassifier(alpha=1),
    RandomForestClassifier(max_depth=5, max_features=1, n_estimators=10),
    AdaBoostClassifier(n_estimators=10),
    BaggingClassifier(max_features=1,n_estimators=10)
]

for name, clf in zip(names, classifiers):
    clf.fit(X_train,Y_train)
    score = clf.score(X_test,Y_test)
    y_pred = clf.predict(X_test)
    print('{}: {}'.format(name, score))
```

```
Decision Tree Classifier: 0.9158407382091592
MLP Classifier: 0.9129357484620643
Random Forest Classifier: 0.9143028024606972
AdaBoost: 0.9137901572112098
Bagging Classifier: 0.9126794258373205
```