Лабораторная работа №3

«КЛАССИФИКАЦИЯ В МАШИННОМ ОБУЧЕНИИ. ПОСТРОЕНИЕ МОДЕЛИ КЛАССИФИКАЦИЙ»

В наборе данных содержаться сведения об ингредиентах, широко используемых в кухнях Азии и Индии. Необходимо предсказать тип кухни на основе ингредиентов.

1. Загрузка библиотек

```
pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (from imblearn) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (3.4.0)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from imblearn.over_sampling import SMOTE
```

2. Загрузка данных

```
cuisines_df = pd.read_csv("Primer.csv")
cuisines_df.head()
```

	Unnamed: 0	cuisine	almond	angelica	anise	anise_seed	apple	apple_brandy	apricot	armagnac	...	whiskey	white_bread
0	65	indian	0	0	0	0	0	0	0	0	...	0	0
1	66	indian	1	0	0	0	0	0	0	0	...	0	0
2	67	indian	0	0	0	0	0	0	0	0	...	0	0
3	68	indian	0	0	0	0	0	0	0	0	...	0	0
4	69	indian	0	0	0	0	0	0	0	0	...	0	0

5 rows × 385 columns

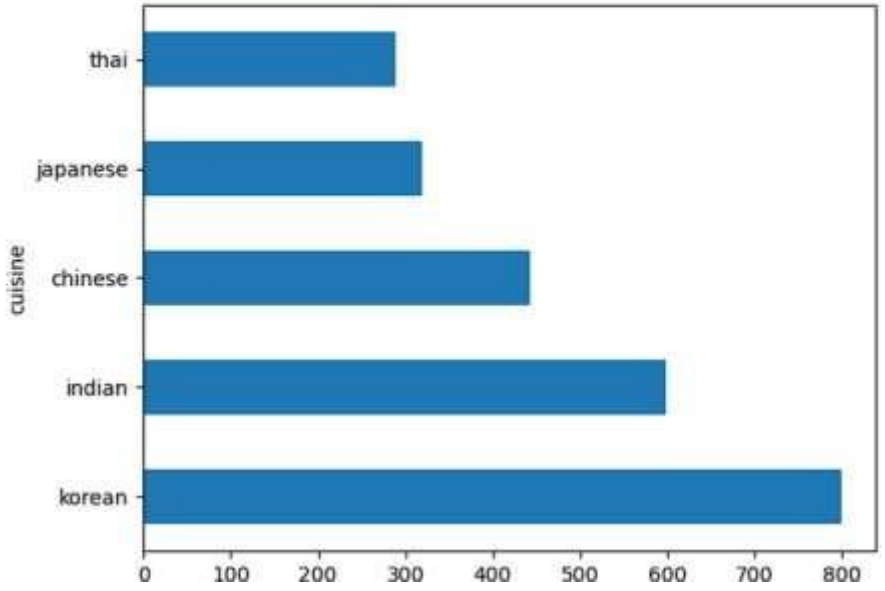
```
cuisines_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2448 entries, 0 to 2447
Columns: 385 entries, Unnamed: 0 to zucchini
dtypes: int64(384), object(1)
memory usage: 7.2+ MB
```

3. Распределение данных по кухням

```
cuisines_df.cuisine.value_counts().plot.barh()
```

```
<Axes: ylabel='cuisine'>
```



Существует конечное число кухонь, но распределение данных неравномерно. Это можете это исправить!

Узнаем, сколько данных доступно по каждой кухне:

```
thai_df = cuisines_df[(cuisines_df.cuisine == "thai")]
japanese_df = cuisines_df[(cuisines_df.cuisine == "japanese")]
chinese_df = cuisines_df[(cuisines_df.cuisine == "chinese")]
indian_df = cuisines_df[(cuisines_df.cuisine == "indian")]
korean_df = cuisines_df[(cuisines_df.cuisine == "korean")]

print(f'thai df: {thai_df.shape}')
print(f'japanese df: {japanese_df.shape}')
print(f'chinese df: {chinese_df.shape}')
print(f'indian df: {indian_df.shape}')
print(f'korean df: {korean_df.shape}')
```

```
thai df: (289, 385)
japanese df: (320, 385)
chinese df: (442, 385)
indian df: (598, 385)
korean df: (799, 385)
```

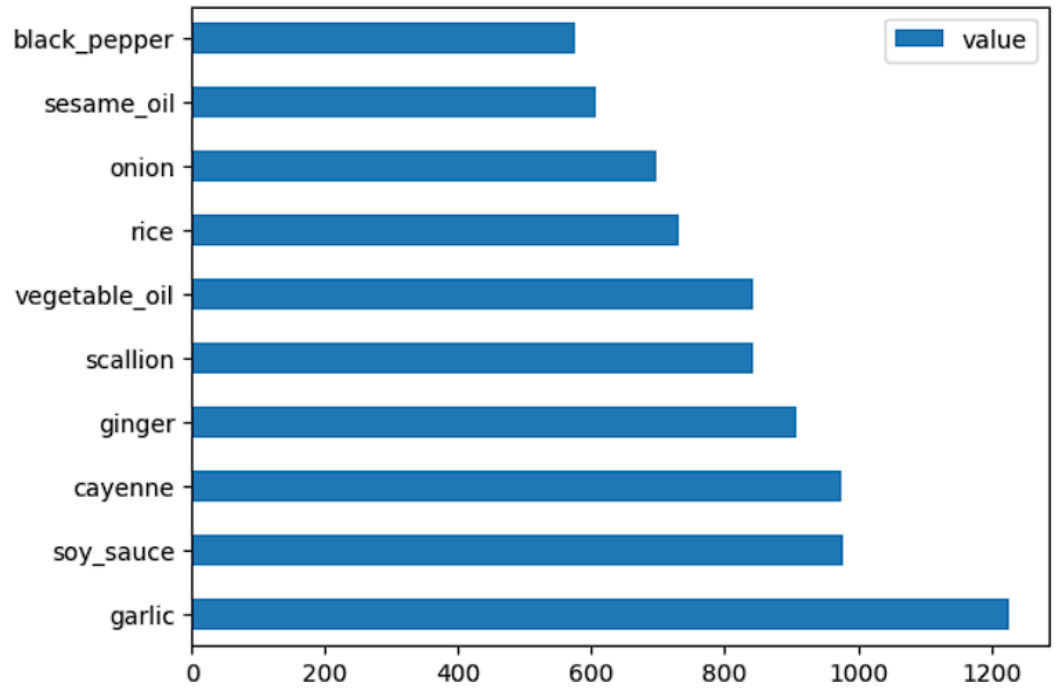
4. Знакомство с ингредиентами

Теперь можно углубиться в данные и узнать, какие ингредиенты типичны для каждой кухни. Для этого следует удалить повторяющиеся данные, которые создают путаницу между кухнями, поэтому сперва узнаем об этой проблеме. Создадим функцию **create_ingredient()** в Python для создания кадра данных ингредиентов. Эта функция начнет с удаления бесполезного столбца и сортиров- ки ингредиентов по их количеству:

```
def create_ingredient_df(df):
    ingredient_df = cuisines_df.T.drop(['cuisine','Unnamed: 0']).sum(axis=1).to_frame('value')
    ingredient_df = ingredient_df[(ingredient_df.T != 0).any()]
    ingredient_df = ingredient_df.sort_values(by='value', ascending=False,
    inplace=False)
    return ingredient_df
```

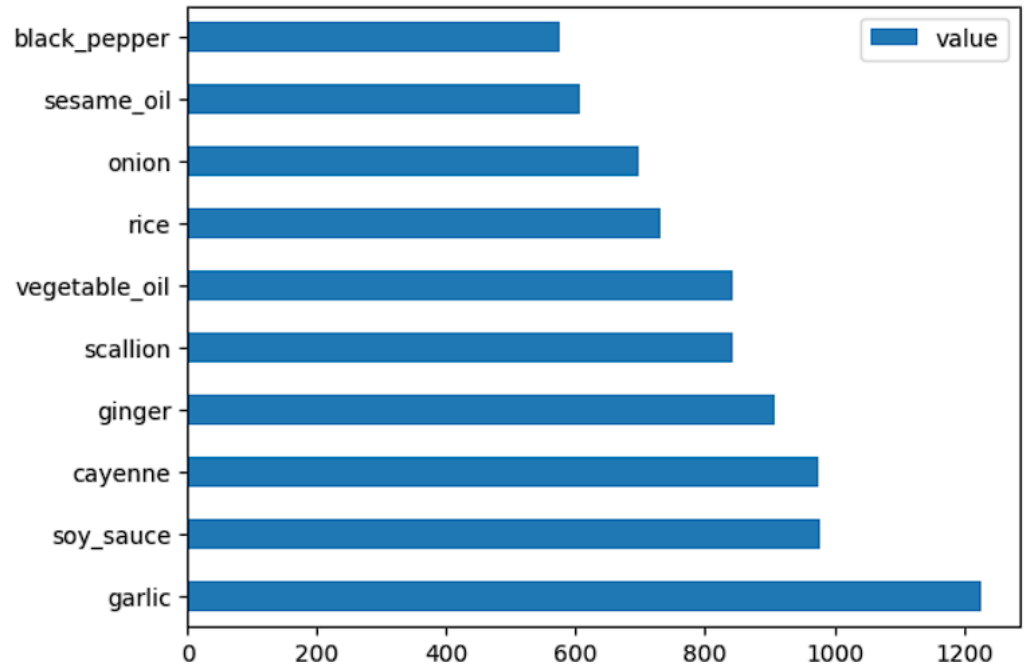
```
thai_ingredient_df = create_ingredient_df(thai_df)
thai_ingredient_df.head(10).plot.barh()
```

```
<Axes: >
```



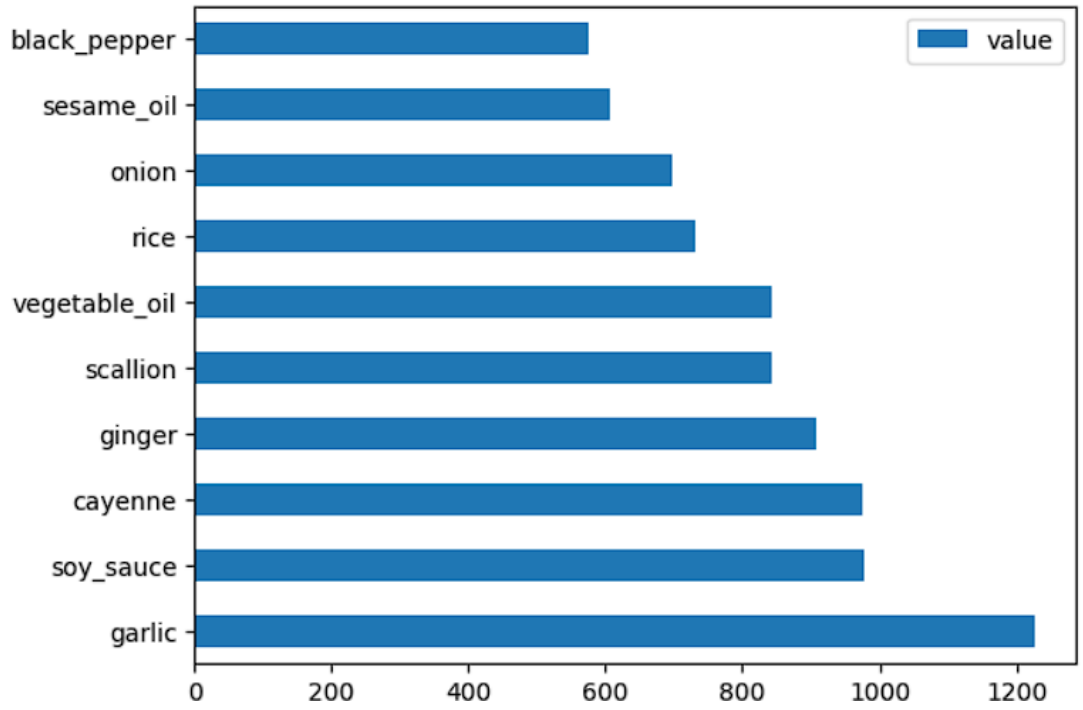

```
japanese_ingredient_df = create_ingredient_df(japanese_df)
japanese_ingredient_df.head(10).plot.barh()
```

```
<Axes: >
```



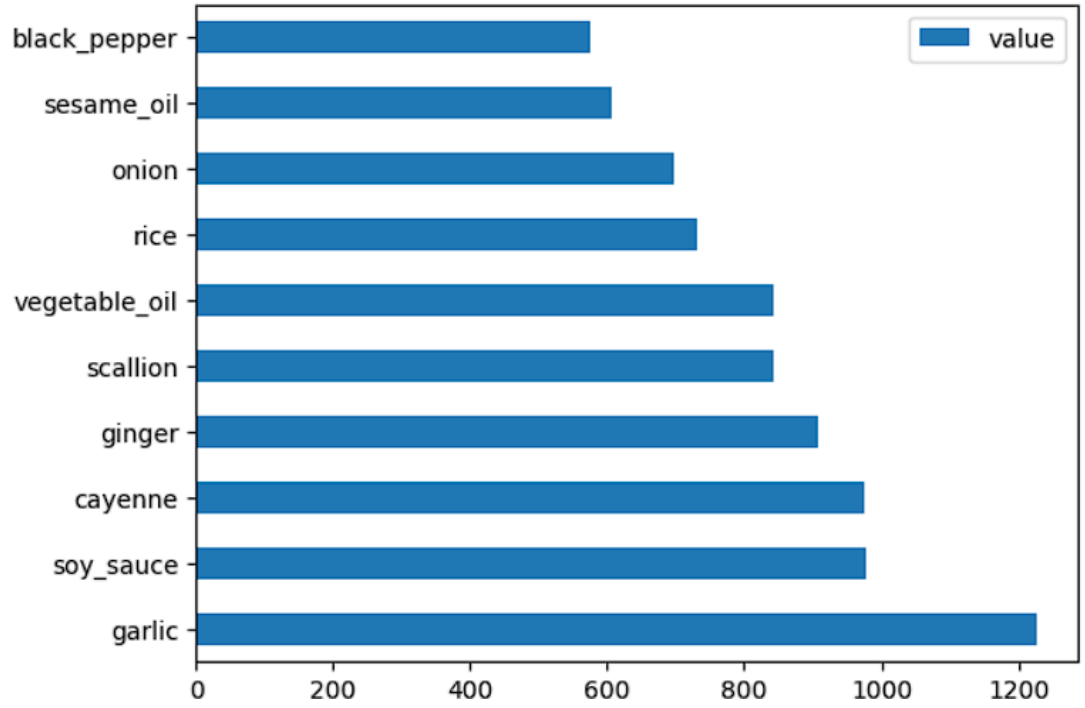
```
chinese_ingredient_df = create_ingredient_df(chinese_df)
chinese_ingredient_df.head(10).plot.barh()
```

```
<Axes: >
```



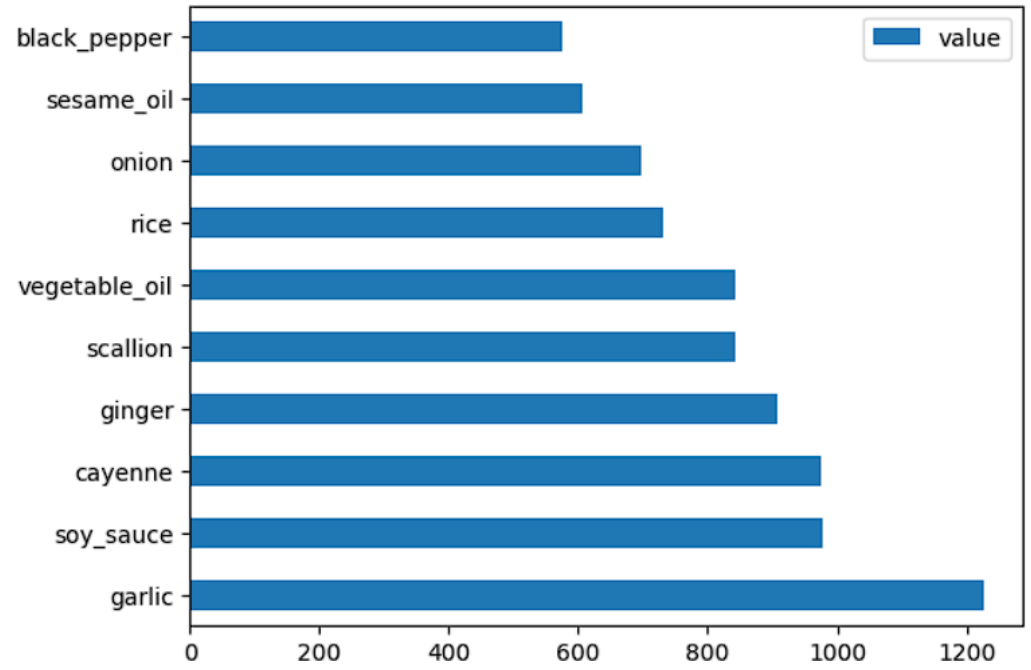

```
indian_ingredient_df = create_ingredient_df(indian_df)
indian_ingredient_df.head(10).plot.barh()
```

```
<Axes: >
```



```
korean_ingredient_df = create_ingredient_df(korean_df)
korean_ingredient_df.head(10).plot.barh()
```

```
<Axes: >
```



Теперь отбросьте наиболее распространенные ингредиенты, которые создают путаницу между разными кухнями, вызвав drop():

Все любят рис, чеснок и имбирь!

```
feature_df= cuisines_df.drop(['cuisine','Unnamed: 0','rice','garlic','ginger'], axis=1)
labels_df = cuisines_df.cuisine
feature_df.head()
```

	almond	angelica	anise	anise_seed	apple	apple_brandy	apricot	armagnac	artemisia	artichoke	...	whiskey	white_bread	white_wine
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 380 columns

5. Сбалансируем набор данных

Теперь, когда вы очистили данные, используйте **SMOTE** — «Технику синтетической избыточной выборки меньшинства» — чтобы сбалансировать их. **Call fit_resample()** - эта стратегия генерирует новые выборки путем интерполяции.

```
oversample = SMOTE()
transformed_feature_df, transformed_label_df = oversample.fit_resample(feature_df, labels_df)
```

Сбалансировав данные, вы получите лучшие результаты при их классификации. Подумай-

те о бинарной классификации. Если большая часть ваших данных относится к одному классу, модель машинного обучения будет чаще прогнозировать этот класс просто потому, что для него больше данных. Балансировка данных учитывает любые искаженные данные и помогает устранить этот дисбаланс.

```
print(f'new label count: {transformed_label_df.value_counts()}')
print(f'old label count: {cuisines_df.cuisine.value_counts()}')
```

```
new label count: cuisine
indian      799
thai        799
chinese     799
japanese    799
korean      799
Name: count, dtype: int64
old label count: cuisine
korean      799
indian      598
chinese     442
japanese    320
thai        289
Name: count, dtype: int64
```

6. Сохранение и загрузка сбалансированного набора данных

Сохранить сбалансированные данные, включая метки и объекты, в новый фрейм данных, который можно экспортировать в файл:

```
transformed_df = pd.concat([transformed_label_df,transformed_feature_df],axis=1, join='outer')
```

```
transformed_df.head()
transformed_df.info()
transformed_df.to_csv("cleaned_primer.csv")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3995 entries, 0 to 3994
Columns: 381 entries, cuisine to zucchini
dtypes: int64(380), object(1)
memory usage: 11.6+ MB
```

```
df = pd.read_csv("cleaned_primer.csv")
```

```
cuisines_label_df = cuisines_df['cuisine']
cuisines_label_df.head()
```

```
0    indian
1    indian
2    indian
3    indian
4    indian
Name: cuisine, dtype: object
```

7. Построение модели классификаций

```
cuisines_feature_df = cuisines_df.drop(['Unnamed: 0', 'cuisine'], axis=1)
cuisines_feature_df.head()
```

	almond	angelica	anise	anise_seed	apple	apple_brandy	apricot	armagnac	artemisia	artichoke	...	whiskey	white_bread	white_wine
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 383 columns

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score,precision_score,confusion_matrix,classification_report, precision_recall_curve
import numpy as np
```

```
X_train, X_test, y_train, y_test = train_test_split(cuisines_feature_df, cuisines_label_df, test_size=0.3)
```

```
C = 10
# построение модели классификаторов
classifiers = {
    'Linear SVC': SVC(kernel='linear', C=C, probability=True,random_state=0),
    'KNN classifier': KNeighborsClassifier(C),
    'SVC': SVC(),
    'RFST': RandomForestClassifier(n_estimators=100),
    'ADA': AdaBoostClassifier(n_estimators=100)
}
```

```
n_classifiers = len(classifiers)

for index, (name, classifier) in enumerate(classifiers.items()):
    classifier.fit(X_train, np.ravel(y_train))

    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy (train) for %s: %0.1f%% " % (name, accuracy * 100))
    print(classification_report(y_test,y_pred))
```