

🔗 Nanodegree Engenheiro de Machine Learning

Projeto final

Bruno Aurélio Rôzza de Moura Campos

10 de fevereiro de 2019

I. Definição

1.1 Visão geral do projeto

Quando ocorre um acidente de carro, as pessoas mantem o foco em família, amigos e outros entes queridos. A parte burocrática de entrar em contato com um agente de seguros é o último lugar que alguém vai gastar esforços. Para minimizar esforços na utilização de um serviço de sinistro será feito uma análise preditiva e assim, garantir a menor perda de tempo possível ao usuário.

1.2 Descrição do problema

O projeto será voltado a automatizar a previsão de custo e análise da gravidade de um sinistro. Como resultado, será possível garantir uma experiência mais eficiente aos clientes que necessitam. A seguradora que esta realizando o desafio, Allstate, forneceu os seguintes dados:

1. Variáveis em train.csv e test.csv:

- **id**: o id de um par de perguntas do conjunto de treinamento
- **cat1** até **cat116**: variáveis de categoria (o intervalo de valores não é fornecido, nem os nomes das colunas).
- **cont1** até **cont14**: variáveis contínuas (o intervalo de valores não é fornecido, nem os nomes das colunas).
- **loss**: o valor que a empresa tem que pagar por uma determinada reivindicação. Esta é a variável de destino. - Em test.csv, a perda não está presente, já que vamos prever isso.

2. Em train.csv:

- Número de linhas = 188318
- Número de colunas = 132

3. Em test.csv:

- Número de linhas = 125546
- Número de colunas = 131

O problema é criar um algoritmo que prevê com precisão a gravidade das reivindicações. Como entrada, será recebido diferentes variáveis que os agentes examinam para decidir o status das reivindicações. Eles podem ser contínuos ou discretos. Como a variável de destino é uma quantidade contínua (o valor a ser pago ao cliente), é essencialmente uma função de regressão. Desta forma, analisando o objetivo da seguradora neste desafio, é possível notar o algoritmo resultante deverá fazer recomendações para os usuários.

1.3 Métricas

Há várias maneiras de avaliar este problema. Como a avaliação oficial deste projeto é feita pelo Kaggle usando erro absoluto médio (Mean absolute error - MAE), o mesmo será usado para avaliação de modelos entre a predição de perda e a perda atual reivindicada, conforme consta nos datasets. Abaixo há a fórmula do MAE:

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|.$$

- y : valor True
- \hat{y} : valor predito
- n samples: número de exemplos estimados

II. Análise

2.1 Exploração dos dados

Neste projeto foi utilizado a base de dados que a seguradora Allstate forneceu na competição do Kaggle contendo diversas informações técnicas (features).

O conjunto de dados contém 2 arquivos .csv com informações necessárias para fazer uma previsão. Ao concatenar estes arquivos obtemos em torno de 318.5 MB com 132 colunas e 313864 linhas. Eles contem as seguintes features:

1. Features em train.csv e test.csv:

- **id**: o id de um par de perguntas do conjunto de treinamento
- **cat1** até **cat116**: variáveis de categoria (o intervalo de valores não é fornecido, nem os nomes das colunas).
- **cont1** até **cont14**: variáveis contínuas (o intervalo de valores não é fornecido, nem os nomes das colunas).
- **loss**: o valor que a empresa tem que pagar por uma determinada reivindicação. Esta é a variável de destino. - Em test.csv, a perda não está presente, já que vamos prever isso.

Cabe observar que em test.csv a coluna loss não esta presente pois é o que será predito.

2.2 Visualização exploratória

Nesta seção, você precisará fornecer alguma forma de visualização que sintetize ou evidencie uma característica ou atributo relevante sobre os dados. A visualização deve sustentar adequadamente os dados utilizados. Discuta por que essa visualização foi escolhida e por que é relevante. Questões para se perguntar ao escrever esta seção:

- *Você visualizou uma característica ou um atributo relevante acerca do conjunto de dados ou dados de entrada?*
- *A visualização foi completamente analisada e discutida?*
- *Se um gráfico foi fornecido, os eixos, títulos e dados foram claramente definidos?*

Ao analisar os dados fornecidos, observamos que temos os seguintes tipos de dados:

- int64(1) = id
- float64(15) = features contínuas + loss
- object(116) = features categóricas

As imagens 1 e 2 são amostras dos dados de train e test.

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	cat11	cat12	cat13	cat14	cat15
0	1	A	B	A	B	A	A	A	A	B	A	B	A	A	A	A
1	2	A	B	A	A	A	A	A	A	B	B	A	A	A	A	A
2	5	A	B	A	A	B	A	A	A	B	B	B	B	B	A	A
3	10	B	B	A	B	A	A	A	A	B	A	A	A	A	A	A
4	11	A	B	A	B	A	A	A	A	B	B	A	B	A	A	A
5	13	A	B	A	A	A	A	A	A	B	A	A	A	A	A	A
6	14	A	A	A	A	B	A	A	A	A	A	A	A	A	A	A
7	20	A	B	A	B	A	A	A	A	B	A	A	A	A	A	A
8	23	A	B	B	B	B	A	A	A	B	B	B	B	B	A	A
9	24	A	B	A	A	B	B	A	A	B	A	A	A	A	A	A
10	25	A	B	A	A	A	A	A	A	B	B	B	B	B	A	A

Imagem 1. Amostra do arquivo de train.csv

cat115	cat116	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8	cont9	cont10	cont11	cont12	cont13	cont14
Q	HG	0.321594	0.299102	0.246911	0.402922	0.281143	0.466591	0.317681	0.612229	0.34365	0.38016	0.377724	0.369858	0.704052	0.392562
L	HK	0.634734	0.620805	0.654310	0.946616	0.836443	0.482425	0.443760	0.71330	0.51890	0.60401	0.689039	0.675759	0.453468	0.208045
K	CK	0.290813	0.737068	0.711159	0.412789	0.718531	0.212308	0.325779	0.29758	0.34365	0.30529	0.245410	0.241676	0.258586	0.297232
P	DJ	0.268622	0.681761	0.592681	0.354893	0.397069	0.369930	0.342355	0.40028	0.33237	0.31480	0.348867	0.341872	0.592264	0.555955
J	HA	0.553846	0.299102	0.263570	0.696873	0.302678	0.398862	0.391833	0.23688	0.43731	0.50556	0.359572	0.352251	0.301535	0.825823
I	HY	0.594934	0.245921	0.397983	0.849584	0.643315	0.407351	0.390540	0.46477	0.46853	0.50556	0.607500	0.594646	0.250991	0.283976
Q	MD	0.964279	0.620805	0.356819	0.838840	0.281143	0.960845	0.740081	0.75964	0.98330	0.82249	0.863052	0.879347	0.888944	0.787807
H	KC	0.633389	0.047297	0.129646	0.864586	0.651246	0.451115	0.316313	0.27320	0.52100	0.50556	0.415029	0.481306	0.199940	0.450597
Q	GC	0.261841	0.245921	0.484196	0.463029	0.534484	0.343492	0.358758	0.81900	0.32128	0.36458	0.453334	0.443374	0.695650	0.295075
O	DT	0.527930	0.488789	0.397983	0.775744	0.281143	0.394921	0.287416	0.92347	0.48320	0.24766	0.359572	0.352251	0.519989	0.602666
J	HK	0.484469	0.555782	0.777587	0.229617	0.405415	0.457821	0.774678	0.33372	0.41471	0.47779	0.760322	0.747533	0.304350	0.305920
M	HX	0.636078	0.488789	0.280933	0.761209	0.696981	0.627435	0.451447	0.52450	0.64873	0.79139	0.472726	0.500382	0.689974	0.307258
N	HK	0.436958	0.620805	0.673861	0.336414	0.281143	0.644854	0.724803	0.34987	0.48320	0.67065	0.695685	0.682413	0.642600	0.838149
K	DJ	0.546670	0.681761	0.634224	0.373816	0.867056	0.364464	0.401162	0.26847	0.46226	0.50556	0.366788	0.359249	0.345247	0.725605
N	HK	0.455585	0.827585	0.692825	0.229617	0.281143	0.636189	0.793953	0.37754	0.41471	0.58796	0.705501	0.698722	0.611431	0.822254

Imagem 2. Amostra do arquivo de test.csv

2.3 Algoritmos e técnicas

A relação entre as *features* de 130 (116 + 14) com a variável *loss* é fundamental para compreensão do problema. Para isso, a solução desenvolvida utiliza a técnica de *Hybrid filtering*, para melhorar os resultados do sistema. Mais especificamente, o sistema utiliza o método de `_ Feature-combination_` para determinar a similaridade das *features*. Também uma técnica fundamental para o entendimento do problema é analisar as correlações entre as *features* para selecionar os melhores resultados. Na parte de exploração de dados foi convertido valores categóricos de alfabetos em números que podem ser mais fáceis de serem processados. Os modelos testados foram:

- **Regressão linear:** É o algoritmo estatístico no qual consiste em expressar a saída desejada na forma de função linear, onde cada instancia é relacionada com um peso. Sua principal vantagem é a simplicidade, sendo reconhecido como o algoritmo mais simples de trabalhar com dados numéricos, contudo já as suas desvantagens consistem em trabalhar apenas com dados numéricos e sua eficiência em dados não lineares é baixa.
- **XGBoost:** O XGBoost é uma biblioteca de otimização de gradiente distribuída otimizada projetada para ser altamente eficiente, flexível e portátil. Ele implementa algoritmos de aprendizado de máquina sob a estrutura Gradient Boosting. O XGBoost fornece um reforço de árvore paralela (também conhecido como GBDT, GBM) que resolve muitos problemas de ciência de dados de maneira rápida e precisa. O mesmo código é executado em ambientes distribuídos principais (Hadoop, SGE, MPI) e pode resolver problemas além de bilhões de exemplos. Este algoritmo de boosting tem regularização para evitar overfitting, garante um processamento paralelo e ainda pode ser altamente flexível, o XGBoost permite aos usuários definir personalizar objetivos de otimização e critérios de avaliação. Além disso, o gerenciando valores faltantes ('missing values') pois ele possui rotinas internas para lidar com valores faltantes. Em relação à poda (prune) de árvores: XGBoost faz splits até o `max_depth` especificado e, em seguida, começa a podar a árvore de trás pra frente e remove as divisões para além das quais não há ganho positivo.
- **Random Forest (Bagging):** Um algoritmo de classificação capaz de executar classificação em várias classes de conjunto de dados. Esse algoritmo requer pouca preparação dos dados e tem um custo logarítmico para o treinamento. Contudo, pode criar sistemas altamente complexos que não generalizam bem e assim ficarem instáveis pois pequenas variações nos dados podem resultar na geração de árvores completamente diferentes

2.4 Benchmark

Após esta preparação dos dados foi realizado testes com modelos de machine learning para verificar qual tem melhor performance usando a divisão do Kfold e, finalmente, calculado o erro médio quadrático (MAE).

III. Metodologia

3.1 Pré-processamento de dados

Foi necessário aplicar várias técnicas de processamento para preparar os dados. Abaixo segue a lista:

- Primeiramente, foi verificado se não havia valores "missing" em cada coluna.
- Em seguida, foi separar as features categoricas e features continuas, obtendo o seguinte resultado:

Continuos Features: ['cont1', 'cont10', 'cont11', 'cont12', 'cont13', 'cont14', 'cont2', 'cont3', 'cont4', 'cont5', 'cont6', 'cont7', 'cont8', 'cont9']

Categorical Features: ['cat1', 'cat10', 'cat100', 'cat101', 'cat102', 'cat103', 'cat104', 'cat105', 'cat106', 'cat107', 'cat108', 'cat109', 'cat11', 'cat110', 'cat111', 'cat112', 'cat113', 'cat114', 'cat115', 'cat116', 'cat12', 'cat13', 'cat14', 'cat15', 'cat16', 'cat17', 'cat18', 'cat19', 'cat2', 'cat20', 'cat21', 'cat22', 'cat23', 'cat24', 'cat25', 'cat26', 'cat27', 'cat28', 'cat29', 'cat3', 'cat30',

'cat31', 'cat32', 'cat33', 'cat34', 'cat35', 'cat36', 'cat37', 'cat38', 'cat39', 'cat4', 'cat40', 'cat41', 'cat42', 'cat43', 'cat44', 'cat45', 'cat46', 'cat47', 'cat48', 'cat49', 'cat5', 'cat50', 'cat51', 'cat52', 'cat53', 'cat54', 'cat55', 'cat56', 'cat57', 'cat58', 'cat59', 'cat6', 'cat60', 'cat61', 'cat62', 'cat63', 'cat64', 'cat65', 'cat66', 'cat67', 'cat68', 'cat69', 'cat7', 'cat70', 'cat71', 'cat72', 'cat73', 'cat74', 'cat75', 'cat76', 'cat77', 'cat78', 'cat79', 'cat8', 'cat80', 'cat81', 'cat82', 'cat83', 'cat84', 'cat85', 'cat86', 'cat87', 'cat88', 'cat89', 'cat9', 'cat90', 'cat91', 'cat92', 'cat93', 'cat94', 'cat95', 'cat96', 'cat97', 'cat98', 'cat99']

- Por fim, foi visualizado os valores únicos, e obtido algumas métricas:

métricas	unique_values
count	116.000000
mean	9.818966
std	33.666807
min	2.000000
25%	2.000000
50%	2.000000
75%	4.000000
max	326.000000

- Transformação da feature loss: Foi aplicado *log* na feature loss para garantir uma distribuição mais gaussiana. Abaixo, há as imagens com as distribuições:

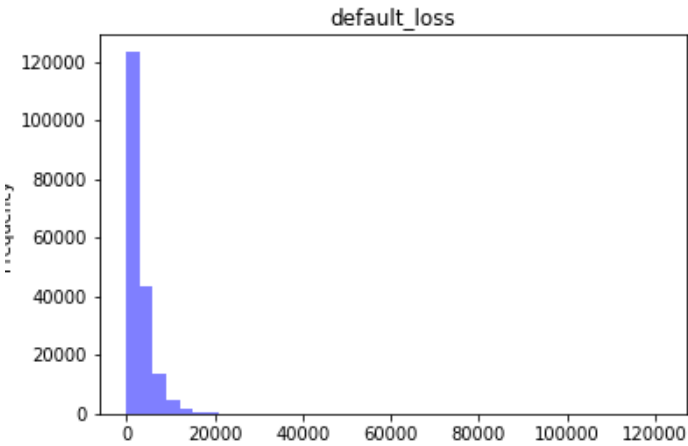


Imagem 3. Distribuição dos dados recebidos.

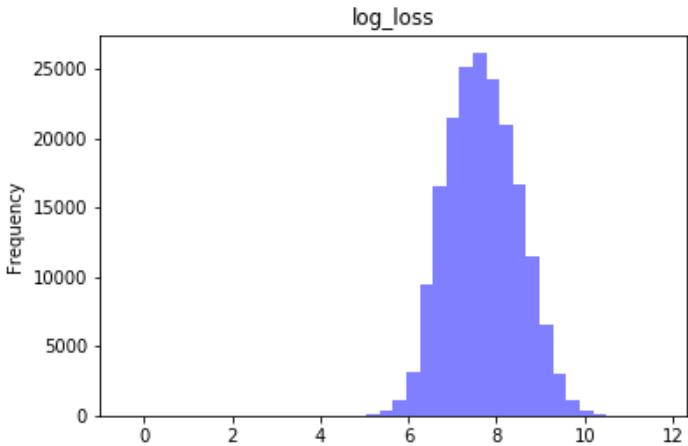


Imagem 4. Distribuição dos dados com aplicação logarítmica.

3.2 Implementação

No primeiro momento foi realizado o merge dos data sets de training e test afim de obter uma noção geral do que havia sido fornecido para o desafio. Isso facilitou a análise das features e o entendimento dos objetivos deste desafio.

Em seguida foi feito um **data cleaning** para fazer as seguintes funções:

- Checar valor ausentes
- Separar *features* categoricas e *features* continuas
- Checar valor únicos

Depois disso, foi realizado um **data preprocessing** para fazer as seguintes atividades:

- Transformação da *feature* loss para log *loss*
- Conversão de valores com *string* para valores numéricos

Na sequencia, se viu necessidade de realizar um **feature engineer** com os seguintes passos:

- Analisar graficamente a *feature* loss

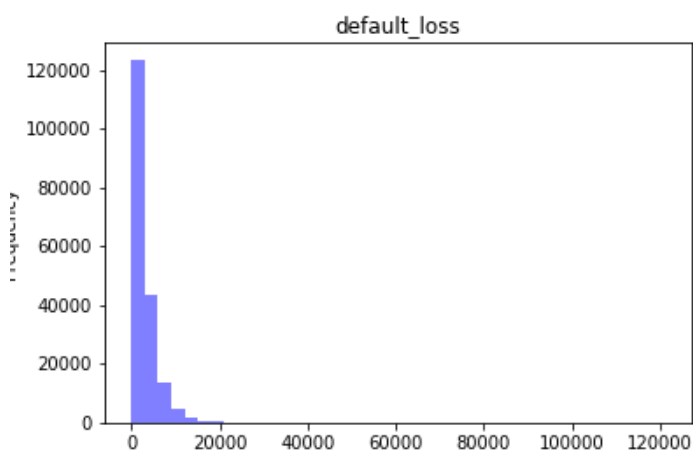


Imagem 5. Distribuição da *feature* loss.

- Analisar graficamente as *feature* contínuas

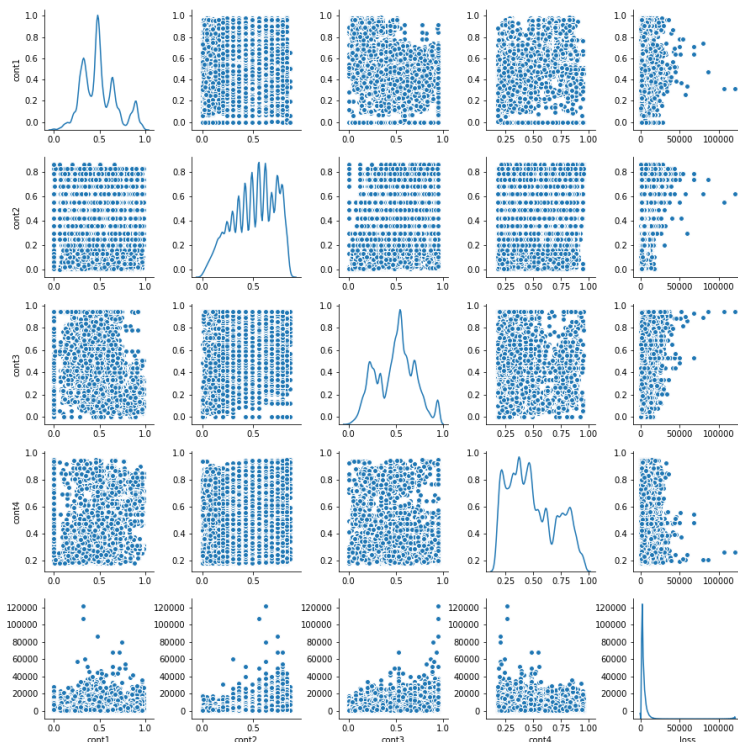


Imagem 6. correlação das *features* contínuas entre loss, 1 e 4.

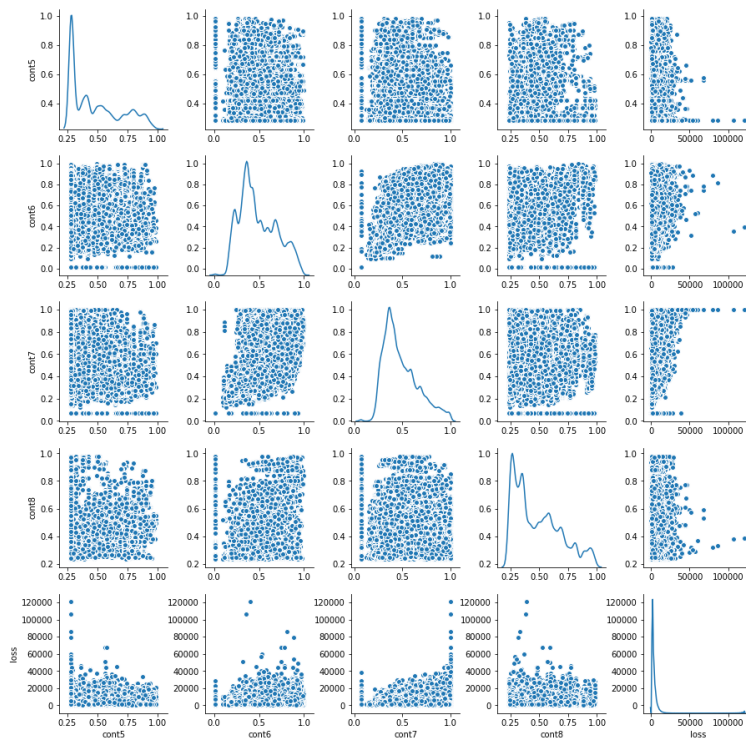


Imagem 7. correlação das *features* contínuas entre loss, 5 e 8.

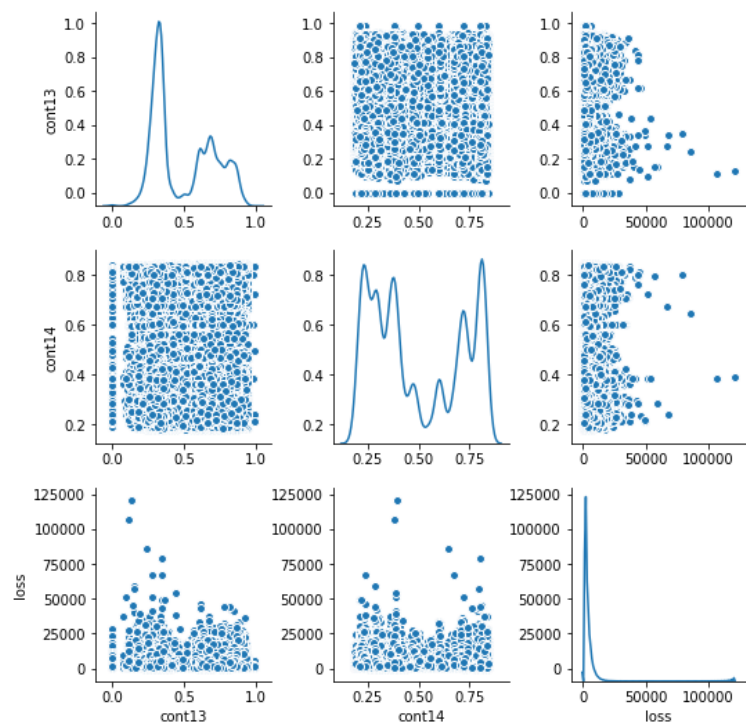


Imagem 8. correlação das *features* contínuas entre loss, 13 e 14.

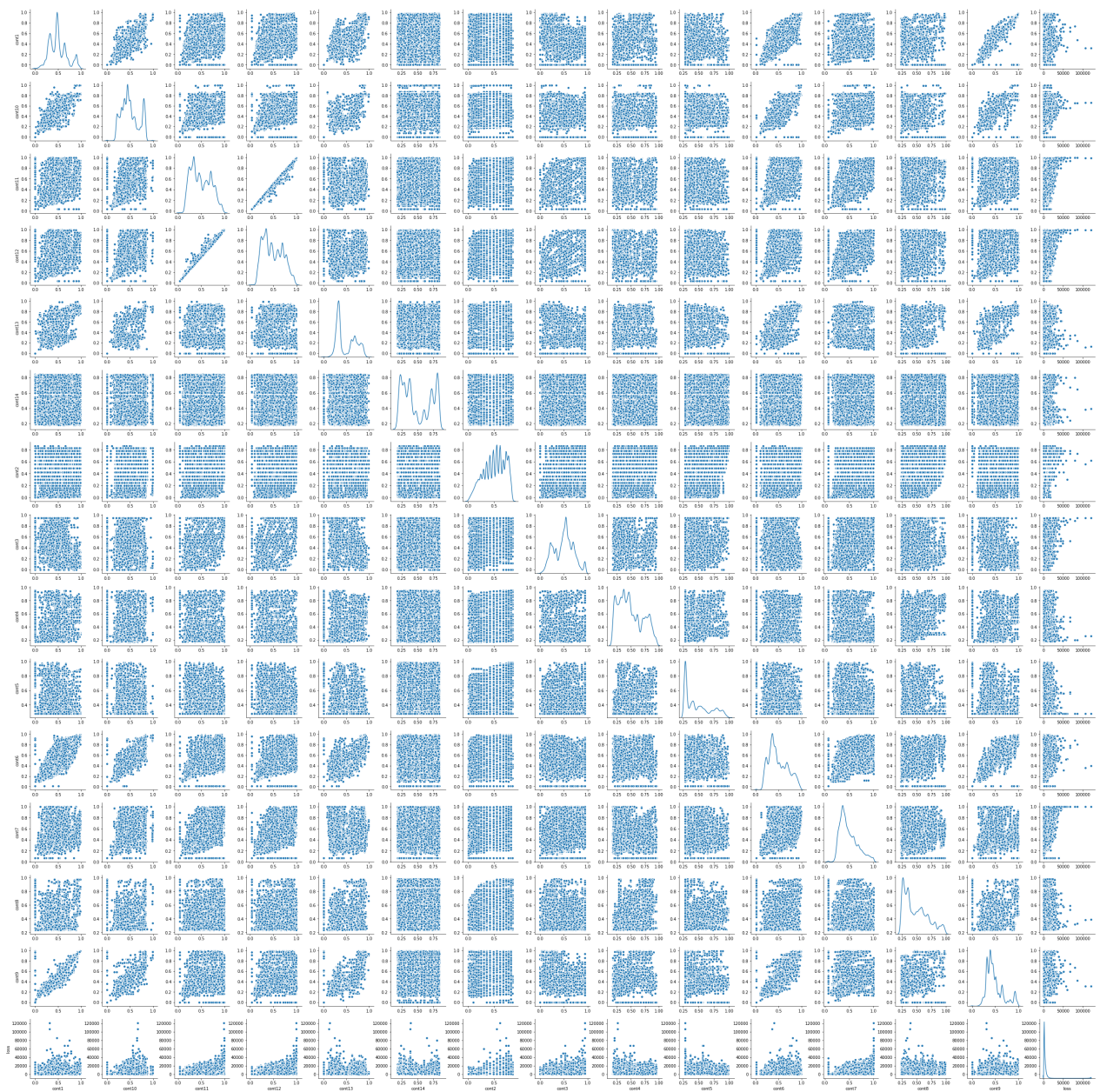


Imagem 9. correlação de todas as *features* contínuas e loss.

- Analisar a matriz de correlação das *feature* contínuas

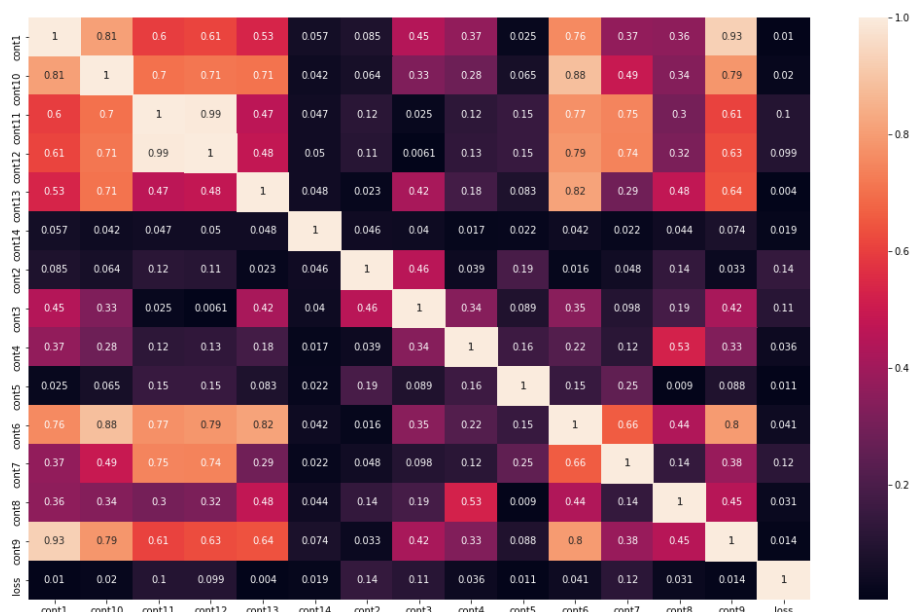


Imagem 10. Distribuição dos dados recebidos.

- Foi decidido não analisar graficamente as *features* categóricas pois havia muitas. Como melhor solução ainstao foi calculado e obtido uma lista com as 10 melhores correlações, conforme resultado abaixo:

Depois, foi analisado como deve ser feito a submissão e preparado o arquivo para isto.

Após esta fase de data engineer, foi feito um *Split train and test* e preparado a função *Mean absolute error (MAE)* e a divisão com *K-Folds Cross*.

Para não repetir código, decidiu-se que uma função training seria útil nesta fase do projeto. Esta função faz as seguintes coisas:

- Calcula tempo de processamento
- Executa a predição em cada fold
- Calcula o MAE

Código da função de training:

```
def train_model(model, num_folds):
    """Function by Train model"""

    print("Begin training")
    start = time.time()

    # declare a KFold instance
    kfold = KFold(n_splits = num_folds, random_state = 10)

    # number of models
    num_models = 1

    # array to store results after each fold
    results = np.zeros((X_test.shape[0], k))

    # train K-1 Random Forests
    for i, (train, val) in enumerate(kfold.split(X_train)):
        # get smaller training set and create validation set
        X_train_mini, X_val = X_train.iloc[train], X_train.iloc[val]
        y_train_mini, y_val = y_train[train], y_train[val]

        # train model
        model.fit(X_train_mini, y_train_mini)

        # make predictions
        preds = model.predict(X_val)
```



```

# absolute error
error = mean_absolute_error(np.exp(y_val) - shift, np.exp(preds) - shift)
print("MAE on fold {} is {}".format(i, error))

# Predict on test set
test_predictions = np.exp(model.predict(X_test)) - shift

# Sum predictions
results[:,i] = test_predictions

end = time.time()
print("\nTraining done! Time Elapsed:", end - start, " seconds.")

# Error over k folds
avg_error = np.mean(results)

return test_predictions

```

Após finalizar a preparação dos dados foi feita uma aplicação dos modelos de machine learning que serão melhor detalhados na parte de refinamento.

3.3 Refinamento

Foi utilizado três modelos de machine learning:

- Linear Regression
 - A modelagem da regressão linear foi analisada com os dados normalizados e não normalizados e se chegou a conclusão que não há diferença no resultado final.
 - O melhor resultado ocorreu com um erro absoluto médio de 1791.1369743102152
- Random Forest (Bagging)
 - Foi testado o número de estimadores para 20, 50 e 100 para encontrar a modelagem com melhor performance. Como análise de resultado observou-se que o aumento do número de estimadores melhorou o score.
 - O melhor resultado ocorreu com um erro absoluto médio de 1854.7308563695506, sendo pior em relação a regressão linear.
 - Um problema observado foi que com o aumento de estimadores o tempo de processamento também aumentou consideravelmente.
- XGBoost
 - Foi realizado três diferentes modelagens para o XGBoost
 - Primeiramente foi utilizada uma estrutura de dados interna do XGBoost para garantir o uso de memória mais eficiente e assim treinar mais rápido.
 - Para este algoritmo se viu a necessidade de criar um método específico para realizar as seguintes funcionalidades:
 - calcular o tempo de processamento
 - embaralhar os dados durante cada fold
 - executar a predição
 - armazenar a predição num array
 - analisar a média de previsões sobre os folds
 - Abaixo, segue a função:

```

def train_test_xgboost(model, early_stopping_rounds):
    kf = KFold(n_splits = k, shuffle = True, random_state = random_state)
    results = np.zeros((X_test.shape[0], k))

    for i, (train_index, val_index) in enumerate(kf.split(X_train)):

```

```

print("Begin training and testing base model on fold {}".format(i))
start = time.time()

X_train_mini, X_val = X_train.iloc[train_index], X_train.iloc[val_index]
y_train_mini, y_val = y_train[train_index], y_train[val_index]

# train model
model.fit(X_train_mini,
          y_train_mini,
          eval_metric = eval_error,
          eval_set = [(X_train_mini, y_train_mini), (X_val, y_val)],
          early_stopping_rounds = early_stopping_rounds,
          verbose = False)

end = time.time()
print("Training time elapsed on fold {} is {}".format(i, end - start))

# Predict on validation set
val_predictions = model.predict(X_val, ntree_limit = model.best_ntree_limit)
error = mean_absolute_error(np.exp(y_val) - shift, np.exp(val_predictions) - shift)
print("Error on fold {} is {} \n".format(i, error))

# Predict on test set
test_predictions = np.exp(model.predict(X_test, ntree_limit = model.best_ntree_limit)) - shift
# Sum predictions
results[:,i] = test_predictions

# Average predictions
mean_results = results.mean(axis = 1)
print(test_predictions.shape)
return mean_results

```

- A escolha dos parametro de cada um dos três modelos foi crucial para um melhor resultado.
- Para reduzir o overfitting foi testado a profundidade da árvore com 5, 7 e 9 níveis.
- Para reduzir a perda ao mínimo necessário em cada leaf da árvore foi testatdo o *gamma* com 0 e 1.
- Para garantir a soma mínima dos pesos das instancias foi testado o *min_child_weight* com 5 e 6.
- Um fato importante para se considerar é que foi mantido o parâmetro de *learning_rate* com um valor default de 0.1.
- Por fim, o melhor resultado ocorreu com um erro absoluto médio de 1753.6550894377651, sendo o melhor resultado do projeto.

IV. Resultados

4.1 Modelo de avaliação e validação

Para avaliar cada algoritmo de machine learning testado foi utilizado o erro absoluto médio (MAE) com a seguinte implemetação:

```

# Custom eval metric
def eval_error(preds, dtrain):
    """evaluation"""
    labels = dtrain.get_label()
    return 'mae', mean_absolute_error(np.exp(preds), np.exp(labels))

```

Foi feito um comparativo entre a perda prevista e a perda real para cada afirmação do conjunto de testes. Isso garantiu a escolha do algoritmo melhor performático dentre os testados.

Já para validação dos dados foi utilizado *K-Folds Cross Validation*. O KFold divide todas as amostras em grupos de amostras, chamadas dobras de tamanhos iguais (se possível). A função de predição é aprendida usando dobras e a dobra deixada de fora é usada para teste. Foi implmentado da seguinte forma:

```

# replicate the results
random_state = 16

# folds
k = 5

# declare a KFold instance
kfold = KFold(n_splits = num_folds, random_state = 10)

```

Ao testar o algoritmo de XGBoost houve a necessidade de alterar a implementação, conforme abaixo:

```
kf = KFold(n_splits = k, shuffle = True, random_state = random_state)
```

4.2 Justificativa

Após testar os três algoritmos, foi analisado qual obteve o menor erro absoluto médio. A conclusão que se chegou foi um MAE de 1753.6550894377651 para o XGBoost

V. Conclusão

5.1 Reflexão

Durante todo o projeto foi documentado da melhor forma possível através de notebooks. Essencialmente, este projeto consistiu em uma comparação de modelos de regressão linear, random forest e xgboost para criar previsões acima dos dados fornecidos.

O projeto foi revisado para garantir a melhor qualidade possível. Foi seguido as orientações da Udacity para deixar o trabalho padronizado.

A parte de visualização dos dados foi útil para tomar as decisões sobre as técnicas usadas. Todos os gráficos gerados foram salvos para poder utilizar neste documento e assim melhor explicar as suas importancias.

Ao final do projeto foi possível criar previsões muito boas, graças a forma que os dados foram tratados. Isso facilitou muito a aplicação dos algoritmos de machine learning.

5.2 Melhorias

Uma possível melhoria para este caso de predição pode ser a utilização de redes neurais artificiais treinadas para conseguir previsões mais confiáveis e modelos mais potentes. Se houver a utilização de novas técnicas como redes neurais artificiais pode ser feito um comparativo com os modelos testados neste projeto.

A solução final pode ser usada como referência pois houve a utilização e justificativa de cada modelo acima dos dados fornecidos e obtido o melhor resultado na comparação.