CS188_Project3

March 18, 2021

[305]:
```
from google.colab import drive
drive.mount('/content/drive')

%cd /content/drive/Shareddrives/188_project/
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
/content/drive/Shareddrives/188_project
```

[306]:
```
import os
import pandas as pd
import datetime
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
```

[307]:
```
# Updated to newest dataset
TRAIN_DATASET_PATH = "training_dataset_V3.csv"
data = pd.read_csv(TRAIN_DATASET_PATH)
data.head()
```

[307]:
```
   Unnamed: 0          dt  … brokerage_loads  total_loads
0           0  2019-12-16  …              45          483
1           1  2021-01-15  …               1           75
2           2  2019-12-26  …               2          182
3           3  2021-02-10  …               0           62
4           4  2017-07-24  …             314          371

[5 rows x 31 columns]
```

[308]:
```
# aggregate columns based on driver ID and keep only most recent entry
data = data.groupby('id_driver').apply(lambda x: x[x['dt'] == x['dt'].max()])
data
```

[308]:
```
                  Unnamed: 0          dt  … brokerage_loads  total_loads
id_driver                                 …
20        79771        79771  2016-06-20  …              42           42
26        20681        20681  2015-10-29  …               1            1
```

```
27         44537        44537  2015-12-09  …              11            11
30         842            842  2018-12-05  …               4             4
31         31360        31360  2016-04-01  …              15            15
…                         …          …   …            …          …
38039      3435          3435  2021-02-16  …               0             1
38060      34991        34991  2021-02-13  …               0             3
38065      12596        12596  2021-02-17  …               0             7
38096      6347          6347  2021-02-12  …               0             1
38125      4096          4096  2021-02-16  …               0             1

[5294 rows x 31 columns]
```

0.1 Part 1

[309]:
```python
# we convert the column 'most_recent_load_date' into datetime objects in order to calculate the 75th percentile
most_recent_load_date_datetime = [datetime.datetime.strptime(x, '%Y-%m-%d') for x in data['most_recent_load_date']]
data['most_recent_load_date'] = most_recent_load_date_datetime
```

[310]:
```python
# generate labels based on 75th percentile of 'total_loads' and 'most_recent_load_date'

labels = []

cutoff_date = data['most_recent_load_date'].quantile(0.75)

cutoff_loads = data['total_loads'].quantile(0.75)

for index, row in data.iterrows():
    if(row['total_loads'] >= cutoff_loads and row['most_recent_load_date'] >= cutoff_date):
        labels.append(1)
    else:
        labels.append(0)

data['label'] = labels
```

[311]:
```python
data['label'].value_counts()
```

[311]:
```
0    4587
1     707
Name: label, dtype: int64
```

0.2 Part 2

[312]:
```
data = data.drop(['total_loads', 'most_recent_load_date'], axis=1)

TEST_DATASET_PATH = "score_V3.csv"
testing_data = pd.read_csv(TEST_DATASET_PATH)
```

[313]:
```
data = data.append(testing_data)
```

0.3 Part 3

[314]:
```
data.describe(include='all')
```

[314]:
```
          Unnamed: 0          dt  ...  brokerage_loads        label
count    6294.000000        6294  ...      6294.000000  5294.000000
unique           NaN        1492  ...              NaN          NaN
top              NaN  2021-02-17  ...              NaN          NaN
freq             NaN         162  ...              NaN          NaN
mean    48665.792501         NaN  ...        39.102955     0.133547
std     26905.136188         NaN  ...       180.311313     0.340198
min        10.000000         NaN  ...         0.000000     0.000000
25%     25233.250000         NaN  ...         0.000000     0.000000
50%     49730.000000         NaN  ...         2.000000     0.000000
75%     74446.500000         NaN  ...        11.000000     0.000000
max     84413.000000         NaN  ...      4266.000000     1.000000

[11 rows x 30 columns]
```

[315]:
```
data.info()
```

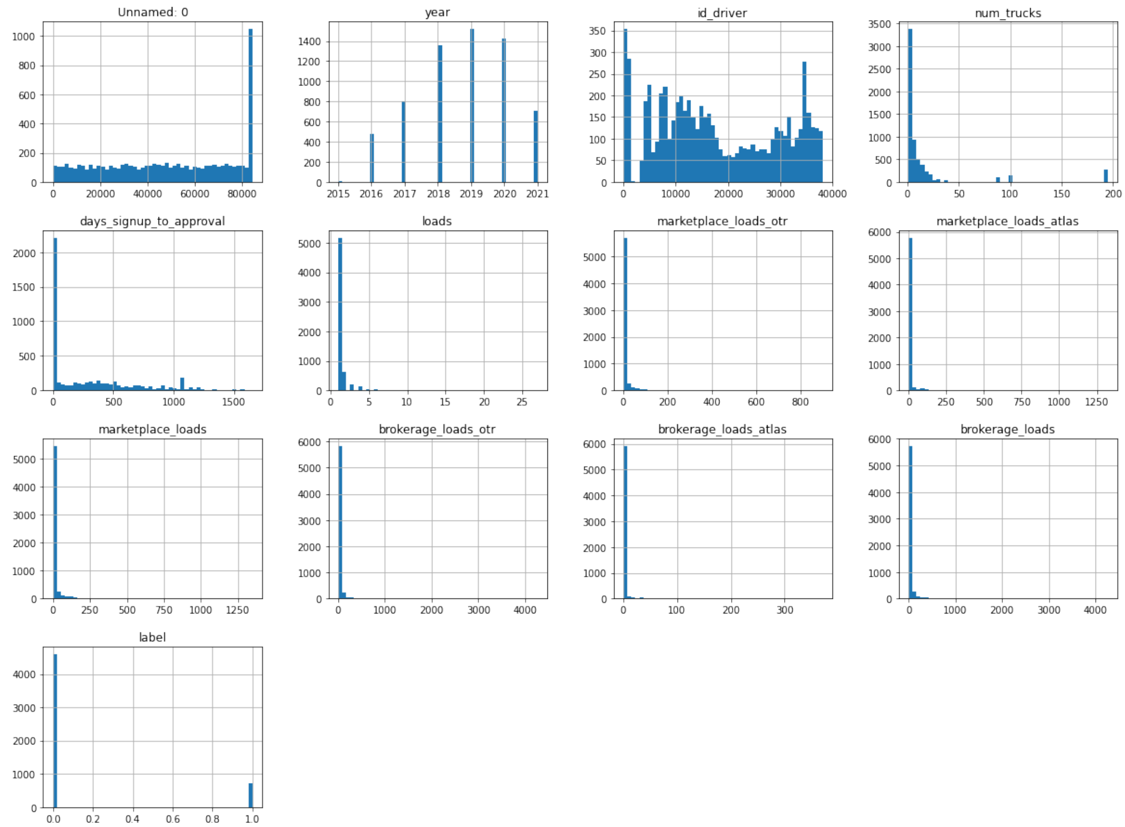
```
<class 'pandas.core.frame.DataFrame'>
Index: 6294 entries, (20, 79771) to 999
Data columns (total 30 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Unnamed: 0                 6294 non-null   int64
 1   dt                         6294 non-null   object
 2   weekday                    6294 non-null   object
 3   year                       6294 non-null   int64
 4   id_driver                  6294 non-null   int64
 5   id_carrier_number          6294 non-null   object
 6   dim_carrier_type           6294 non-null   object
 7   dim_carrier_company_name   6287 non-null   object
 8   home_base_city             6282 non-null   object
 9   home_base_state            6282 non-null   object
 10  carrier_trucks             6294 non-null   object
 11  num_trucks                 6252 non-null   float64
 12  interested_in_drayage      6294 non-null   object
```

```
 13  port_qualified            6294 non-null   object
 14  signup_source             6294 non-null   object
 15  ts_signup                 6294 non-null   object
 16  ts_first_approved         4816 non-null   object
 17  days_signup_to_approval   4816 non-null   float64
 18  driver_with_twic          6294 non-null   object
 19  dim_preferred_lanes       233 non-null    object
 20  first_load_date           6294 non-null   object
 21  load_day                  6294 non-null   object
 22  loads                     6294 non-null   int64
 23  marketplace_loads_otr     6294 non-null   int64
 24  marketplace_loads_atlas   6294 non-null   int64
 25  marketplace_loads         6294 non-null   int64
 26  brokerage_loads_otr       6294 non-null   int64
 27  brokerage_loads_atlas     6294 non-null   int64
 28  brokerage_loads           6294 non-null   int64
 29  label                     5294 non-null   float64
dtypes: float64(3), int64(10), object(17)
memory usage: 1.5+ MB
```

0.3.1 Findings:

About 25% of the data is missing in features ts_first_approved and days_signup_to_approval, and there is very little data available for the dim_preferred_lanes feature.

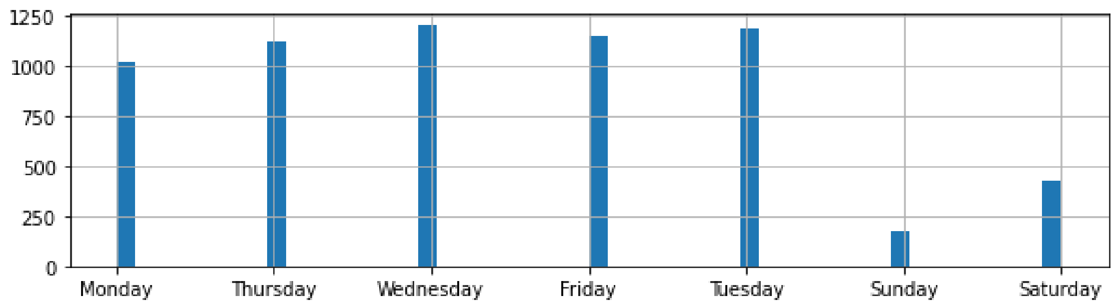
[316]:
```
#int/float valued features
data.hist(bins=50, figsize=(20,15))
plt.show()
```



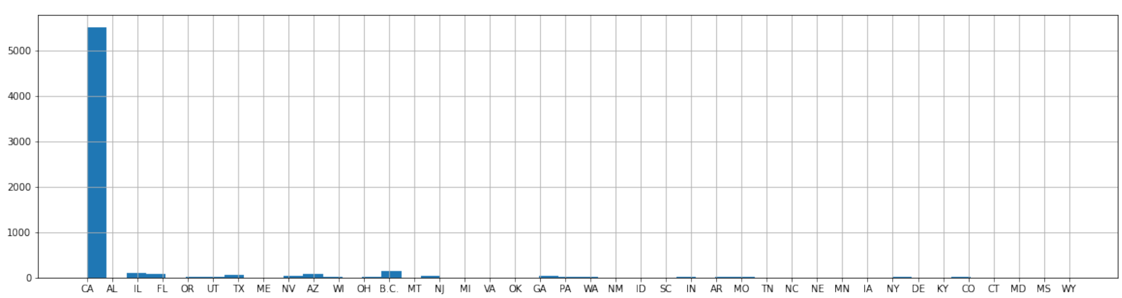

```
[317]: #some other plottable categorical features
data['dim_carrier_type'].hist(bins=10, figsize=(5,2.5))
plt.show()
```


```
[318]: data['port_qualified'].hist(bins=10, figsize=(5,2.5))
plt.show()
```

[319]:
```
data['weekday'].hist(bins=50, figsize=(10,2.5))
plt.show()
```



[320]:
```
data['home_base_state'].hist(bins=50, figsize=(20,5))
plt.show()
```



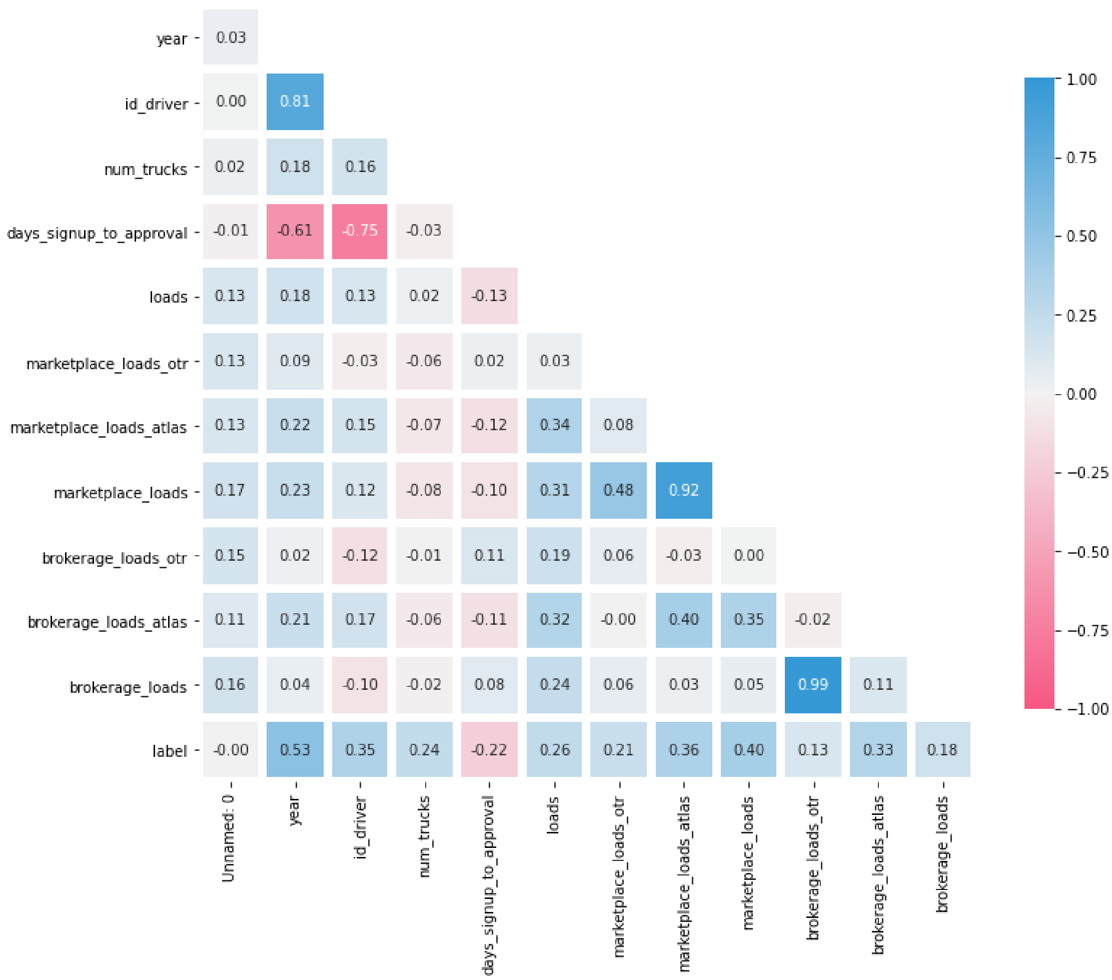
0.3.2 Findings:

We note that most real valued data are highly concentrated in smaller-valued regions with long tails to the right. Year is the only feature that seems more normally distributed. For categorical features, we note that operations are mostly done on work days, and are mostly based in California. There are a lot more fleets than owner operators, and only a minority are port qualified.

[321]:
```python
corr_matrix = data.corr()
corr_matrix['label'].sort_values(ascending=False)
```

[321]:
```
label                       1.000000
year                        0.534059
marketplace_loads           0.400645
marketplace_loads_atlas     0.361230
id_driver                   0.345887
brokerage_loads_atlas       0.327677
loads                       0.257802
num_trucks                  0.244607
marketplace_loads_otr       0.212335
brokerage_loads             0.182466
brokerage_loads_otr         0.126494
Unnamed: 0                 -0.003198
days_signup_to_approval    -0.221869
Name: label, dtype: float64
```

[322]:
```python
#credit: https://towardsdatascience.com/
 ↪heatmap-basics-with-pythons-seaborn-fb92ea280a6c
fig, ax = plt.subplots(figsize=(12, 10))
# mask
mask = np.triu(np.ones_like(corr_matrix, dtype=np.bool))
# adjust mask and df
mask = mask[1:, :-1]
corr = corr_matrix.iloc[1:,:-1].copy()
# color map
cmap = sb.diverging_palette(0, 240, 90, 60, as_cmap=True)
# plot heatmap
sb.heatmap(corr, mask=mask, annot=True, fmt=".2f",
linewidths=5, cmap=cmap, vmin=-1, vmax=1, cbar_kws={"shrink": .8}, square=True)
plt.show()
```



0.3.3 Findings:

- Label is most positively correlated to year, followed by marketplace_loads and marketplace_loads_atlas. In general, it has a positive correlation with all features except days_signup_to_approval.
- marketplace_loads is highly correlated with marketplace_loads_atlas; same thing between brokerage_loads and brokerage_loads_otr.
- id_driver is noticeably (positively) correlated with year; days_signup_to_approval is noticeably (negatively) correlated with year and id_driver.

0.4 Part 4

Ideas for feature extraction: - many null values in dim_preferred_lanes: change it to a binary variable indicating whether a perferred lane was specified (differentiate between drivers that did and didn't specify) - weekday: workday vs. weekend

Imputation: - home_base_city - home_base_state - num_trucks - dim_preferred_lanes

Features not needed: - ts_signup, ts_first_approved: the period between signup and app - dim_carrier_company_name - dt, load_day ==> these are the same as most_recent _load_date

[323]:
```
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder

from sklearn.base import BaseEstimator, TransformerMixin

#drop labels for training set features
data_unlabeled = data.drop("label", axis=1)
data_labels = data["label"].copy()[:5294]

# Drop Features:
data_unlabeled = data_unlabeled.drop(columns=['dim_carrier_company_name',␣
↪'ts_signup', 'ts_first_approved', 'dt', 'load_day'])

# Convert first_load_date to numerical data
data_unlabeled['first_load_date'] = [datetime.datetime.strptime(x, '%Y-%m-%d')␣
↪for x in data_unlabeled['first_load_date']]
diff = [abs(x - datetime.datetime.strptime('2021-3-17','%Y-%m-%d')) for x in␣
↪data_unlabeled['first_load_date']]
data_unlabeled['first_load_date'] = [x.days for x in diff]

# Separate numerical vs. binary vs. multilabel features
binary_features = ['weekday', 'dim_carrier_type', 'carrier_trucks',␣
↪'interested_in_drayage', 'port_qualified', 'signup_source',␣
↪'driver_with_twic', 'dim_preferred_lanes']
categorical_features = ['id_carrier_number', 'home_base_city',␣
↪'home_base_state']
nonnumeric_features = binary_features + categorical_features
data_num = data_unlabeled.drop(columns=nonnumeric_features, axis=1)

marketplace_loads_ix, brokerage_loads_ix = 8, 11
class AugmentFeaturesNum(BaseEstimator, TransformerMixin):
  def __init__(self):
    return None
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    marketplace_vs_brokerage = X[:, marketplace_loads_ix] / (X[:,␣
↪marketplace_loads_ix] + X[:, brokerage_loads_ix] + 1) #deal with division by␣
↪0 error
```

```
    return np.c_[X, marketplace_vs_brokerage]

weekday_ix, carrier_trucks_ix, interested_in_drayage_ix, port_qualified_ix,␣
↪driver_with_twic_ix, dim_preferred_lanes_ix = 0,2,3,4,6,7
class AugmentFeaturesBin(BaseEstimator, TransformerMixin):
  def __init__(self):
    return None
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    X = X.to_numpy()
    weekday = [0 if x=='Saturday' or x=='Sunday' else 1 for x in X[:
↪,weekday_ix]]
    good_port_person = [1 if x[interested_in_drayage_ix]=='yes' and␣
↪x[port_qualified_ix]=='yes' and x[driver_with_twic_ix]=='yes' else 0 for x␣
↪in X]
    specified_preferred_lanes = [0 if x=="missing_value" else 1 for x in X[:
↪,dim_preferred_lanes_ix]]
    poweronly =  [1 if ('poweronly' in x) else 0 for x in X[:
↪,carrier_trucks_ix]]
    dryvan =  [1 if ('dryvan' in x) else 0 for x in X[:,carrier_trucks_ix]]
    reefer =  [1 if ('reefer' in x) else 0 for x in X[:,carrier_trucks_ix]]
    boxtruck =  [1 if ('boxtruck' in x) else 0 for x in X[:,carrier_trucks_ix]]
    flatbed =  [1 if ('flatbed' in x) else 0 for x in X[:,carrier_trucks_ix]]
    truck_unknown =  [1 if (x=='unknown') else 0 for x in X[:
↪,carrier_trucks_ix]]
    X = np.delete(X, weekday_ix, 1)              #remove original weekday␣
↪objects after augmenting
    X = np.delete(X, carrier_trucks_ix-1, 1)     #remove original␣
↪carrier_trucks objects after augmenting
    X = np.delete(X, dim_preferred_lanes_ix-2,1) #remove original␣
↪dim_preferred_lanes objects after augmenting
    return np.c_[X, weekday, good_port_person, specified_preferred_lanes,␣
↪poweronly, dryvan, reefer, boxtruck, flatbed, truck_unknown]

# numerical features -> impute (median) and standard scaler mean 0 unit variance
numerical_features = list(data_num)
num_pipeline = Pipeline([
                         ('imputer', SimpleImputer(strategy="median")),
                         ('attribs_adder', AugmentFeaturesNum()),
                         ('std_scaler', StandardScaler())
])

cat_pipeline = Pipeline([
                         ('imputer', SimpleImputer(strategy="constant",␣
↪fill_value="missing_value")),
```

```
                          ('oneHotEncoder', OneHotEncoder())
])

bin_pipeline = Pipeline([
                          ('attribs_adder', AugmentFeaturesBin()),
                          ('ordinalEncoder', OrdinalEncoder())
])

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", cat_pipeline, categorical_features),
        ("bin", bin_pipeline, binary_features)
])

data_prepared = full_pipeline.fit_transform(data_unlabeled)

train_data = data_prepared[:5294]
test_data = data_prepared[5294:]

train_data.shape
test_data.shape
```

[323]: (1000, 3174)

[324]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter

print('Before oversampling: ', Counter(data_labels))

SMOTE = SMOTE()
train_data, data_labels = SMOTE.fit_resample(train_data, data_labels)

print('After oversampling: ', Counter(data_labels))
```

```
Before oversampling:  Counter({0.0: 4587, 1.0: 707})

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)

After oversampling:  Counter({0.0: 4587, 1.0: 4587})
```

0.5 Part 5

[325]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```python
from sklearn.model_selection import cross_validate
from statistics import mean


# create train/test/ split dataset
train, test, target, target_test = train_test_split(train_data, data_labels,␣
↪test_size=0.2, random_state=0)

# fit basic linear regression
lin_reg = LinearRegression()
lin_reg.fit(train, target)


# Scoring (on single train/test split)
preds = lin_reg.predict(test)
mse = mean_squared_error(target_test, preds)
print("Test MSE:", mse)
r2 = r2_score(target_test, preds)
print("R^2: ", r2)

# Feature Importance -> print the first 10 most significant predictors
importance = lin_reg.coef_
feature_importance = []
for i,v in enumerate(importance):
  feature_importance.append((i,abs(v)))   # want magnitude of feature importance
sorted_importance = sorted(feature_importance, reverse=True, key=lambda x: x[1])

print("\nTop 10 Most Significant Predictors (by coefficients):")
for i in range(10):
  print('Feature %d: \tCoefficient: %.4f' % (sorted_importance[i][0],␣
↪sorted_importance[i][1]))


# Cross Validation
scoring = ['r2']
scores = cross_validate(lin_reg, train_data, data_labels, cv=10,␣
↪scoring=scoring)
print('\nR^2 After Cross Validation:')
# print(sorted(sklearn.metrics.SCORERS.keys()))
# print(sorted(scores.keys()))
# print(scores['test_r2'])
print("R^2: ", mean(scores['test_r2']))
```

```
Test MSE: 0.04022398635092772
R^2:  0.8390868031016369

Top 10 Most Significant Predictors (by coefficients):
```

```
Feature 1760:   Coefficient: 1.6740
Feature 1402:   Coefficient: 1.0018
Feature 2233:   Coefficient: 0.9080
Feature 1981:   Coefficient: 0.8953
Feature 2047:   Coefficient: 0.8932
Feature 1172:   Coefficient: 0.8829
Feature 2430:   Coefficient: 0.8651
Feature 2513:   Coefficient: 0.8631
Feature 1913:   Coefficient: 0.8481
Feature 1320:   Coefficient: 0.8223

R^2 After Cross Validation:
R^2:  -0.1567204973687807
```

[326]:

```python
# Use F-stat (f_regression) to evaluate feature importance

from sklearn.feature_selection import f_regression

# f-regression
f_vals, p_vals = f_regression(train_data, data_labels)

# get top 10 most important predictors by F-stat val
feature_importance = []
for i,v in enumerate(f_vals):
  feature_importance.append((i,v))   # f-stat
sorted_importance = sorted(feature_importance, reverse=True, key=lambda x: x[1])

print("\nTop 10 Most Significant Predictors (by F-stat):")
for i in range(10):
  print('Feature %d: \tCoefficient: %.4f' % (sorted_importance[i][0],␣
 ↪sorted_importance[i][1]))


# get top 10 most important predictors by p-val
feature_importance = []
for i,v in enumerate(p_vals):
  feature_importance.append((i,v))   # f-stat
sorted_importance = sorted(feature_importance, reverse=True, key=lambda x: x[1])

print("\nTop 10 Most Significant Predictors (by p-val):")
for i in range(10):
  print('Feature %d: \tCoefficient: %.4f' % (sorted_importance[i][0],␣
 ↪sorted_importance[i][1]))
```

```
Top 10 Most Significant Predictors (by F-stat):
Feature 1:      Coefficient: 12453.2503
Feature 5:      Coefficient: 3407.4411
```

```
Feature 2:      Coefficient: 3358.9261
Feature 13:     Coefficient: 2672.1354
Feature 9:      Coefficient: 1026.1165
Feature 8:      Coefficient: 799.8267
Feature 1007:   Coefficient: 795.4265
Feature 6:      Coefficient: 788.6501
Feature 4:      Coefficient: 733.2667
Feature 11:     Coefficient: 655.7454

Top 10 Most Significant Predictors (by p-val):
Feature 2783:   Coefficient: 0.9346
Feature 1291:   Coefficient: 0.8449
Feature 843:    Coefficient: 0.8382
Feature 0:      Coefficient: 0.6710
Feature 1002:   Coefficient: 0.6535
Feature 827:    Coefficient: 0.5689
Feature 217:    Coefficient: 0.5380
Feature 1225:   Coefficient: 0.5345
Feature 1123:   Coefficient: 0.5237
Feature 884:    Coefficient: 0.5219
```

```
/usr/local/lib/python3.7/dist-
packages/sklearn/feature_selection/_univariate_selection.py:299: RuntimeWarning:
invalid value encountered in true_divide
  corr /= X_norms
```

[326]:

```
```

0.6 Part 6

[327]:

```
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD

# PCA to make 100 most significant predictors(linear combinations of og
 ↪predictors)
# pca = PCA(n_components=100)
# pca.fit(data_prepared)
# Cannot do PCA since 'PCA does not support sparse input.' error -> too many 0s

# PCA to make 10 most significant predictors(linear combinations of og
 ↪predictors)
svd = TruncatedSVD(n_components=10, n_iter=10)
svd.fit(data_prepared)

svd_train_data = svd.transform(train_data)
svd_test_data = svd.transform(test_data)
```

```
# pd.DataFrame(data=data_prepared)
# data_prepared.describe(include='all')
```

0.7 Part 7

[328]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score

train, test, target, target_test = train_test_split(svd_train_data,␣
↪data_labels, test_size=0.2, random_state=0)

ensemble_clf = BaggingClassifier(base_estimator=SVC(), n_estimators=10,␣
↪random_state=0)
ensemble_clf.fit(train, target)
ensemble_preds = ensemble_clf.predict(test)
acc_score = accuracy_score(target_test, ensemble_preds)
f1_score = f1_score(target_test, ensemble_preds)

print('Test accuracy: ', acc_score)
print('Test F1 Score: ', f1_score)
```

```
Test accuracy:  0.9542234332425068
Test F1 Score:  0.9548872180451128
```

0.8 Part 8

[329]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score

param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (200,), (50,50), (100,50,),␣
↪(100,100,), (200,100,50,)],
    'max_iter': [200, 400, 500, 600]
}

#nn = MLPClassifier()
#clf = GridSearchCV(estimator=nn, scoring='f1', param_grid=param_grid)
#clf.fit(train, target)

#print(clf.best_score_)
#print(clf.best_params_)

# activation = ReLU
# solver = Adam
nn_clf = MLPClassifier(hidden_layer_sizes=(100,100,100,50), max_iter=400)
```

```
nn_clf.fit(train, target)
nn_preds = nn_clf.predict(test)

nn_acc_score = accuracy_score(target_test, nn_preds)
nn_f1_score = f1_score(target_test, nn_preds)

print('Test accuracy: ', nn_acc_score)
print('Test F1 Score: ', nn_f1_score)
```

```
Test accuracy:  0.9814713896457765
Test F1 Score:  0.9819915254237288
```

0.9 Part 9

[330]:
```
from sklearn.model_selection import cross_validate

# Cross Validation
scoring = ['accuracy', 'f1', 'roc_auc']
ensemble_scores = cross_validate(ensemble_clf, svd_train_data, data_labels,␣
↪cv=10, scoring=scoring)

nn_clf = MLPClassifier(hidden_layer_sizes=(100,100,100,50,), max_iter=400)

nn_scores = cross_validate(nn_clf, svd_train_data, data_labels, cv=10,␣
↪scoring=scoring)

print('\nEnsemble Method After Cross Validation:')
print("Classification Accuracy: ", mean(ensemble_scores['test_accuracy']))
print("F1 Score: ", mean(ensemble_scores['test_f1']))
print("ROC Curve AUC Value: ", mean(ensemble_scores['test_roc_auc']))

print('\nNeural Network After Cross Validation:')
print("Classification Accuracy: ", mean(nn_scores['test_accuracy']))
print("F1 Score: ", mean(nn_scores['test_f1']))
print("ROC Curve AUC Value: ", mean(nn_scores['test_roc_auc']))
```

```

Ensemble Method After Cross Validation:
Classification Accuracy:  0.9321843750222736
F1 Score:  0.9368000265521321
ROC Curve AUC Value:  0.9754391151058092

Neural Network After Cross Validation:
Classification Accuracy:  0.9723056143577024
F1 Score:  0.9738818738149024
ROC Curve AUC Value:  0.9856434454217079
```

0.10 Part 10

[331]:
```
nn_clf = MLPClassifier(hidden_layer_sizes=(100,100,100,50), max_iter=400)
nn_clf.fit(svd_train_data, data_labels)
scoring_preds = nn_clf.predict(svd_test_data)

score_array = np.c_[testing_data['Unnamed: 0'], scoring_preds]
score_df = pd.DataFrame(data=score_array, columns=['Id', 'Predicted'])
score_df = score_df.astype(int)

score_df.to_csv('scoring.csv', index=False)
```

[332]:
```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=200, max_depth=7, max_features=0.8)
rf_clf.fit(train, target)
rf_preds = rf_clf.predict(test)

rf_acc_score = accuracy_score(target_test, rf_preds)
rf_f1_score = f1_score(target_test, rf_preds)

print('Test accuracy: ', rf_acc_score)
print('Test F1 Score: ', rf_f1_score)

#rf_clf.fit(svd_train_data, data_labels)
#scoring_preds = rf_clf.predict(svd_test_data)

#ids = testing_data['Unnamed: 0']

#score_array = np.c_[ids, scoring_preds]
#score_df = pd.DataFrame(data=score_array, columns=['Id', 'Predicted'])
#score_df = score_df.astype(int)

#score_df.to_csv('scoring.csv', index=False)
```

```
Test accuracy:  0.9378746594005449
Test F1 Score:  0.9396825396825397
```

[333]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

base_est = DecisionTreeClassifier(max_depth=2)

ada_clf = AdaBoostClassifier(base_estimator=base_est, n_estimators=100)
ada_clf.fit(train, target)
ada_preds = ada_clf.predict(test)

ada_acc_score = accuracy_score(target_test, ada_preds)
```

```python
ada_f1_score = f1_score(target_test, ada_preds)

print('Test accuracy: ', ada_acc_score)
print('Test F1 Score: ', ada_f1_score)
```

```
Test accuracy:  0.9613079019073569
Test F1 Score:  0.9620117710005349
```