Insurance sales leads data analysis

A term project for the IDA I course at the University of Potsdam

The task

Which of the leads can convert to a paying customer and how certain are we about it?

You have been asked by an insurance company to analyze customer and sales data (caravan.train). These data contain 86 different kinds of information about the company's customers (see caravan.info). The insurance company would like to know from you ***which customers*** *(caravan.test) could be interested in a caravan insurance* ***and especially why****. The insurance company expects you to present your results in a credible manner - only then will the results be taken into account in the company's decision-making processes!*

Which characteristics make a lead that converts to a paying customer?

Agenda

- The data and preprocessing
- Model and metrics selection
- Comparing results and hyperparameter tuning
- Discussing the why
- Conclusions (client takeaways)

The data

- 86 features - customer characteristics
- 5821 customers train, 3999 customers test
- A mixture of categorical and numerical (L0, L2)
- Data quality: no missing values, some duplicates*, no erroneous values
- Very unbalanced classes, positive is rare - ratio: 15:1 in train
- The train data should represent the fresh data well
- One source, company's sales department
- Fixed, given dataset, no need for collection and update protocols

- Perform feature selection if necessary for dimensionality redu

```
df = pd.read_csv("data/caravan.train", delimiter
df.columns = range(1, 87)
df
```

	1	2	3	4	5	6	7	8	9	10	...	77	78	79	80
0	37	1	2	2	8	1	4	1	4	6	...	0	0	0	1
1	37	1	2	2	8	0	4	2	4	3	...	0	0	0	1
2	9	1	3	3	3	2	3	2	4	5	...	0	0	0	1
3	40	1	4	2	10	1	4	1	4	7	...	0	0	0	1
4	23	1	2	1	5	0	5	0	5	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5816	36	1	1	2	8	0	6	1	2	1	...	0	0	0	1
5817	35	1	4	4	8	1	4	1	4	6	...	0	0	0	1
5818	33	1	3	4	8	0	6	0	3	5	...	0	0	0	1
5819	34	1	3	2	8	0	7	0	2	7	...	0	0	0	0
5820	33	1	3	3	8	0	6	1	2	7	...	0	0	0	0

5821 rows × 86 columns

Preprocessing

The not necessary

No need to solve data conflicts, transform categorical to numeric*, handle duplicates*, feature construction, handle missing values, handle erroneous values, feature selection for dimensionality reduction*.

The necessary

Feature normalisation*.

Missing values

```
In [5]: df.isnull().values.any()
        #There aren't any missing val

Out[5]: False
```

No missing values!

In [6]:
```
df.duplicated().value_counts()
#There seem to be 602 duplicated items in the data.
```

Out[6]:
```
False    5219
True      602
dtype: int64
```

In [7]:
```
if (duplicated := df.duplicated(keep=False)).any():
    some_duplicates = df[duplicated].sort_values(by=df.columns.to_list()).head()
    print(f"Dataframe has one or more duplicated rows, for example:\n{some_duplicates}")
```

```
Dataframe has one or more duplicated rows, for example:
      MOSTYPE  MAANTHUI  MGEMOMV  MGEMLEEF  MOSHOOFD  MGODRK  MGODPR  MGODOV  \
2057        1         1        2         4         1       0       2       0
5699        1         1        2         4         1       0       2       0
3524        1         1        2         4         1       0       2       0
3942        1         1        2         4         1       0       2       0
983         1         1        2         4         1       0       4       0

      MGODGE  MRELGE  ...  APERSONG  AGEZONG  AWAOREG  ABRAND  AZEILPL  \
2057       7       6  ...         0        0        0       0        0
5699       7       6  ...         0        0        0       0        0
3524       7       6  ...         0        0        0       1        0
3942       7       6  ...         0        0        0       1        0
983        5       7  ...         0        0        0       0        0
```

There were 602 duplicate rows in train from example. However, I decided to keep them. The models improved.

```
In [10]: df.dtypes.value_counts()
         #All dtypes are indeed numerical!

Out[10]: int64    86
         dtype: int64
```

```
In [11]: #Converting the names of the columns to their description to better understand the values shown
         df.columns = [data_dict.get(i, {}).get('Description') for i in range (1, 87)]

         #Checking for numbers outside of the range
         with pd.option_context('display.max_rows', None, 'display.max_columns', None):
             print(df.describe().loc[['min','max']])
```

```
     Customer Subtype see L0  Number of houses 1  10  \
min                      1.0                     1.0
max                     41.0                    10.0

     Avg size household 1  6  Avg age see L1  Customer main type see L2  \
min                      1.0             1.0                        1.0
max                      5.0             6.0                       10.0

     Roman catholic see L3  Protestant ...  Other religion  No religion  \
min                    0.0             0.0             0.0          0.0
max                    9.0             9.0             5.0          9.0

     Married  Living together  Other relation  Singles  \
min      0.0              0.0             0.0      0.0
max      9.0              7.0             9.0      9.0

     Household without children  Household with children  \
min                         0.0                      0.0
max                         9.0                      9.0
```

```
In [12]: df['Married'].plot(kind='kde')

Out[12]: <AxesSubplot: ylabel='Density'>
```

Nothing strange or out of range. Not a domain expert and no access to the data creators. Left alone.

Pitch

```
In [15]: one_hot_df = pd.get_dummies(df, columns=['MOSTYPE', 'MOSHOOFD'])
         one_hot_df
```

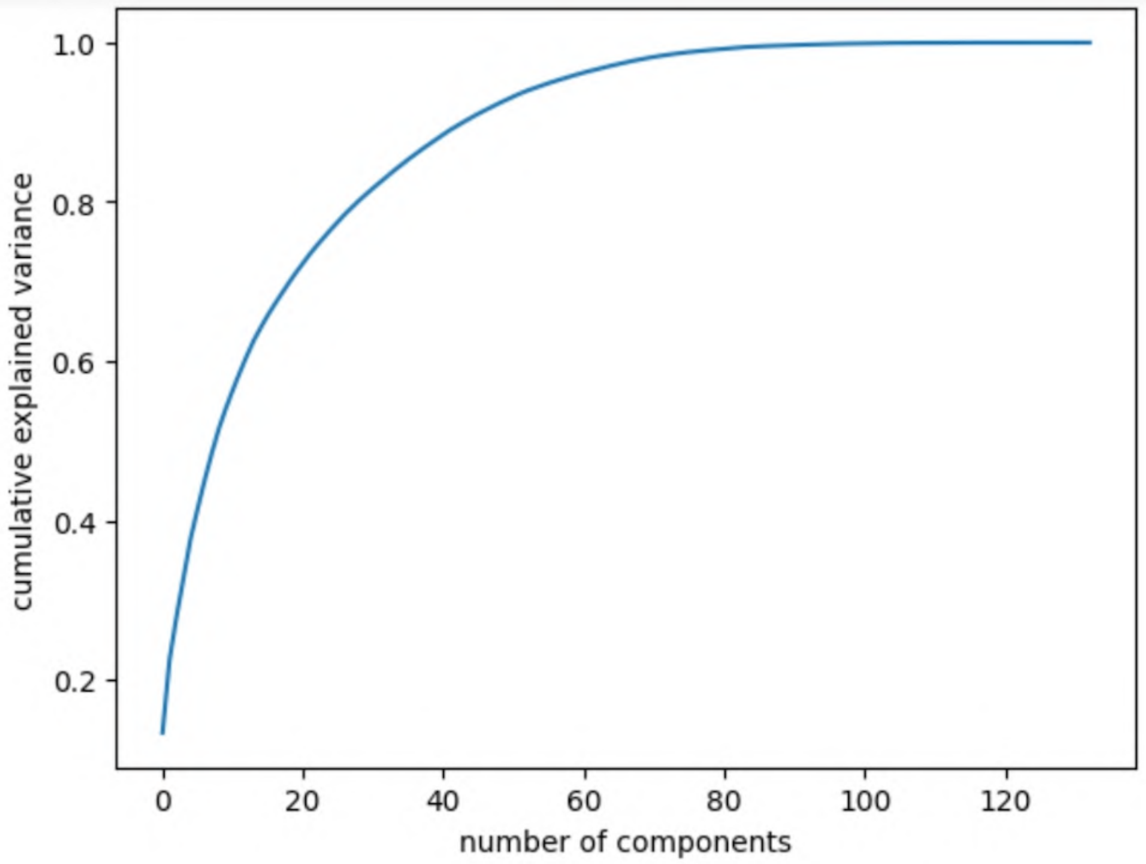
Out[15]:

	MAANTHUI	MGEMOMV	MGEMLEEF	MGODRK	MGODPR	MGODOV	MGODGE	MRELGE	MRELSA	MRELOV	...	MOSHOOFD_1	MOSHOOFD_2	MOS
0	1	2	2	1	4	1	4	6	2	2	...	0	0	
1	1	2	2	0	4	2	4	3	2	4	...	0	0	
2	1	3	3	2	3	2	4	5	2	2	...	0	0	
3	1	4	2	1	4	1	4	7	1	2	...	0	0	
4	1	2	1	0	5	0	5	0	6	3	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5816	1	1	2	0	6	1	2	1	2	6	...	0	0	
5817	1	4	4	1	4	1	4	6	0	3	...	0	0	
5818	1	3	4	0	6	0	3	5	1	4	...	0	0	
5819	1	3	2	0	7	0	2	7	2	0	...	0	0	
5820	1	3	3	0	6	1	2	7	1	2	...	0	0	

5821 rows × 133 columns

```
In [16]: scaler = MinMaxScaler()
         df_normal = scaler.fit_transform(one_hot_df)
         #df_normal = one_hot_df.apply(zscore)
         df_normal

Out[16]: array([[0.  , 0.25, 0.2 , ..., 1.  , 0.  , 0.  ],
                [0.  , 0.25, 0.2 , ..., 1.  , 0.  , 0.  ],
                [0.  , 0.5 , 0.4 , ..., 0.  , 0.  , 0.  ],
                ...,
                [0.  , 0.5 , 0.6 , ..., 1.  , 0.  , 0.  ],
                [0.  , 0.5 , 0.2 , ..., 1.  , 0.  , 0.  ],
                [0.  , 0.5 , 0.4 , ..., 1.  , 0.  , 0.  ]])
```

One hot coded the "real" categorical features for non-skewed results. Applied normalisation and used selectively.



We can see that the first 20 componenets contain the most of variance, and we need about 60 to achieve almost 100%. This means that with about 30 we would be able to capture above 90% of the variance. This will reduce our dimensionality from 86 to 30 which is a considerable amount.

```
In [20]: pca = PCA(30)  # project from 133 to 30 dimensions
         df_pca = pca.fit_transform(df_normal)
         print(df_normal.shape)
         print(df_pca.shape)
         df_pca
```

```
(5821, 133)
(5821, 30)
```

```
Out[20]: array([[-0.17981691,  0.53811949, -0.37646153, ...,  0.00339412,
                 -0.05627206,  0.15473352],
```

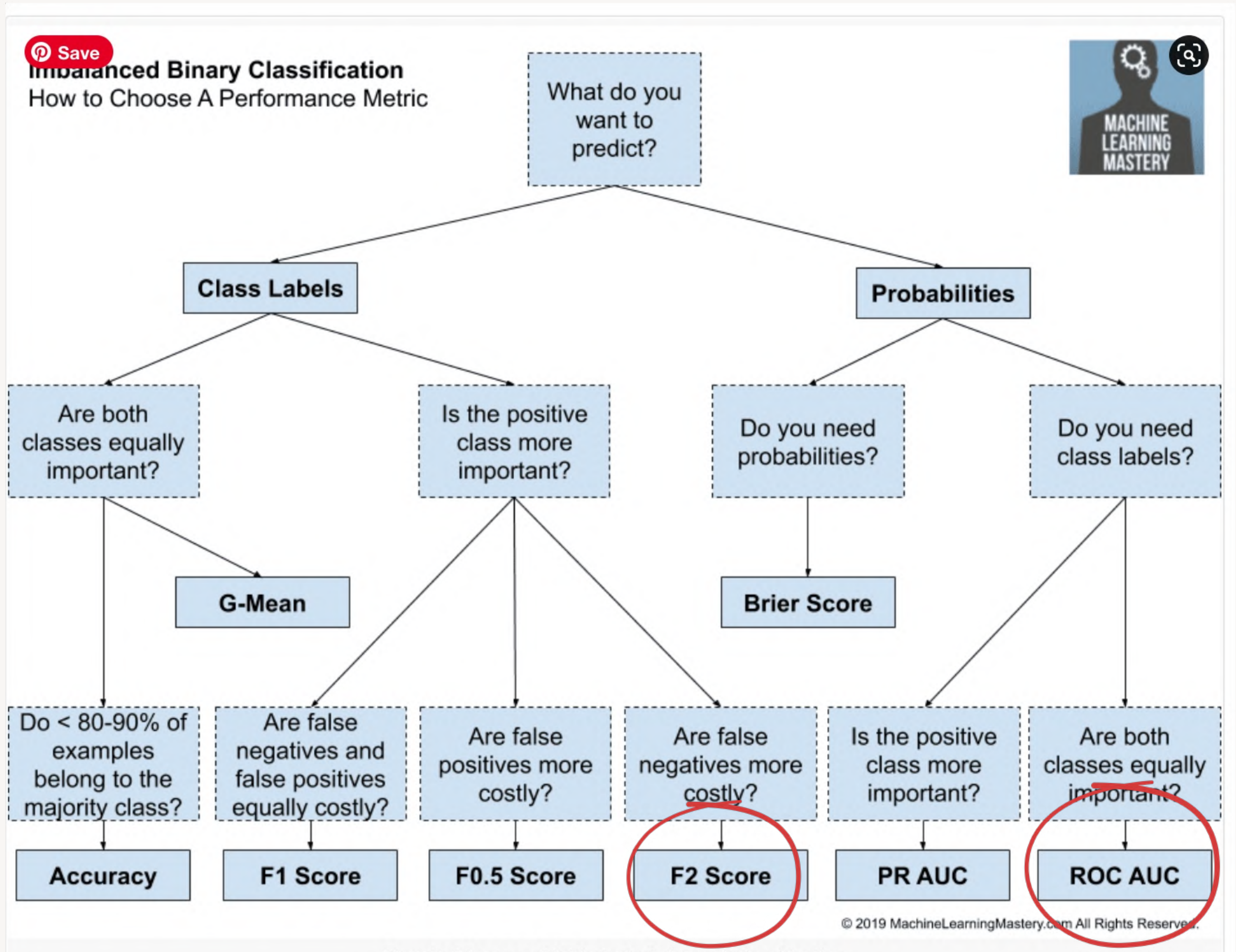
Didn't use this at the end, it was out of personal curiosity. It blocked the proper why exploration.

Model and metrics selection

- Models explored: One-class classifiers(OneClassSVM and IsolationForest, Decision trees and RandomForest(Bagging with undersampling and RandomForest with class weighting), XGBoost (with class weighting), Logistic Regression (with class weighting), **SVM (with class weighting)**
- Metrics: ROC AUC and F2

```
# svm with weighted grid search and roc auc evaluation

model = SVC()
balance = [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {(
C = [0.1,1, 10, 100]
gamma = [1,0.1,0.01,0.001]
kernel = ['rbf', 'poly', 'sigmoid']
param_grid = dict(class_weight=balance, C=C, gamma=gamma, ke
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, randor
grid = GridSearchCV(estimator=model, param_grid=param_grid, r
grid_result = grid.fit(df_normal, df_target)

# report the best configuration
print("Best: %f using %s" % (grid_result.best_score_, grid_re
report_all_configurations(grid_result)
```

```
0.662314 (0.039173) with: {'C': 100, 'class_weight': {0: 1,
0.716182 (0.039221) with: {'C': 100, 'class_weight': {0: 1,
0.743176 (0.038093) with: {'C': 100, 'class_weight': {0: 1,
0.731586 (0.034335) with: {'C': 100, 'class_weight': {0: 1,
0.740691 (0.034722) with: {'C': 100, 'class_weight': {0: 1,
0.724434 (0.041812) with: {'C': 100, 'class_weight': {0: 1,
0.742245 (0.034542) with: {'C': 100, 'class_weight': {0: 1,
0.638189 (0.030392) with: {'C': 100, 'class_weight': {0: 1,
0.640263 (0.040042) with: {'C': 100, 'class_weight': {0: 1,
0.459198 (0.041503) with: {'C': 100, 'class_weight': {0: 1,
0.662001 (0.039304) with: {'C': 100, 'class_weight': {0: 1,
0.645080 (0.040945) with: {'C': 100, 'class_weight': {0: 1,
0.612333 (0.036034) with: {'C': 100, 'class_weight': {0: 1,
0.682932 (0.037207) with: {'C': 100, 'class_weight': {0: 1,
0.689404 (0.034951) with: {'C': 100, 'class_weight': {0: 1,
0.680504 (0.044505) with: {'C': 100, 'class_weight': {0: 1,
0.698244 (0.036729) with: {'C': 100, 'class_weight': {0: 1,
0.679983 (0.034941) with: {'C': 100, 'class_weight': {0: 1,
0.695142 (0.035029) with: {'C': 100, 'class_weight': {0: 1,
```



How to Choose a Metric for Imbalanced Classification

Source: https://machinelearningmastery.com/framework-for-imbalanced-classification-projects/

Model selection and results

- One-class classifiers(OneClassSVM and IsolationForest) 🙅‍♀️
- Decision trees and RandomForest(Bagging with undersampling and RandomForest with class weighting) 🤷‍♀️
- XGBoost (with class weighting) 🙋‍♀️
- Logistic Regression (with class weighting) 🙋‍♀️
- **SVM (with class weighting)** 🙋‍♀️

- Used RepeatedStratifiedKFold and Cross Validation/GridSearch combined with metrics to evaluate

Switching to the notebook shortly...

(scrolling through the various models and results)

Best model: why the weighted SVM?

- Showing comparable results with the Logistic Regression as two best models
- They why is very easy to infer because built in feature importance (can be done in LR too, SVM easier)
- The same model that maximises the ROC AUC also maximises F2
- With hyperparameter tuning XGBoost and LG could've potentially(?) been better
- …picking it over XGBoost as it is a model we covered in the lectures
- Optimising for recall slightly better than LG

Best model predictions and explorations

In [91]:
```python
from numpy import mean

best_model_roc_auc = SVC(C=1, class_weight={0: 1, 1: 10}, gamma=0.01, kernel='rbf')
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(best_model_roc_auc, df_normal, df_target, scoring='roc_auc',
                         cv=cv, n_jobs=-1)

print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.744
```

In [92]:
```python
from numpy import mean

best_model_f2 = SVC(C=1, class_weight={0: 1, 1: 10}, gamma=0.01, kernel='rbf')
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(best_model_f2, df_normal, df_target, scoring=make_scorer(fbeta_score, beta=2),
                         cv=cv, n_jobs=-1)

print('Mean F2: %.3f' % mean(scores))
```

```
Mean F2: 0.343
```

Winning hyperparamters, same for both values.

Which customers and why

and other insights for the client

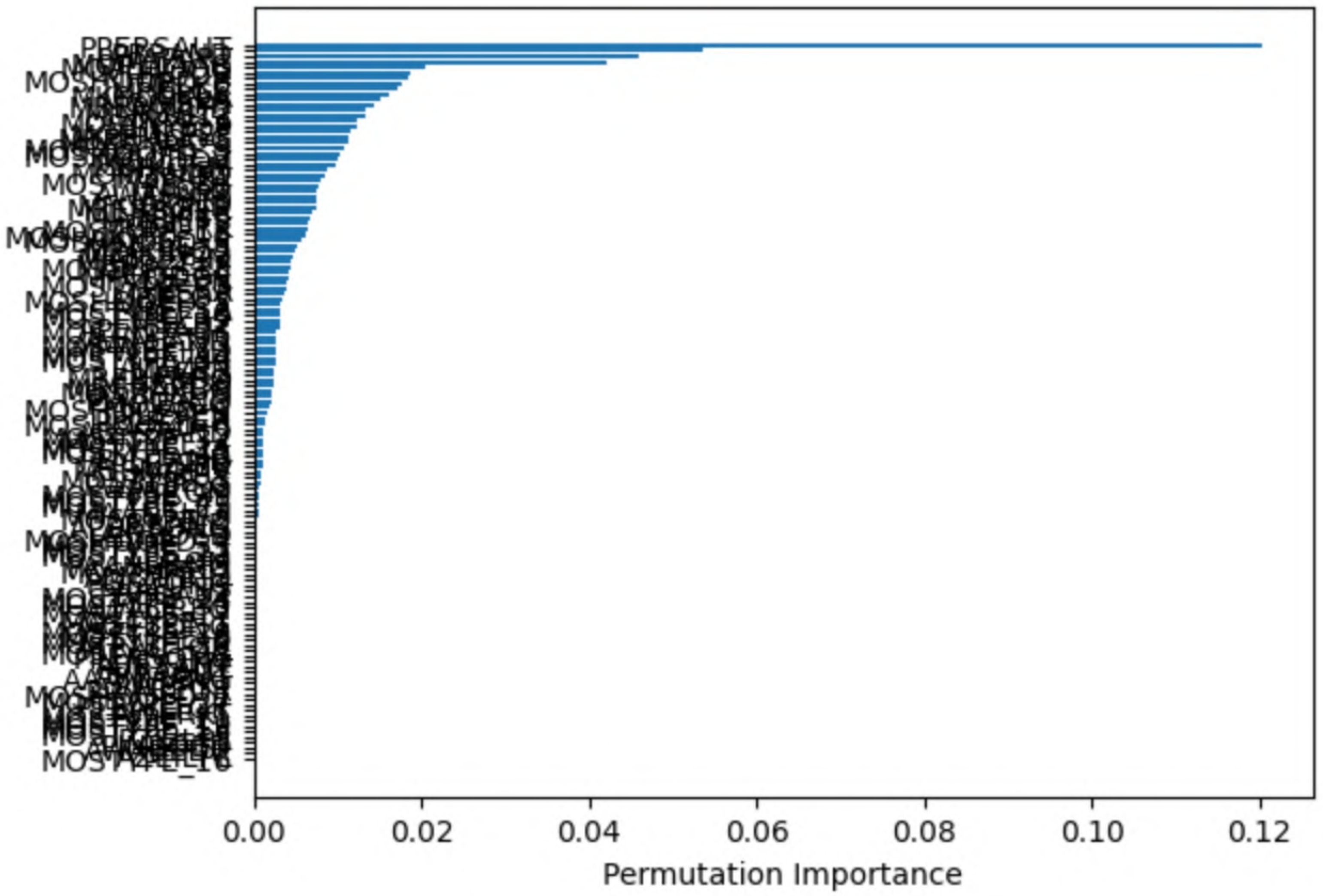
```
#Predict the response for test dataset
y_pred = best_model.predict(df_test_normal)
```

```
In [101]: from collections import Counter

          print(Counter(y_pred))

          df_test_normal[np.where(y_pred == 1)]

Out[101]: Counter({1: 603, 0: 3396})

In [*]:   from sklearn.inspection import permutation_importance
          %matplotlib inline
```

Which customers?

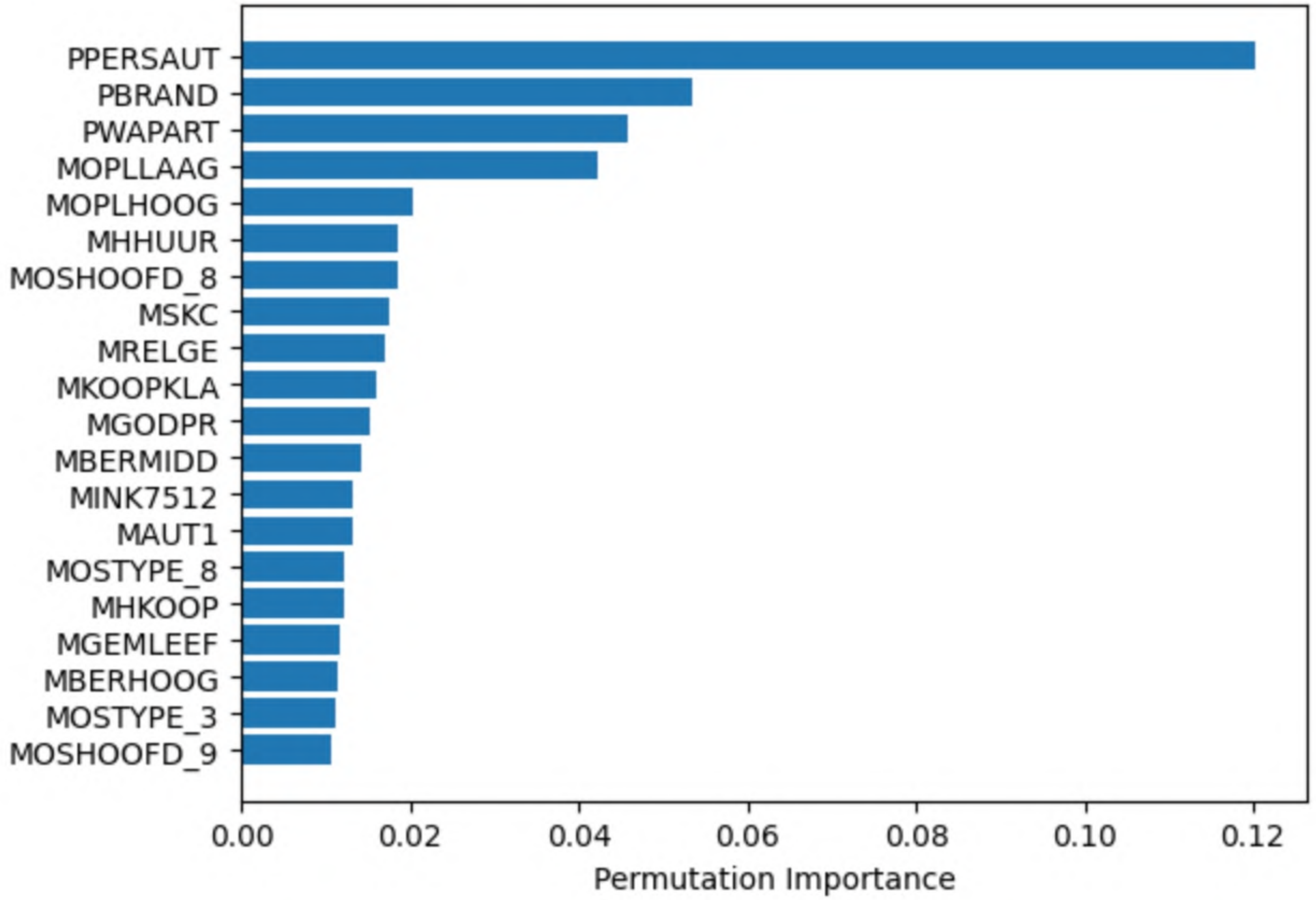
The model has identified 603 promising leads on the basis of what it learned from the training data. It does seem a bit generous in number when we think of the 15:1 ratio of the training data, but it does provide a better starting point for the sellers than being completely in the dark. This model maximises both the ROC AUC and the F2(Recall), so it offers the company more value for each call by highlighting the true positives.

```
In [119]: features = np.array(one_hot_df_test.columns)

          plt.barh(features[sorted_idx], perm_importance.importances_mean[sorted_idx])
          plt.xlabel("Permutation Importance")

Out[119]: Text(0.5, 0, 'Permutation Importance')
```



The importance of all features

```
In [120]: plt.barh(features[sorted_idx[-20:]], perm_importance.importances_mean[sorted_idx[-20:]])
          plt.xlabel("Permutation Importance")

Out[120]: Text(0.5, 0, 'Permutation Importance')
```



- **Contribution car policies**
- **Contribution fire policies**
- **Contribution private third party insurance**
- **Lower level education**
- High level education
- Rented house
- Customer main type: Family with grown kids
- Social class C
- Married
- Purchasing power class
- Protestant
- Middle management
- Income 75-122.000
- 1 car
- Customer type: Middle class families
- …

Which characteristics made the model decide in favour of these customers as possible buyers?

Conclusions (client takeaways)

- 603 possible leads
- Top most important features contributing considerably more than the rest: contribution car policies, contribution fire policies, contribution private third party insurance, lower level education
- Best model: weighted SVM. Promising models: Logistic Regression, XGBoost
- The models can be tweaked depending on what is important for business: calls leading to sales or maybe a broader lead search?

Thank you!

GitHub link: https://github.com/TamaraAtanasoska/IDA-Final-Project