Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____

КАФЕДРА _____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

__ Построение модели, зависящей от расстояния и рейтинга, согласующейся с наибольшей долей оценок пользователей Яндекс.Карт при выборе ресторана__

Студент _____ИУ5-31М___ _____ **Саклаков И.К.**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ **Гапанюк Ю.Е.**
(Подпись, дата) (И.О.Фамилия)

Консультант _____
(Подпись, дата) (И.О.Фамилия)

2021 г.

Оглавление

Цель работы

Регулярно пользователи Яндекс.Карт выбирают подходящий для них ресторан по множеству критериев. Для упрощения будут рассмотрены два фактора, влияющие на их выбор: расстояние до пользователя и рейтинг организации. Имеется несколько тысяч попарных оценок от реальных пользователей, в каждой из которых одна пара (расстояние, рейтинг) сравнивается с другой. Необходимо построить модель, монотонно зависящую от двух этих факторов, которая согласуется с наибольшей долей оценок.

Формат ввода

Каждая строка обучающего датасета содержит 5 чисел, разделенных табуляцией: winner, r_1 , r_2, d_1,d_2. При этом winner равен 0, если победил первый ресторан, 1, если второй и 0.5, если случалась ничья. Пары r_i, d_i, соответствуют рейтингам и расстояниям для первого и второго ресторанов. Рейтинги r_i равны либо -1, что означает, что рейтинг отсутствует, либо принимают действительные значения от 0 до 10.
Расстояния d_i равны 1, если настоящее расстояние не меньше 500 километров и отношению distance_in_kilometers / 500 в противном случае.

Descriptive Statistics

```
df = pd.read_table('restaurants_train.txt', delim_whitespace=True, names=('winner', 'r1', 'r2', 'd1', 'd2'))
df
```

	winner	r1	r2	d1	d2
0	0.5	8.154642	-1.000000	0.000552	0.000483
1	1.0	-1.000000	9.105132	0.075709	0.024765
2	0.0	7.349630	-1.000000	0.045557	0.006901
3	0.0	7.077312	-1.000000	0.000369	0.004083
4	1.0	-1.000000	-1.000000	0.002481	0.002171
...	...	...	...	...	...
995	0.0	7.731112	-1.000000	0.009292	0.007016
996	0.0	8.194613	6.501558	0.005693	0.004549
997	1.0	3.090112	5.335786	0.002175	0.001366
998	1.0	9.143142	-1.000000	0.441040	0.172363
999	1.0	-1.000000	-1.000000	0.005809	0.003495

1000 rows × 5 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   winner  1000 non-null   float64
 1   r1      1000 non-null   float64
 2   r2      1000 non-null   float64
 3   d1      1000 non-null   float64
 4   d2      1000 non-null   float64
dtypes: float64(5)
memory usage: 39.2 KB
```

```
df.describe()
```

	winner	r1	r2	d1	d2
count	1000.00000	1000.000000	1000.000000	1000.000000	1.000000e+03
mean	0.46900	4.297690	4.025088	0.049533	5.013072e-02
std	0.46718	4.515008	4.551024	0.143624	1.443761e-01
min	0.00000	-1.000000	-1.000000	0.000000	4.661730e-07
25%	0.00000	-1.000000	-1.000000	0.003998	4.080090e-03
50%	0.50000	6.636791	6.266058	0.008904	8.317535e-03
75%	1.00000	8.554018	8.452399	0.021556	2.085935e-02
max	1.00000	9.675350	9.775452	1.000000	1.000000e+00

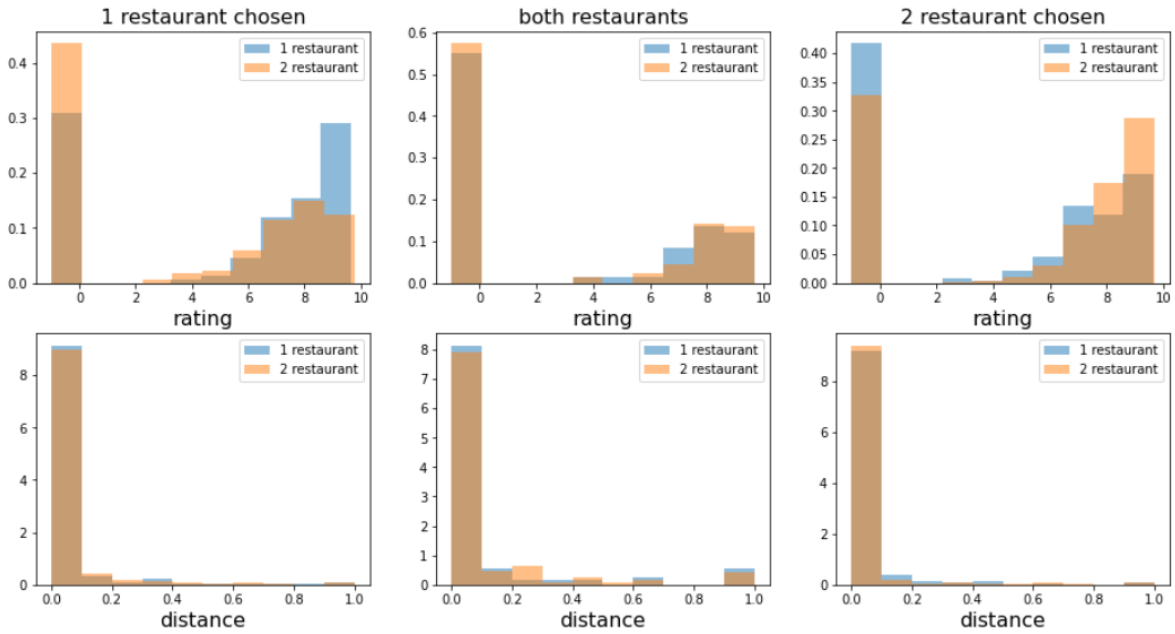
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le = le.fit(df['winner'])
df['winner'] = le.transform(df['winner'])
```

```
from sklearn.model_selection import train_test_split

# divide data into train and test samples
x, X, y, Y = train_test_split(df.drop('winner', axis = 1), df['winner'], test_size=0.1, random_state=42)
x.columns
```

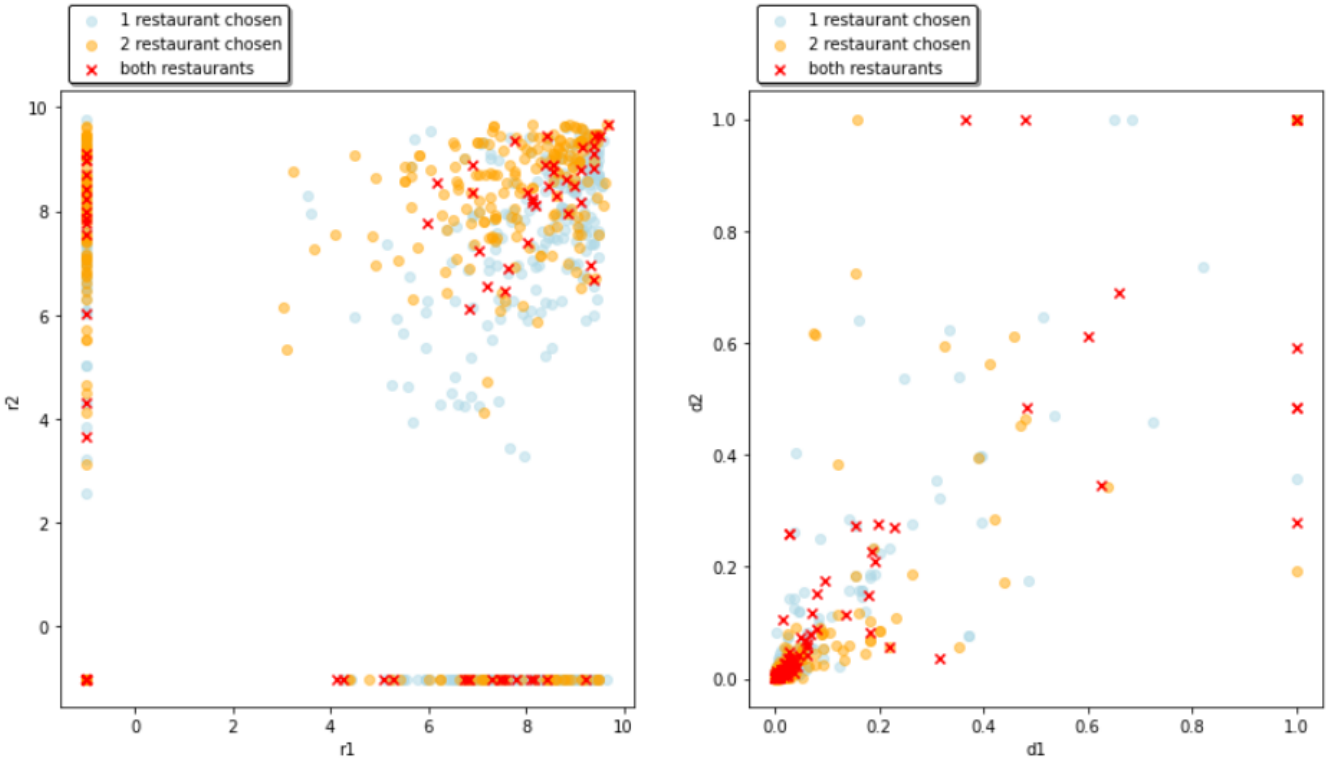
```
Index(['r1', 'r2', 'd1', 'd2'], dtype='object')
```

```
plt.figure(figsize=(16, 8))

# ratings
plt.subplot(2, 3, 1)
plt.title('1 restaurant chosen', fontsize=16)
plt.xlabel('rating', fontsize=16)
plt.hist(df[df.winner==0].r1, density=True, alpha=0.5, label='1 restaurant')
plt.hist(df[df.winner==0].r2, density=True, alpha=0.5, label='2 restaurant')
plt.legend()

plt.subplot(2, 3, 2)
plt.title('both restaurants', fontsize=16)
plt.xlabel('rating', fontsize=16)
plt.hist(df[df.winner==1].r1, density=True, alpha=0.5, label='1 restaurant')
plt.hist(df[df.winner==1].r2, density=True, alpha=0.5, label='2 restaurant')
plt.legend()

plt.subplot(2, 3, 3)
plt.title('2 restaurant chosen', fontsize=16)
plt.xlabel('rating', fontsize=16)
plt.hist(df[df.winner==2].r1, density=True, alpha=0.5, label='1 restaurant')
plt.hist(df[df.winner==2].r2, density=True, alpha=0.5, label='2 restaurant')
plt.legend()

# distances
plt.subplot(2, 3, 4)
plt.xlabel('distance', fontsize=16)
plt.hist(df[df.winner==0].d1, density=True, alpha=0.5, label='1 restaurant')
plt.hist(df[df.winner==0].d2, density=True, alpha=0.5, label='2 restaurant')
plt.legend()

plt.subplot(2, 3, 5)
plt.xlabel('distance', fontsize=16)
plt.hist(df[df.winner==1].d1, density=True, alpha=0.5, label='1 restaurant')
plt.hist(df[df.winner==1].d2, density=True, alpha=0.5, label='2 restaurant')
plt.legend()
```

```
plt.subplot(2, 3, 6)
plt.xlabel('distance', fontsize=16)
plt.hist(df[df.winner==2].d1, density=True, alpha=0.5, label='1 restaurant')
plt.hist(df[df.winner==2].d2, density=True, alpha=0.5, label='2 restaurant')
plt.legend()

plt.show()
```



```
plt.figure(figsize=(14,7))
plt.subplot(1, 2, 1)

plt.scatter(df[df.winner==0].r1, df[df.winner==0].r2,
            marker='o', label='1 restaurant chosen', c = 'lightblue', alpha=0.5)
plt.scatter(df[df.winner==2].r1, df[df.winner==2].r2,
            marker='o', label='2 restaurant chosen', c= 'orange', alpha=0.5)
plt.scatter(df[df.winner==1].r1, df[df.winner==1].r2,
            marker='x', label='both restaurants', c='red')
plt.legend(loc='lower left', shadow=True, edgecolor='k', frameon='True', bbox_to_anchor=(0, 1))
plt.xlabel('r1')
plt.ylabel('r2')

plt.subplot(1, 2, 2)
plt.scatter(df[df.winner==0].d1, df[df.winner==0].d2,
            marker='o', label='1 restaurant chosen', c = 'lightblue', alpha=0.5)
plt.scatter(df[df.winner==2].d1, df[df.winner==2].d2,
            marker='o', label='2 restaurant chosen', c= 'orange', alpha=0.5)
plt.scatter(df[df.winner==1].d1, df[df.winner==1].d2,
            marker='x', label='both restaurants', c='red')
plt.legend(loc='lower left', shadow=True, edgecolor='k', frameon='True', bbox_to_anchor=(0, 1))
plt.xlabel('d1')
plt.ylabel('d2')
plt.show()
```



MCMC approach

Модель:

$$y_i|\mathbf{x}_i, \mathbf{w} \sim \text{Categorical}\left(\theta(\mathbf{x}_i)\right), \quad i = 1, \ldots, n,$$
$$\theta(\mathbf{x}) = \text{Softmax}(\mathbf{w}_1^T\mathbf{x}, \mathbf{w}_2^T\mathbf{x}, 0),$$
$$w_{kj} \sim N(0, S_{kj}),\ k = 1, 2\, j = 0, 1, 2, 3, 4.$$

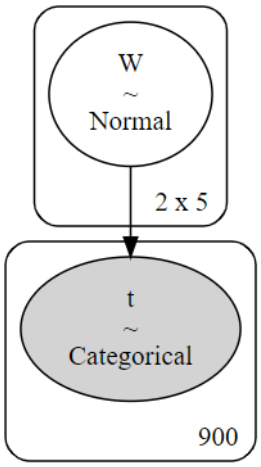
, где

J – число объясняющих регрессоров +1 свободный член
K – число классов за вычетом одного
Функция Softmax является обобщением логистической функции на несколько измерений

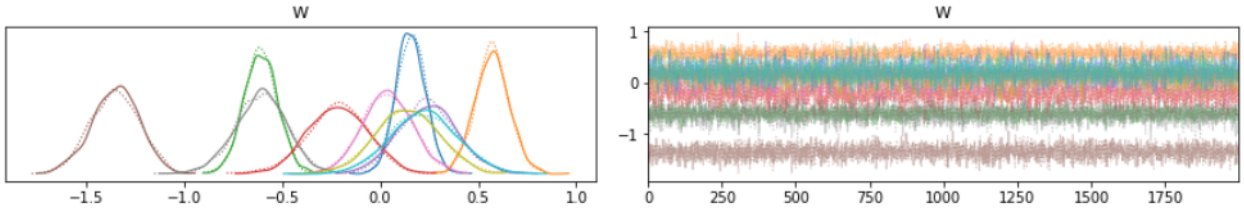
```
print('x.shape=', x.shape)
print('y.shape=', y.shape)

x_means = x.mean(axis=0)
print('\nx_means= \n',x_means)
x_scales = x.std(axis=0)
print('\nx_scales= \n', x_scales)
zX = ((x-x_means)/x_scales)
zX = np.hstack([np.ones(shape=(zX.shape[0], 1)), zX])
print('\nzX.shape: ', zX.shape)
```

```
x.shape= (900, 4)
y.shape= (900,)

x_means=
 r1    4.290365
r2    4.045761
d1    0.046435
d2    0.046914
dtype: float64

x_scales=
 r1    4.521262
r2    4.555765
d1    0.137519
d2    0.137902
dtype: float64

zX.shape:  (900, 5)
```

```
import theano.tensor as tt

with pm.Model() as model:
    W = pm.Normal('W', mu=0, sd=np.array([[5, 2, 2, 2, 2],
                                          [5, 2, 2, 2, 2]]), shape=(2,5))
    mu = tt.concatenate([pm.math.dot(zX, W.T),
                         np.zeros((zX.shape[0], 1))], axis=-1)
    outputs = pm.Categorical('t', tt.nnet.softmax(mu), observed=y)
    trace = pm.sample(draws=2000, tune=1000, chains=2, cores=2,
                      return_inferencedata=True)

pm.model_to_graphviz(model)
```




```
az.plot_trace(trace, var_names=['W']); # weights distributions and values for all draws
```



```
az.summary(trace)
```

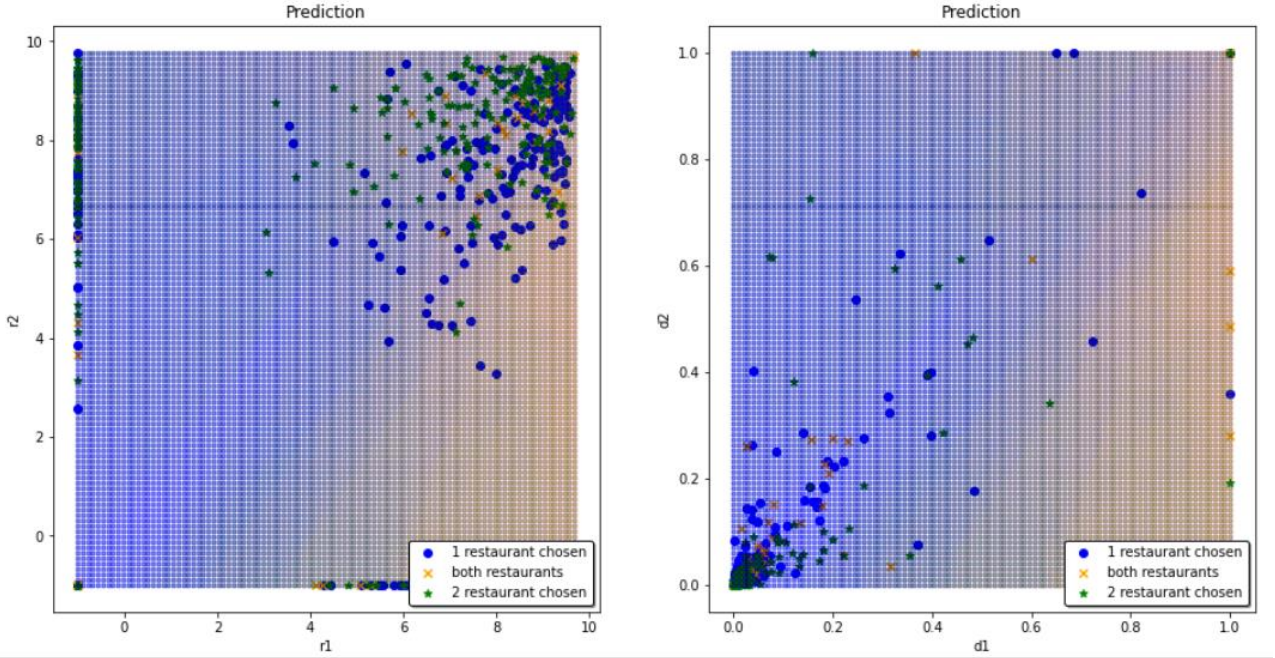
	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
W[0,0]	0.156	0.074	0.016	0.291	0.001	0.001	4637.0	3370.0	1.0
W[0,1]	0.571	0.087	0.406	0.735	0.002	0.001	2927.0	2824.0	1.0
W[0,2]	-0.610	0.086	-0.777	-0.457	0.002	0.001	3269.0	2915.0	1.0
W[0,3]	-0.217	0.153	-0.494	0.072	0.003	0.002	2010.0	2347.0	1.0
W[0,4]	0.245	0.151	-0.044	0.525	0.003	0.002	2054.0	2138.0	1.0
W[1,0]	-1.352	0.123	-1.574	-1.116	0.002	0.001	3907.0	2954.0	1.0
W[1,1]	0.037	0.130	-0.217	0.274	0.002	0.002	2976.0	2202.0	1.0
W[1,2]	-0.609	0.132	-0.868	-0.375	0.002	0.002	3511.0	3065.0	1.0
W[1,3]	0.138	0.161	-0.172	0.434	0.004	0.003	1945.0	2483.0	1.0
W[1,4]	0.223	0.169	-0.084	0.551	0.004	0.003	1961.0	2647.0	1.0

```
# transform weights back to original scale

w_trace = trace.posterior['W'][0]

for k in range(w_trace.shape[1]):
    w_trace[:, k, 0] = w_trace[:, k, 0] \
                     - x_means[0] / x_scales[0] * w_trace[:, k, 1] \
                     - x_means[1] / x_scales[1] * w_trace[:, k, 2] \
                     - x_means[2] / x_scales[2] * w_trace[:, k, 3] \
                     - x_means[3] / x_scales[3] * w_trace[:, k, 4]
    w_trace[:, k, 1] /=  x_scales[0]
    w_trace[:, k, 2] /=  x_scales[1]
    w_trace[:, k, 3] /=  x_scales[2]
    w_trace[:, k, 4] /=  x_scales[3]

w_trace = np.concatenate([w_trace,
                          np.zeros(shape=(w_trace.shape[0], 1, w_trace.shape[2]))],
                         axis=1)
w_trace.shape
```

```
(2000, 3, 5)
```

```
markers = [ 'o', 'x', '*']
label_name = ['1 restaurant chosen', 'both restaurants', '2 restaurant chosen']
color_name = [ 'blue', 'orange', 'green']

plt.figure(figsize=(16,8))
plt.subplot(1, 2, 1)

r1_grid = np.linspace(x['r1'].min(), x['r1'].max(), 100)
r2_grid = np.linspace(x['r2'].min(), x['r2'].max(), 100)
d1_grid = np.linspace(x['d1'].min(), x['d1'].max(), 100)
d2_grid = np.linspace(x['d2'].min(), x['d2'].max(), 100)
r1, r2 = np.meshgrid(r1_grid, r2_grid)
r1 = r1.reshape(np.prod(r1.shape))
r2 = r2.reshape(np.prod(r2.shape))
d1, d2 = np.meshgrid(d1_grid, d2_grid)
d1 = d1.reshape(np.prod(d1.shape))
d2 = d2.reshape(np.prod(d2.shape))
xxx = np.stack([np.ones(r1.shape), r1, r2, d1, d2])


for r in np.arange(0, w_trace.shape[0], 50):  # every 50th step in MC
    M = np.dot(w_trace[r,:,:], xxx)
    t_predict = np.argmax(M, axis=0)
    t_predict.shape
    for k in range(3):
        plt.plot(r1[t_predict==k], r2[t_predict==k],
                 marker='.', color=color_name[k], linestyle='none', alpha=0.01);
for k in range(3):
    plt.scatter(x['r1'][y==k], x['r2'][y==k],
                marker=markers[k], label=label_name[k], color=color_name[k])

plt.legend(loc='lower right', shadow=True, edgecolor='k', frameon='True')
plt.xlabel("r1")
plt.ylabel("r2")
plt.title('Prediction')
```

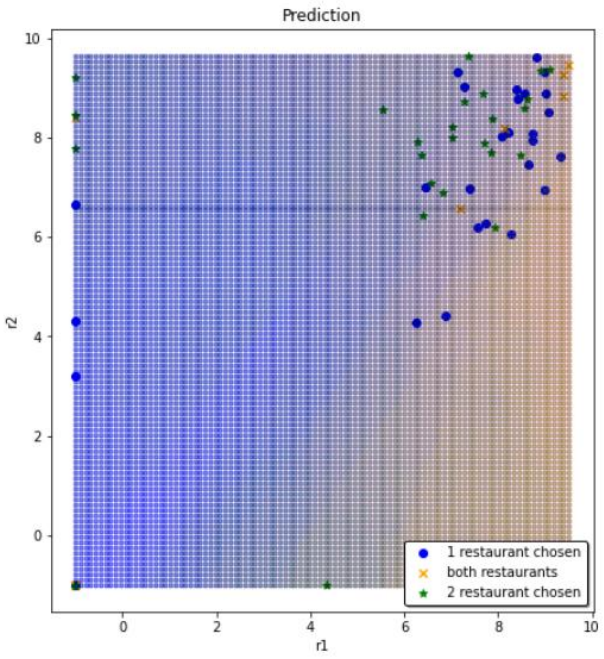
```
        plt.plot(d1[t_predict==k], d2[t_predict==k],
                 marker='.', color=color_name[k], linestyle='none', alpha=0.01);
for k in range(3):
    plt.scatter(x['d1'][y==k], x['d2'][y==k],
                marker=markers[k], label=label_name[k], color=color_name[k])

plt.legend(loc='lower right', shadow=True, edgecolor='k', frameon='True')
plt.xlabel("d1")
plt.ylabel("d2")
plt.title('Prediction');
```

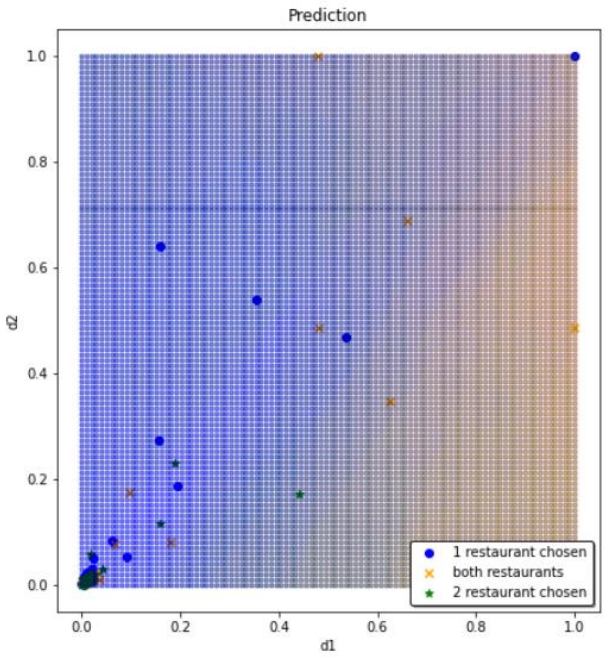


Prediction on new data

```
plt.figure(figsize=(16,8))
plt.subplot(1, 2, 1)

r1_grid = np.linspace(X['r1'].min(), X['r1'].max(), 100)
r2_grid = np.linspace(X['r2'].min(), X['r2'].max(), 100)
d1_grid = np.linspace(X['d1'].min(), X['d1'].max(), 100)
d2_grid = np.linspace(X['d2'].min(), X['d2'].max(), 100)
r1, r2 = np.meshgrid(r1_grid, r2_grid)
r1 = r1.reshape(np.prod(r1.shape))
r2 = r2.reshape(np.prod(r2.shape))
d1, d2 = np.meshgrid(d1_grid, d2_grid)
d1 = d1.reshape(np.prod(d1.shape))
d2 = d2.reshape(np.prod(d2.shape))
xxx = np.stack([np.ones(r1.shape), r1, r2, d1, d2])


for r in np.arange(0, w_trace.shape[0], 50):  # every 50th step in MC
    M = np.dot(w_trace[r,:,:], xxx)
    t_predict = np.argmax(M, axis=0)
    t_predict.shape
    for k in range(3):
        plt.plot(r1[t_predict==k], r2[t_predict==k],
                 marker='.', color=color_name[k], linestyle='none', alpha=0.01);
for k in range(3):
    plt.scatter(X['r1'][Y==k], X['r2'][Y==k],
                marker=markers[k], label=label_name[k], color=color_name[k])

plt.legend(loc='lower right', shadow=True, edgecolor='k', frameon='True')
plt.xlabel("r1")
plt.ylabel("r2")
plt.title('Prediction')

plt.subplot(1, 2, 2)

for r in np.arange(0, w_trace.shape[0], 50):  # every 50th step in MC
    M = np.dot(w_trace[r,:,:], xxx)
```

```
    t_predict = np.argmax(M, axis=0)
    t_predict.shape
    for k in range(3):
        plt.plot(d1[t_predict==k], d2[t_predict==k],
                 marker='.', color=color_name[k], linestyle='none', alpha=0.01);
for k in range(3):
    plt.scatter(X['d1'][Y==k], X['d2'][Y==k],
                marker=markers[k], label=label_name[k], color=color_name[k])

plt.legend(loc='lower right', shadow=True, edgecolor='k', frameon='True')
plt.xlabel("d1")
plt.ylabel("d2")
plt.title('Prediction');
```






```
import statistics
from collections import Counter

xxx = np.stack([np.ones([len(Y),]), X.r1, X.r2, X.d1, X.d2])
preds = []
for r in np.arange(0, w_trace.shape[0]):
    M = np.dot(w_trace[r,:,:], xxx)
    t_predict = np.argmax(M, axis=0)
    preds.append(t_predict)

probs = list()
for i in zip(*preds):
    probs.append([Counter(i)[0]/2000,Counter(i)[1]/2000,Counter(i)[2]/2000])

probs[1:10]
```

```
[[0.8775, 0.0, 0.1225],
 [0.2185, 0.0, 0.7815],
 [0.8865, 0.0, 0.1135],
 [0.829, 0.0, 0.171],
 [0.7645, 0.0, 0.2355],
 [1.0, 0.0, 0.0],
 [0.9665, 0.0, 0.0335],
 [0.0205, 0.888, 0.0915],
 [0.8875, 0.0, 0.1125]]
```

```
mcmc_logloss = log_loss(Y,probs)
print("Logloss on validation sample : {}".format(mcmc_logloss))
```

```
Logloss on validation sample : 7.385086489854537
```

```
preds = [statistics.mode(i) for i in zip(*preds)]
mcmc_acc = accuracy_score(Y, preds)
print(f'Fraction of correct predictions: {mcmc_acc}')
```

```
Fraction of correct predictions: 0.52
```

Multinomial Logistic Regression

Мультиномиальная логистическая регрессия — это метод классификации, который обобщает логистическую регрессию на многоклассовые задачи, т.е. с более чем двумя возможными дискретными результатами.

$$f(k,i) = \beta_k \mathbf{x}_i$$

где β представляет собой набор коэффициентов регрессии, связанных с результатом k, а xi (вектор строк) представляет собой набор объясняющих переменных, связанных с наблюдением i.

```
[ ]  logreg = LogisticRegression(fit_intercept=True, intercept_scaling=5,
                                 solver='lbfgs', multi_class='multinomial')
     logreg.fit(x,y)
     probs = logreg.predict_proba(X)
     logreg_logloss = log_loss(Y,probs)
     print("Logloss on validation sample : {}".format(logreg_logloss))
```

```
Logloss on validation sample : 0.9973946652435682
```

```
[ ]  preds = logreg.predict(X)
     logreg_acc = accuracy_score(Y,preds)
     print("Fraction of correct predictions: {}".format(logreg_acc))
```

```
Fraction of correct predictions: 0.52
```

LightGDM

Light Gradient Boosting Machine — это структура повышения градиента, основанная на деревьях решений для повышения эффективности модели и сокращения использования памяти.

```
[ ]  from lightgbm import LGBMClassifier

     lgb = LGBMClassifier(random_state=42, num_class = 3)
     lgb.fit(x, y)
     preds = lgb.predict_proba(X)
     LGB_score = log_loss(Y,preds)
     print("Logloss on validation sample : {}".format(LGB_score))
```

```
     Logloss on validation sample : 1.3201774903698382
```

```
from sklearn.model_selection import StratifiedKFold
import lightgbm as lgb

folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
lgb_output = np.zeros((len(Y), 3))
oof_lgb = np.zeros((len(x), 3))
score = 0

params = {
    'objective': 'multiclass',   'num_class': 3,   'verbosity': 1
}

for fold_n, (train_index, valid_index) in enumerate(folds.split(x,y)):
    print('Fold', fold_n)
    X_train, X_valid = x.iloc[train_index], x.iloc[valid_index]
    y_train, y_valid = y.iloc[train_index].astype(int), y.iloc[valid_index].astype(int)

    train_data = lgb.Dataset(X_train, label=y_train)
    valid_data = lgb.Dataset(X_valid, label=y_valid)

    lgb_model = lgb.train(params, train_set = train_data,num_boost_round=30000,
                          valid_sets = [train_data, valid_data],verbose_eval=300,
                          early_stopping_rounds = 300)

    y_pred_valid = lgb_model.predict(X_valid, num_iteration=lgb_model.best_iteration)
    oof_lgb[valid_index] = y_pred_valid
    score += log_loss(y_valid, y_pred_valid)

    lgb_output += lgb_model.predict(X, num_iteration=lgb_model.best_iteration)/5
```

```
Fold 0
Training until validation scores don't improve for 300 rounds.
[300]   training's multi_logloss: 0.0699493     valid_1's multi_logloss: 1.67579
Early stopping, best iteration is:
[17]    training's multi_logloss: 0.622202      valid_1's multi_logloss: 0.849934
Fold 1
Training until validation scores don't improve for 300 rounds.
[300]   training's multi_logloss: 0.0690975     valid_1's multi_logloss: 1.63206
Early stopping, best iteration is:
[21]    training's multi_logloss: 0.580292      valid_1's multi_logloss: 0.870407
Fold 2
Training until validation scores don't improve for 300 rounds.
[300]   training's multi_logloss: 0.073224      valid_1's multi_logloss: 1.74574
Early stopping, best iteration is:
[14]    training's multi_logloss: 0.650263      valid_1's multi_logloss: 0.877965
Fold 3
Training until validation scores don't improve for 300 rounds.
[300]   training's multi_logloss: 0.0725368     valid_1's multi_logloss: 1.37075
Early stopping, best iteration is:
[20]    training's multi_logloss: 0.600419      valid_1's multi_logloss: 0.807533
Fold 4
Training until validation scores don't improve for 300 rounds.
[300]   training's multi_logloss: 0.0833517     valid_1's multi_logloss: 1.56998
Early stopping, best iteration is:
[20]    training's multi_logloss: 0.595247      valid_1's multi_logloss: 0.852516
```

```
lgb_output[1:10]
```

```
array([[0.67155485, 0.14342551, 0.18501964],
       [0.64733638, 0.0532892 , 0.29937442],
       [0.41773967, 0.17522122, 0.40703911],
       [0.46131646, 0.20573568, 0.33294786],
       [0.37844817, 0.22649953, 0.3950523 ],
       [0.49652868, 0.08144385, 0.42202747],
       [0.27649099, 0.0730107 , 0.65049831],
       [0.33590518, 0.38252562, 0.2815692 ],
       [0.40084757, 0.21953539, 0.37961704]])
```

```
LGB_logloss = log_loss(Y,lgb_output)
print("Logloss on validation sample : {}".format(LGB_logloss))
```

```
Logloss on validation sample : 1.015155316210143
```

```
LGB_acc = accuracy_score(Y, [i.argmax() for i in lgb_output])
print("Fraction of correct predictions: {}".format(LGB_acc))
```

```
Fraction of correct predictions: 0.54
```

Результаты и вывод

```
from tabulate import tabulate

table = [
    ['Model\Metrics', 'logloss', 'accuracy'],
    ['MCMC', '%.2f' % mcmc_logloss, '%.2f' % mcmc_acc],
    ['Logistic regression', '%.2f' % logreg_logloss , '%.2f' % logreg_acc],
    ['LightGBM', '%.2f' % LGB_logloss, '%.2f' % LGB_acc]
]
print(tabulate(table, tablefmt='grid'))
```

```
+---------------------+---------+----------+
| Model\Metrics       | logloss | accuracy |
+---------------------+---------+----------+
| MCMC                | 7.39    | 0.52     |
+---------------------+---------+----------+
| Logistic regression | 1.00    | 0.52     |
+---------------------+---------+----------+
| LightGBM            | 1.02    | 0.54     |
+---------------------+---------+----------+
```

Несмотря на то, что логистическая регрессия и LightGBM имеют показатели логарифмических потерь намного меньше, чем логарифмические потери по методам Монте-Карло с цепями, показатели точности всех рассматриваемых моделей незначительно отличаются