# ML MODEL TO PREDICT CREDIT LIMIT



**Boliang Liu**

# DATA

## Credit card history records

| Features |
| --- |
| Age |
| Gender |
| Education_Level |
| Income_Category |
| Total_Amount_Checking |
| ......... |

Model →

| Target |
| --- |
| Credit_Limit |

# PROCESS DATA

1. Choose meaningful features
2. Check continuous columns and categorical columns
3. Standardize continuous columns and impute them
4. OneHotEncode categorical columns and impute them
5. Construct the column transformer

# CODE

```
target = credit['Credit_Limit']
target = target.values.ravel()
```

```
categorical_columns = (X_train.dtypes == object)
continuous_columns  = (X_train.dtypes != object)
```

```
con_pipe = Pipeline([('scalar', MaxAbsScaler()),
                     ('imputer', SimpleImputer(missing_values=np.nan, strategy='median', add_indicator=True))
                     ])

cat_pipe = Pipeline([('ohe', OneHotEncoder(handle_unknown='ignore')),
                     ('imputer', SimpleImputer(strategy='most_frequent', add_indicator=True))])

preprocessing = ColumnTransformer([('categorical', cat_pipe,  categorical_columns),
                                   ('continuous',  con_pipe,  continuous_columns),
                                   ])
```

# TRAIN, TEST, PIPELINE

1. Split dataset into train set and test
2. Use pca in the pipeline to focus on important features and improve generality
3. Construct pipeline

# CODE

```
X_train, X_test, y_train, y_test = train_test_split(data,
                                                    target,
                                                    test_size=0.2)
```

```
pipe = Pipeline([('preprocessing', preprocessing),
                 ('pca', PCA(n_components=15)),
                 ('clf',DummyEstimator())
                ])
```

# SEARCH BEST MODEL AND HYPERPARAMETERS

1. Construct search space by using 7 different models
2. Construct search algorithm
3. Search 5 times for the best model and hyper parameters

# CODE

```python
search_space = [{'clf': [RandomForestRegressor()],
                 'clf__n_estimators': np.arange(100, 1000, 150), # decides how many trees
                 'clf__max_features': ['log2','sqrt'], # decides how many features in each tree
                 'clf__max_depth' : np.arange(15,25,1), # decides how deep in each tree
                 'clf__min_samples_leaf': np.arange(1,10,1), # decides hwo many samples at minimum in each leaf
                 'clf__bootstrap': [True, False] # decides if using bootstrap technique
                },

                {'clf': [SVC()],
                 'clf__C': np.logspace(0.1, 1000, 5), # decides the best C value
                 'clf__gamma': np.logspace(0.0001,1,5), # decides the best gamma
                 'clf__kernel':['rbf','poly'], # decides kernal type
                 'clf__class_weight': ['balanced',None] # decides the type of weighted class
                },

                {'clf': [RidgeCV()],
                 'clf__normalize': [False, True], # decides if doing normalization
                 'clf__alpha_per_target' : [False, True] # decides if using alpha in per target
                },

                {'clf': [LassoCV()],
                 'clf__eps': np.arange(0.0005, 0.01, 0.0005), # decides the best epsilon
                 'clf__normalize': [False, True], # decides if doing normalization
                 'clf__max_iter': np.arange(1000,5000,1000), # decides the maximum times of iteration
                 'clf__n_alphas': np.arange(100,500,100) # decides the best n_alphas
                },

                {'clf': [BayesianRidge()],
                 'clf__normalize': [False, True], # decides if doing normalization
                 'clf__n_iter': np.arange(100, 1000, 100) # decides the times of iteration
                },

                {
                 'clf': [HuberRegressor()],
                 'clf__alpha': np.arange(0.0001, 0.001, 0.0001), # decides the best alpha
                 'clf__max_iter': np.arange(100,1000,100), # decides the times of iteration
                 'clf__epsilon': np.arange(1,2,0.1) # decides the best epsilon
                },

                {
                 'clf': [ExtraTreesRegressor()],
                 'clf__max_features': ['log2','sqrt'], # decides how many features in each tree
                 'clf__max_depth' : np.arange(15,25,1), # decides how deep in each tree
                 'clf__n_estimators': np.arange(100, 1000, 150), # decides how many trees
                 'clf__min_samples_leaf': np.arange(1,10,1), # decides hwo many samples at minimum in each leaf
                 'clf__bootstrap': [True, False] # decides if using bootstrap technique
                }
                 ]

clf_algos_rand = RandomizedSearchCV(estimator=pipe,
                                    param_distributions=search_space,
                                    n_iter=50,
                                    cv=5,
                                    n_jobs=-1,
                                    verbose=10,
                                    scoring='neg_root_mean_squared_error')
```

```python
for i in range(5):

    best_model = clf_algos_rand.fit(X_train, y_train)

    print(best_model.best_estimator_.get_params()['clf'], end='\n')
    print(best_model.best_score_, end='\n')
```

# BEST MODEL AND HYPER PARAMETERS

1. Best model: RandomForestRegressor
2. Best hyperparameters
3. Final pipeline

# CODE

```
params={'bootstrap': False,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 20,
 'max_features': 'log2',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 250,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```
pipe = Pipeline([('preprocessing', preprocessing),
                 ('pca', PCA(n_components=15)),
                 ('reg',RandomForestRegressor(**params))
                ])
```

# FIT MODEL AND PREDICT ON TEST SET

```
pipe.fit(X_train, y_train)
```

```
y_pred    = pipe.predict(X_test)
```

```
mse   = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r_2 = r2_score(y_test, y_pred)
```

```
print('mse = {mse}\nmae = {mae}\nr2  = {r_2}'.format(mse=mse, mae=mae, r_2=r_2))
```

```
mse = 2575826.8301345236
mae = 1044.6272158530253
r2  = 0.9702055308283112
```

# CONCLUSION

1. For this project, RandomForest performs best among RidgeCV, LassoCV, BayesianRidge, HuberRegressor, ExtraTreesRegressor, SVC
2. R2 = 0.97. Pretty good!
3. RandomizedSeachCV has randomness
4. PCA is helpful for increasing generality
5. Do grid search later if possible

# THANK YOU FOR WATCHING!