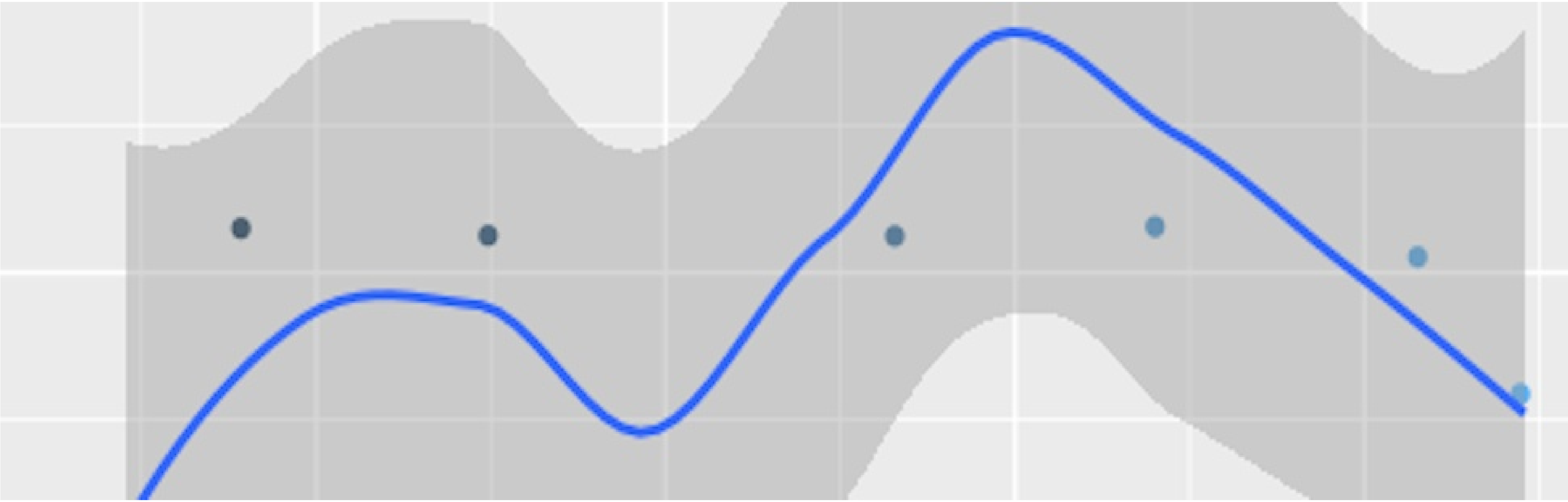


Abschlusspräsentation

Data Mining: Kaufen oder Warten?

Gruppe A: Hung Anh Le, Toan Tran Quoc

Florian Stüber, Frank Kräßke

Agenda

1) **Features**
2) **Modelle**
    2.1) Decision Tree
    2.2) Support Vector Machines (SVM)
    2.3) Logistische Regression
    2.4) Random Forest
3) **Vergleich der Modelle**
    3.1) Accuracy-Score
    3.2) F1-Score
    3.3) Monetäres Maß

1. Features: Rückblick auf Teilprojektaufgabe 01

- **Anzahl**: 27 encodierte Features
- **Features**:

Standort: *Departure_X, Destination_X*

Preis: *Price_Dev_Cat, Price_In_eur, Price_Dev, Price_Dev_Three_Days, Same_Day_Request_Route_Flight_Price*

Zeit: *Request_Month, Request_Time, Request_Day, Request_Count, Request_Count_Sum, Last_Request_Bool, Flight_Day, Departure_hour, Hours_to_Flight*

Holiday: *Is_Holiday_X, Is_School_Holiday_X*

2.1 Modelle: Decision Tree Classifier

- Trainieren mit Standardparametern
- Hyperparameter Tuning mit GridSearch und RandomizedSearch
- Hyperparameter Tuning mit dem Monetären Gütemaß

2.1 Modelle: Decision Tree Classifier

- Trainieren mit Standardparametern: Gini vs. Entropy

```
Results Using Gini Index:
Predicted values:
[0 0 0 ... 0 0 0]
Confusion Matrix:  [[14973  1254]
 [ 1293  3386]]
Accuracy :  87.81689467138621
Report :               precision    recall  f1-score   support

           0       0.92      0.92      0.92     16227
           1       0.73      0.72      0.73      4679

    accuracy                           0.88     20906
   macro avg       0.83      0.82      0.82     20906
weighted avg       0.88      0.88      0.88     20906
```

```
Results Using Entropy:
Predicted values:
[0 1 0 ... 0 0 0]
Confusion Matrix:  [[15004  1223]
 [ 1330  3349]]
Accuracy :  87.78819477661915
Report :               precision    recall  f1-score   support

           0       0.92      0.92      0.92     16227
           1       0.73      0.72      0.72      4679

    accuracy                           0.88     20906
   macro avg       0.83      0.82      0.82     20906
weighted avg       0.88      0.88      0.88     20906
```

2.1 Modelle: Decision Tree Classifier

- Hyperparameter Tuning mit RandomizedSearchCV

```
# Setup the parameters and distributions to sample from: param_dist
param_dist = {
    "criterion": ["gini", "entropy"],
    "max_depth": range(1, 100),
    "min_samples_split": range(2, 100),
    "min_samples_leaf": range(1, 50),
    "max_features": ["auto", "sqrt", "log2"],
}
# Instantiate a Decision Tree classifier: tree
tree = DecisionTreeClassifier(random_state=42)

# Instantiate the RandomizedSearchCV object: tree_cv
tree_cv = RandomizedSearchCV(tree, param_dist, cv=3, random_state=42, n_iter=100)
```

- Ergebnis

```
Tuned Decision Tree Parameters: {'min_samples_split': 23, 'min_samples_leaf': 42, 'max_features': 'sqrt', 'max_depth': 2, 'criterion': 'gini'}
Best score is 0.7761647374749474
```

```
0.78 accuracy with a standard deviation of 0.00
```

2.1 Modelle: Decision Tree Classifier

- Hyperparameter Tuning mit GridSearchCV

```
# Setup the parameters and distributions to sample from
params = {
  "criterion":['gini', 'entropy'],
  "max_depth":range(25,100),
  "min_samples_split":range(25,100),
  "min_samples_leaf":range(10,50),
  "max_features": ['auto', 'sqrt', 'log2']
}
# Instantiate the GridSearchCV object
grid_search_cv = GridSearchCV(
    DecisionTreeClassifier(random_state=42), params, verbose=1, cv=3, n_jobs=-1
)
```

- Ergebnis

```
Best Params: {'criterion': 'gini', 'max_depth': 28, 'max_features': 'auto', 'min_samples_leaf': 10, 'min_samples_split': 31}
Best Score: 0.8272744666602888
```

```
0.83 accuracy with a standard deviation of 0.00
```

2.1 Modelle: Decision Tree Classifier

- Tuning mit Monetärem Gütemaß:

```
# Setup the parameters and distributions to sample from
params = {
    "max_depth": [23, 28, 33],
    "min_samples_split": [26, 31, 36],
    "min_samples_leaf": [5, 10, 15],
}

# Iterate over all parameter combinations and execute money evaluation
for param in itertools.product(
    params["max_depth"], params["min_samples_split"], params["min_samples_leaf"]
):
    clf = DecisionTreeClassifier(
        max_depth=param[0],
        min_samples_split=param[1],
        min_samples_leaf=param[2],
        max_features="auto",
        random_state=42,
    )
    money_scores = []
```

2.1 Modelle: Decision Tree Classifier

- Tuning mit Monetärem Gütemaß:

```
# for each fold create a dataframe
for train_index, test_index in skf.split(X, Y):  # split() return index of each fold
    # get each fold train, test fold with index index
    x_train_fold, x_test_fold = X[train_index], X[test_index]
    y_train_fold, y_test_fold = Y[train_index], Y[test_index]
    clf.fit(x_train_fold, y_train_fold)
    y_pred = clf.predict(x_test_fold)

    X_train_1 = train_set_or.loc[test_index]
    X_train_1 = X_train_1.reset_index(drop=True)
    df  = pd.DataFrame()
    df["buy"] = y_pred
    df["flight_unique_id"] = X_train_1["flight_unique_id"]
    df["Request_Date"] = X_train_1["Request_Date"]
    df["Price"] = X_train_1["Price_In_Eur"]
    # eval with custom func and append
    score = model_quality_evaluation(df)
    money_scores.append(score)
# add list off accuracy to a dict with the combinations as a key
combinations[",".join(str(x) for x in param)] = money_scores
```

2.1 Modelle: Decision Tree Classifier

- Tuning mit Monetärem Gütemaß:

```python
# Finds the parameter combination for the Maximum and Minimum Money
for key, value in combinations.items():
  allMoney.extend(value)
  if first:
    maxMoney = max(value)
    minMoney = min(value)
    moneyParam = key
    first = False
  elif max(value) > maxMoney:
    maxMoney = max(value)
    maxParam = key
  elif min(value) < minMoney:
    minMoney = min(value)

# Print the output
print("Maximum Money That can be obtained from this model is:", maxMoney)
print("\nMinimum Money:", minMoney)
print("\nOverall Money:", np.mean(allMoney))
print("\nStandard Deviation is:", np.std(allMoney))
print("\nParams for Maximum Money:", maxParam, "(max_depth, min_samples_split, min_samples_leaf)")
print("\nAll combinations:", combinations)
print("\nList of possible accuracy:", allMoney)
```

2.1 Modelle: Decision Tree Classifier

- Ergebnis

```
Maximum Money That can be obtained from this model is: -173067.23

Minimum Money: -308867.94

Overall Money: -235928.8205185185

Standard Deviation is: 28314.611159008065

Params for Maximum Money: 28,26,5 (max_depth, min_samples_split, min_samples_leaf)
```

- Classification Report

```
Confusion Matrix:  [[15055  1172]
 [ 2242  2437]]
Accuracy :  83.66975987754711
Report :               precision    recall  f1-score   support

           0       0.87      0.93      0.90     16227
           1       0.68      0.52      0.59      4679

    accuracy                           0.84     20906
   macro avg       0.77      0.72      0.74     20906
weighted avg       0.83      0.84      0.83     20906

0.83 accuracy with a standard deviation of 0.00
Scores: [0.82892219 0.83785077 0.82477679 0.82764668 0.83067602 0.8309949
 0.82987883 0.82366071 0.82411099 0.83351937]
```

2.2 Modelle: Support Vector Machines (SVM)

1) Skalierung der Varianz auf 1 mit

```
scaler = StandardScaler(with_mean=False)
```

2) Reduzierung der Features mit PCA mit

```
pca2 = PCA(n_components=0.95)
```

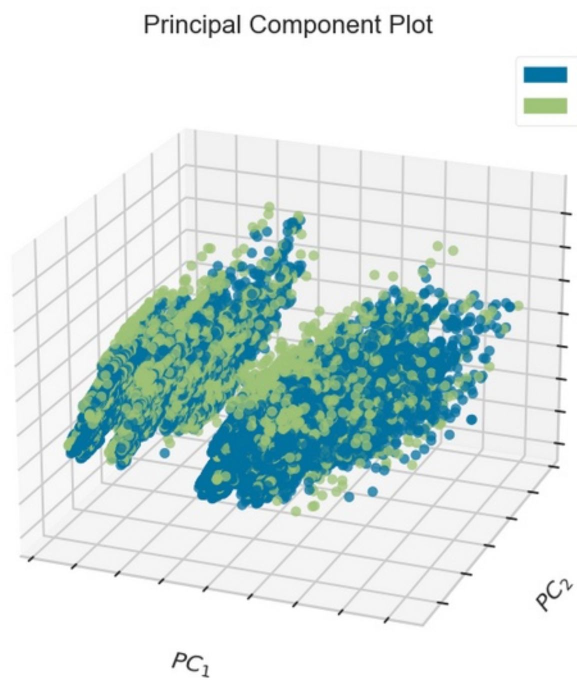
3) Trainieren mit den Standardparameter und allen oder reduzierte Features

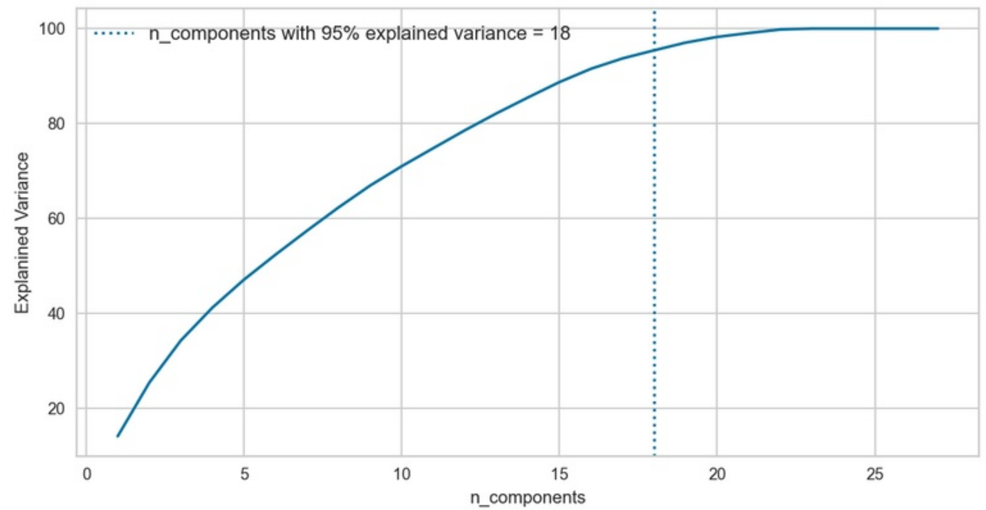
4) Hyperparameter Optimierung

5) Optimierung des monetären Maß

2.2 Modelle: Support Vector Machines (SVM)

PCA - Reduzierung des Feature-Raums auf 18 Komponenten



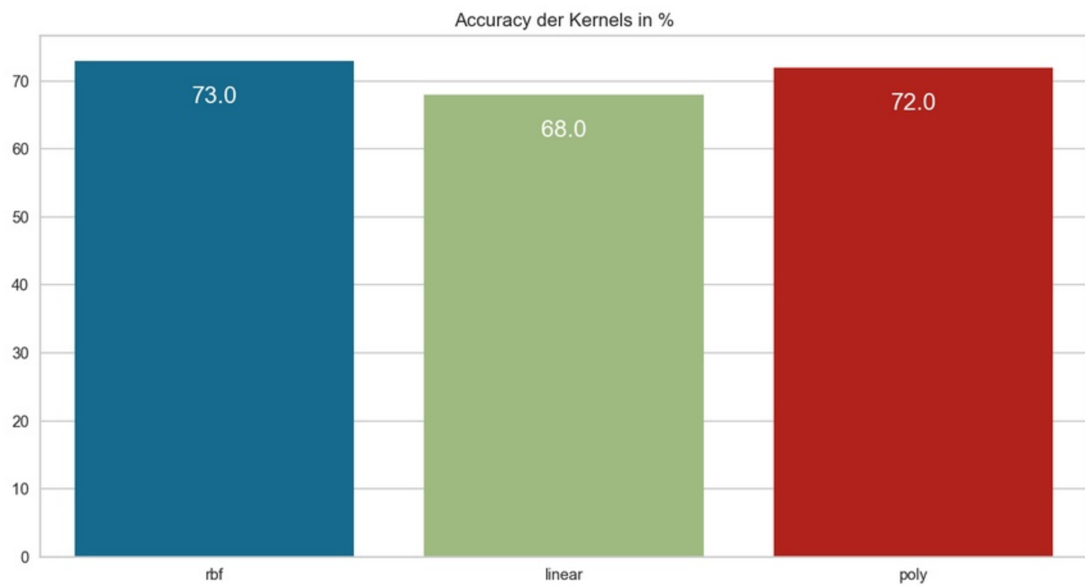


2.2 Modelle: Support Vector Machines (SVM)

Auswahl des besten Kernels:

RBF mit besten Accuracywert und Trainingszeit bei cv=10

```
scores = cross_val_score(svm_linear, X_train_scaled, y_train, cv=10, n_jobs=-1)
```



2.2 Modelle: Support Vector Machines (SVM)

1) Optimierung von Accuracy mit **Gridsearch**
   - Parameterset:

     gamma: [5, 10, 20]; [0.01, 0.1, 3]

     C: [10, 50, 100]; [0.1, 1, 5]

     dabei hatte C keine bis wenig Auswirkung auf die Accuracy

     grid = GridSearchCV(SVC(class_weight="balanced"), parameters, scoring='accuracy', cv=5)

     Gefunden: C: 5, gamma: 3, Accuracy: 79%

2) Optimierung von monetäres Maß bei cv=5

- Parameterset:

  gamma: [0.1, 1, 100, “scale”]

  C: [0.1, 1, 100]

  Gefunden: C: 1, gamma: scale, Money: -17.883

2.3 Modelle: Logistische Regression

- Report mit PCA-reduziertem Merkmalssatz (18 von 27 Merkmalen)

```
Confusion Matrix, TrainSet:
 [[51924     1]
 [14972     2]]

Classification Report:
              precision    recall  f1-score   support

           0      0.776     1.000     0.874     51925
           1      0.667     0.000     0.000     14974

    accuracy                          0.776     66899
   macro avg      0.721     0.500     0.437     66899
weighted avg      0.752     0.776     0.678     66899
```

2.3 Modelle: Logistische Regression

- Report mit normalem, nicht PCA-reduzierten Merkmalssatz

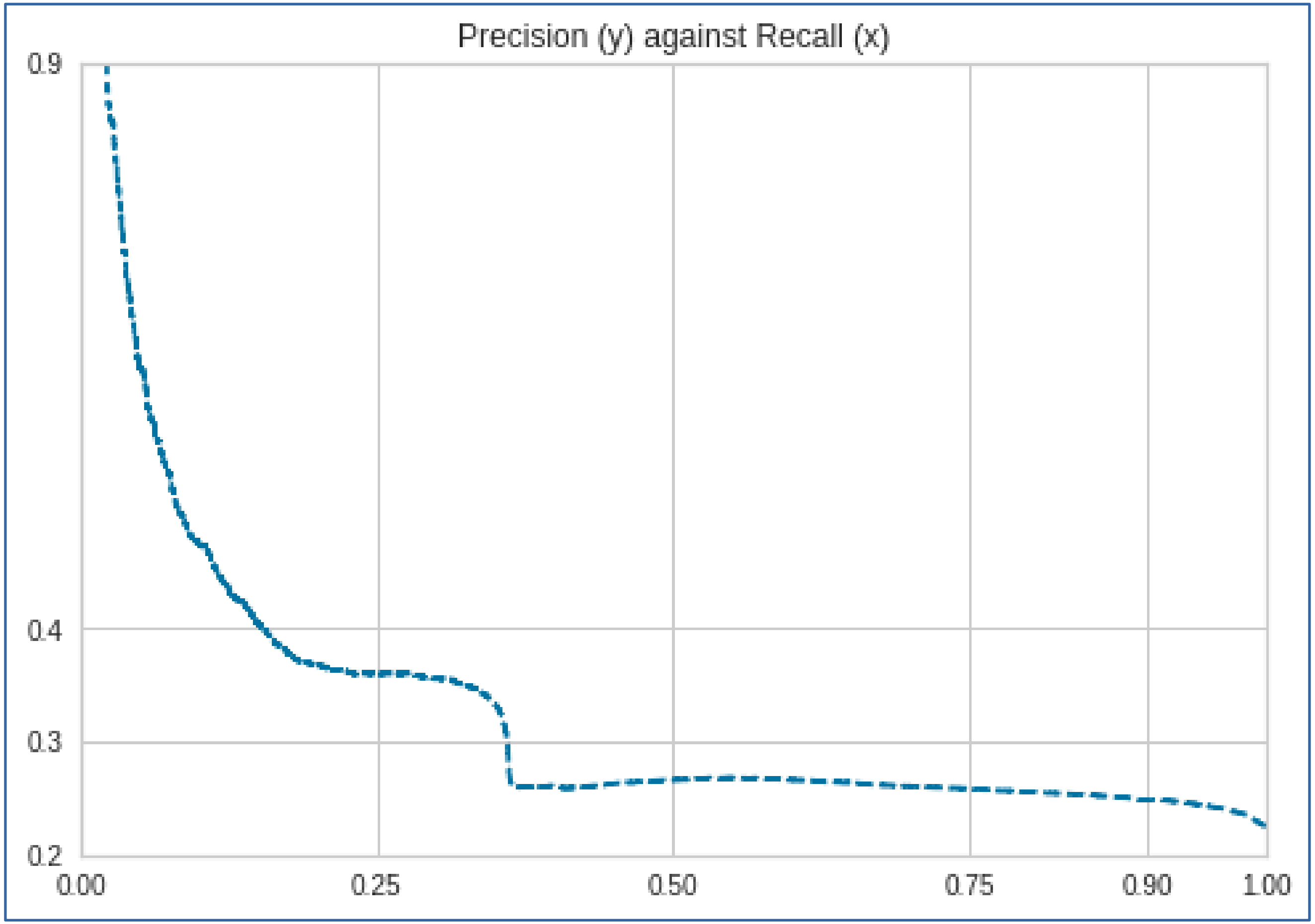
```
Confusion Matrix, TrainSet:
 [[50272  1653]
 [11190  3784]]

Classification Report:
              precision    recall  f1-score   support

           0      0.818     0.968     0.887     51925
           1      0.696     0.253     0.371     14974

    accuracy                          0.808     66899
   macro avg      0.757     0.610     0.629     66899
weighted avg      0.791     0.808     0.771     66899
```

2.3 Modelle: Logistische Regression

- Hyperparameter: Optimierung mit Gridsearch

```
tuned hpyerparameters :(best parameters)  {'C': 100000, 'penalty': 'l2'}
accuracy : 0.8080091134162439
Best parameters set found on X_train:

{'C': 100000, 'penalty': 'l2'}

Grid scores on train_set:

0.776 (+/-0.000) for {'C': 1e-05, 'penalty': 'l2'}
0.789 (+/-0.003) for {'C': 0.0001, 'penalty': 'l2'}
0.806 (+/-0.007) for {'C': 0.001, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 0.01, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 0.1, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 1, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 100, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 10000, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 100000, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 1000000, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 10000000, 'penalty': 'l2'}
0.808 (+/-0.007) for {'C': 100000000, 'penalty': 'l2'}
```

2.3 Modelle: Logistische Regression



2.4 Modelle: Random Forest Classifier

1) Trainieren mit Standardparametern

2) Hyperparameter Tuning

2.4 Modelle: Random Forest Classifier

1) Trainieren mit Standardparametern

Classification Report, Dataset mit 27 Features:

	precision	recall	f1-score	support
0	0.90	0.97	0.94	16227
1	0.87	0.62	0.73	4679
accuracy			0.90	20906
macro avg	0.89	0.80	0.83	20906
weighted avg	0.89	0.90	0.89	20906

Classification Report,

mit 18 Features nach PCA:

	precision	recall	f1-score	support
0	0.89	0.97	0.93	16227
1	0.85	0.59	0.70	4679
accuracy			0.89	20906
macro avg	0.87	0.78	0.81	20906
weighted avg	0.88	0.89	0.88	20906

2.4 Modelle: Random Forest Classifier

2) Hyperparameter Tuning

RandomizedSearchCV

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

```
rfc_random = RandomizedSearchCV(estimator=rfc, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=
42, n_jobs = 1)
```

Beste Parameter

```
rfc_random.best_params_
```

```
{'bootstrap': False,
 'max_depth': 50,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 1000}
```

2.4 Modelle: Random Forest Classifier

2) Hyperparameter Tuning

Accuracy mit den neuen Parametern:

```
rfc = RandomForestClassifier(n_estimators = 1000, max_depth=50, bootstrap=False, max_features='auto', min_samples_leaf=1, min_sa
mples_split=2)
```

```
scores = cross_val_score(rfc, trainX, trainY, cv=5)
scores
```

```
array([0.90082908, 0.89867666, 0.90186543, 0.90456829, 0.89970501])
```

Classification Report

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94     16227
           1       0.87      0.67      0.76      4679

    accuracy                           0.90     20906
   macro avg       0.89      0.82      0.85     20906
weighted avg       0.90      0.90      0.90     20906
```

2.4 Modelle: Random Forest Classifier

2) Hyperparameter Tuning

GridSearchCV:

```
param_grid = {
    'bootstrap': [False],
    'max_depth': [30, 40, 50, 60],
    'max_features': ['auto'],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [1, 2, 3],
    'n_estimators': [800, 1000, 1200]
}
```

Beste Parameter nach GridSearchCV:

```
{'bootstrap': False,
 'max_depth': 50,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 800}
```

2.4 Modelle: Random Forest Classifier

2) Hyperparameter Tuning

Accuracy mit den neuen Parametern:

```
rfc = RandomForestClassifier(bootstrap = False, max_depth=50, max_features='auto', min_samples_leaf=1, min_samples_split=2, n_estimators=800)
```

```
scores = cross_val_score(rfc, trainX, trainY, cv=5)
scores
```

```
array([0.90122768, 0.89819834, 0.90043048, 0.90544527, 0.89874831])
```

Classification Report

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94     16227
           1       0.87      0.67      0.76      4679

    accuracy                           0.90     20906
   macro avg       0.89      0.82      0.85     20906
weighted avg       0.90      0.90      0.90     20906
```

2.4 Modelle: Random Forest Classifier

2) Hyperparameter Tuning

Accuracy mit den neuen Parametern:

```
rfc = RandomForestClassifier(bootstrap = False, max_depth=50, max_features='auto', min_samples_leaf=1, min_samples_split=2, n_es
timators=800)
```

```
scores = cross_val_score(rfc, trainX, trainY, cv=5)
scores
```

```
array([0.90122768, 0.89819834, 0.90043048, 0.90544527, 0.89874831])
```

Classification Report

	precision	recall	f1-score	support
0	0.91	0.97	0.94	16227
1	0.87	0.67	0.76	4679
accuracy			0.90	20906
macro avg	0.89	0.82	0.85	20906
weighted avg	0.90	0.90	0.90	20906

2.4 Modelle: Random Forest Classifier

- Monetäres Gütemaß

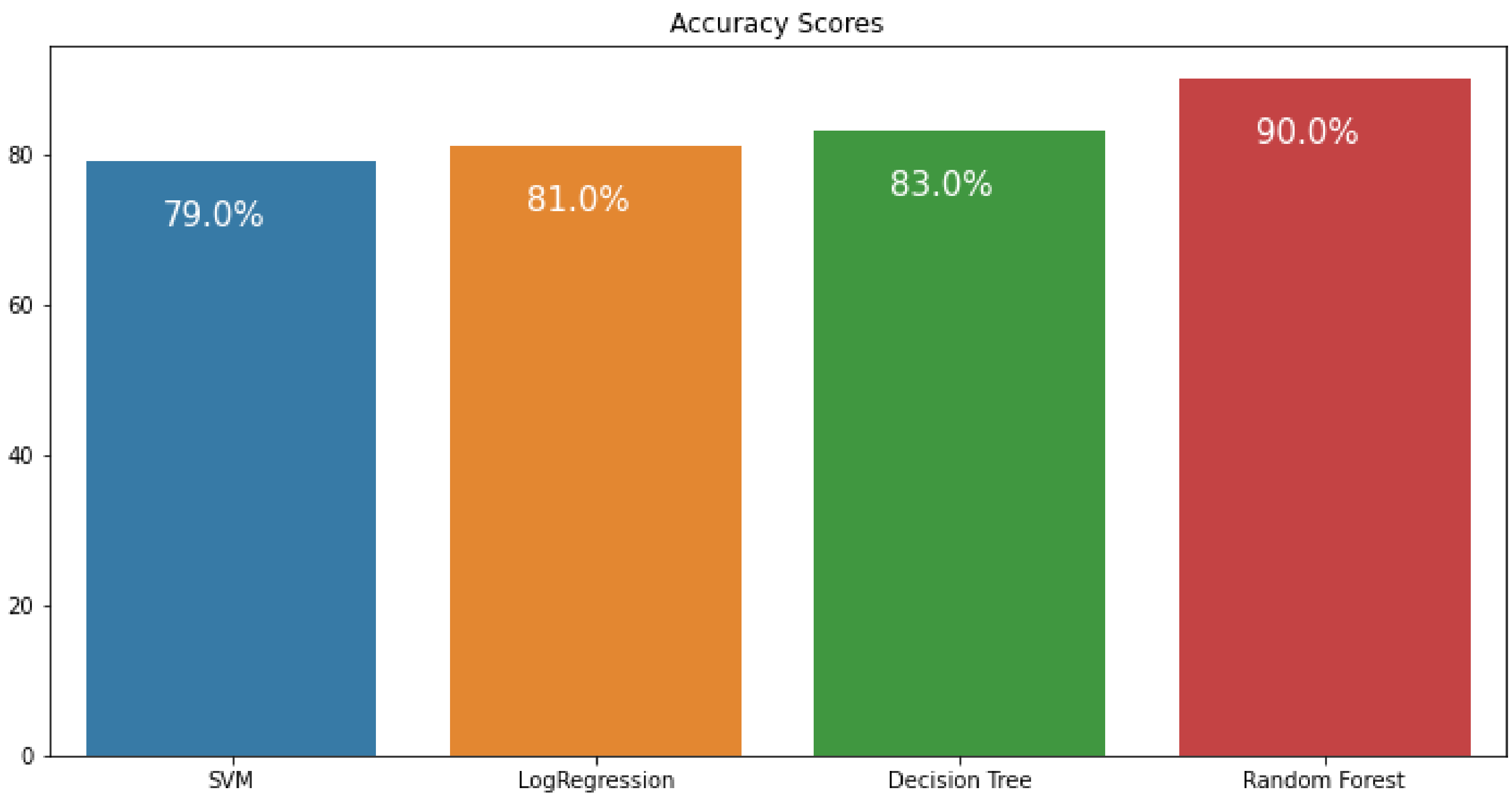
```
rfc = RandomForestClassifier(
        max_depth=50,
        max_features="auto",
        min_samples_leaf=1,
        min_samples_split=2,
        n_estimators=1000
    )
```

Das Modell macht durchschnittlich einen Verlust von -193.544€

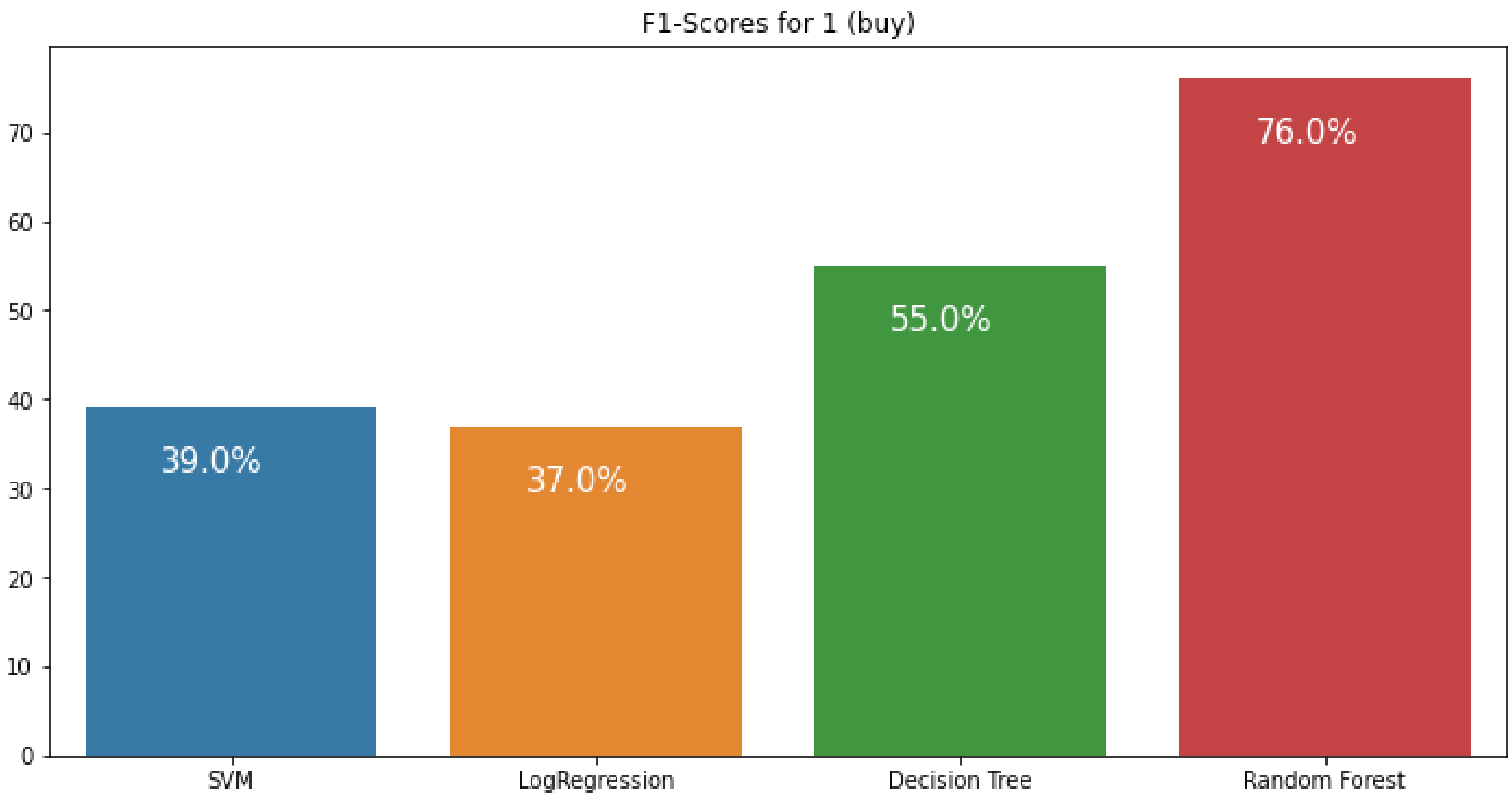
- Das Ergebnis mit den besten Parametern:

```
List of possible money scores: [-203909.1, -167363.74, -198529.13, -194726.21, -203196.54]

Mean Money: -193544.944

Standard Deviation is: 13507.494047931472
```

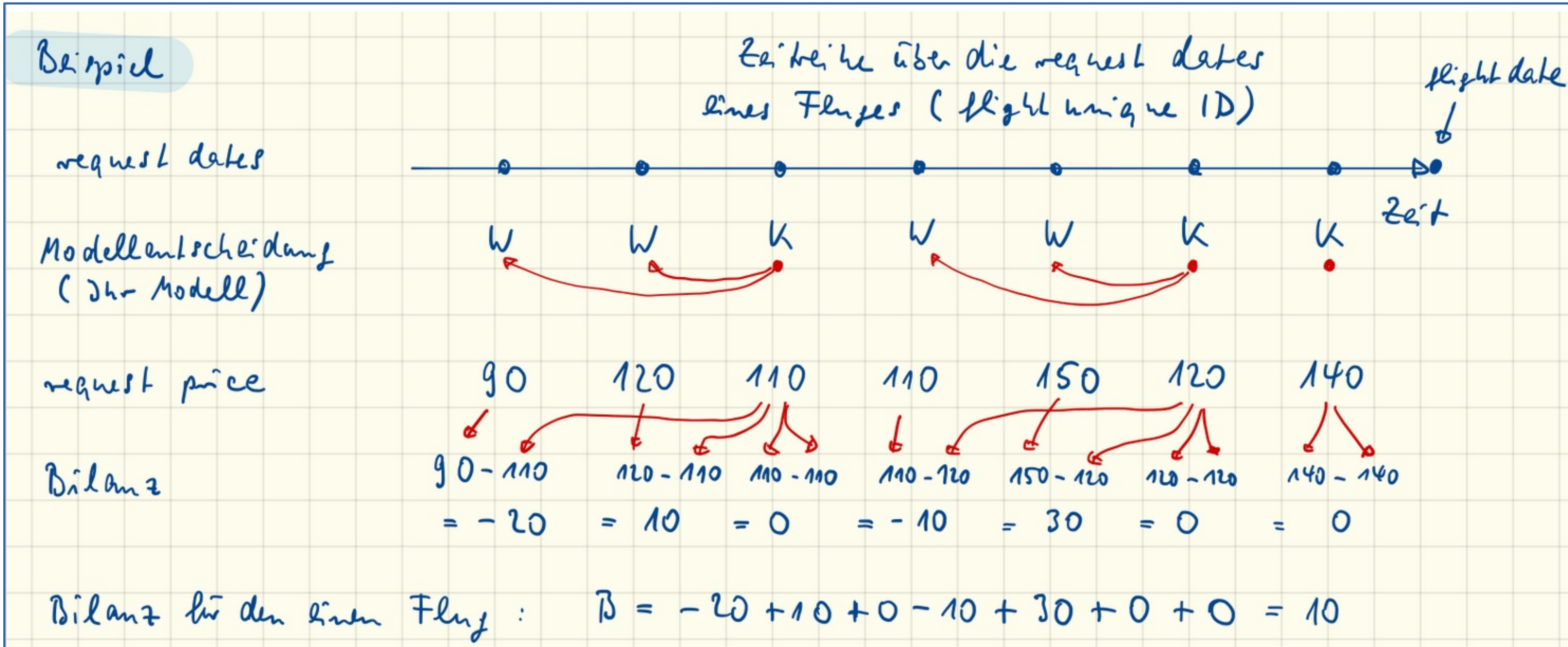
3.1 Vergleich der Modelle, mit Kriterium: Accuracy



3.2 Vergleich der Modelle, mit Kriterium: F1-Score



3.3 Monetäres Maß

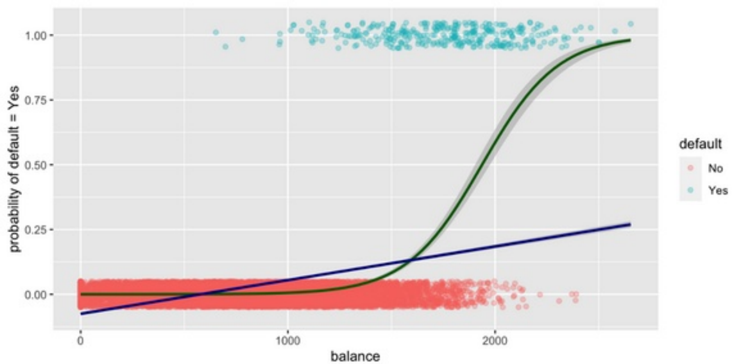


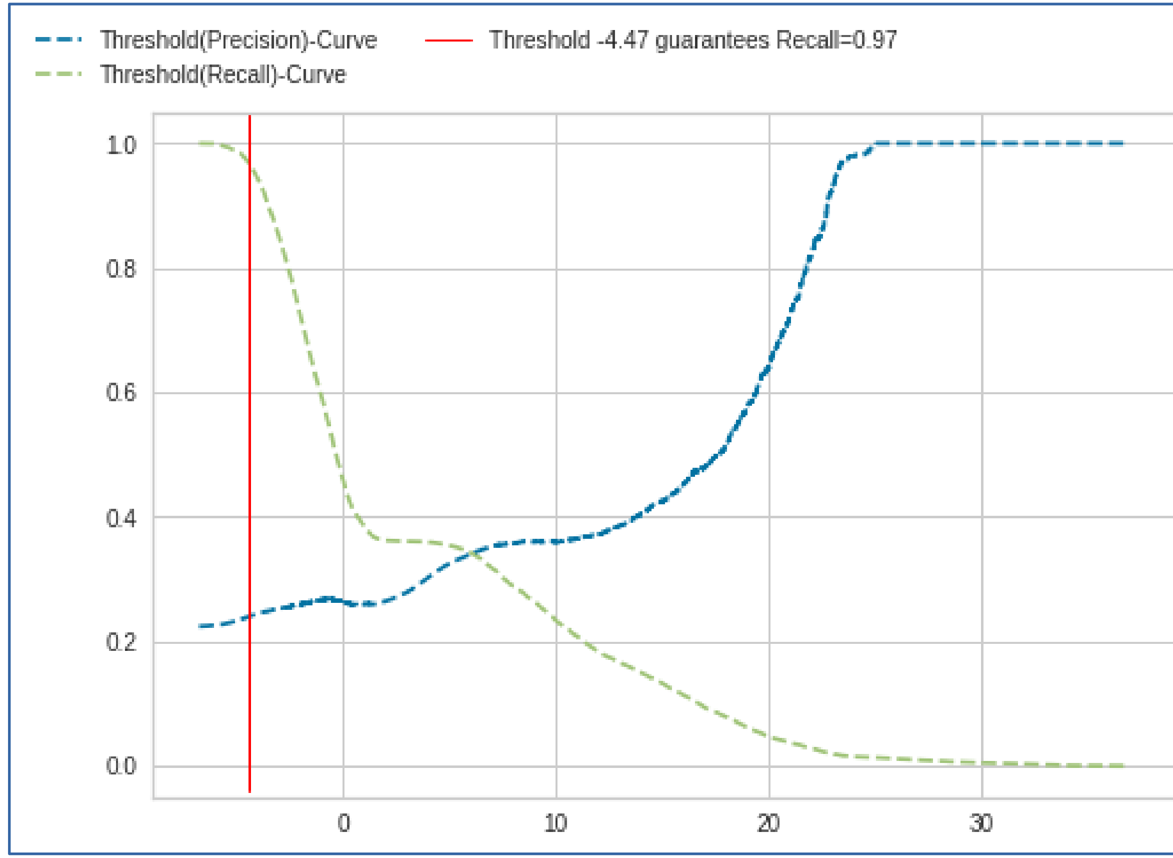
3.2 Monetäres Maß

Monetäres Gütemaß											
RequestDates	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	Summe
Modellentscheidung	K	W	K	W	W	W	W	W	W	K	
RequestPrice	90 €	120 €	110 €	110 €	150 €	160 €	160 €	155 €	140 €	170 €	
ist_zukünftiger_Kaufpreis	90 €	110 €	110 €	170 €	170 €	170 €	170 €	170 €	170 €	170 €	
Differenz/Bilanz	0 €	10 €	0 €	-60 €	-20 €	-10 €	-10 €	-15 €	-30 €	0 €	-135 €

- Begünstigt falsche Kaufempfehlungen (FP)
- Bestraft falsche ‘Noch Warten‘-Empfehlungen (FN)
- Geringe FN sind bei hohem Recall [TP/(TP+FN)] möglich
- Ziel: Hoher Recall!

3.2 Monetäres Maß





- Bestes Modell: Log. Regression
- Verschiebung des Grenzwertes (Thresholds), so dass Recall = 0.97 erreicht wird, um wenig FN zuzulassen

3.2 Monetäres Maß

```
List of possible money scores: [-467308.16, -481619.0, -494242.81, -498824.5, -503630.01]

Mean Money: -489124.896

Standard Deviation is: 13137.595960279197
```

```
Confusion matrix:
 [[ 7216 57690]
 [   562 18156]]

Classification Report:
              precision    recall  f1-score   support

           0      0.928     0.111     0.199     64906
           1      0.239     0.970     0.384     18718

    accuracy                          0.303     83624
   macro avg      0.584     0.541     0.291     83624
weighted avg      0.774     0.303     0.240     83624
```

```
List of possible money scores: [3089.22, 2382.04, 2168.78, 4279.42, 1806.55]

Mean Money: 2745.202

Standard Deviation is: 873.7722578887475
```

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de