



УНИВЕРСИТЕТ ИТМО

Практика 7. Разработка приложения на основе машинного обучения от идеи до конечного продукта

Михаил А. Каканов¹ Олег А. Евстафьев¹

¹Факультет систем управления и робототехники, Университет ИТМО
{makakanov, oaevstafev}@itmo.ru

Октябрь 2021

Курс «Прикладной искусственный интеллект»

1. Постановка задачи
2. Обработка данных
3. Разработка модели
4. Интерфейс
5. Развертывание

1. Постановка задачи

2. Обработка данных

3. Разработка модели

4. Интерфейс

5. Развертывание

Цель

Спрогнозировать вероятность становления диабетиком.

- ▶ Набор данных с [Kaggle](#) собран в Национальном институте диабета и болезней пищеварительной системы и почек.
- ▶ Этот набор данных невелик: 9 признаков и 768 наблюдений, но его вполне достаточно для решения задачи.
- ▶ [Источник материалов](#)

1. **Pregnancies: Число беременностей**
2. **Glucose:** Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. **BloodPressure:** Диастолическое артериальное давление (мм рт. ст.).
4. **SkinThickness:** Толщина кожной пластины трицепса (мм).
5. **Insulin:** 2-часовой сывороточный инсулин (мю Ед/мл).
6. **BMI:** Индекс массы тела (вес в кг/(рост в м)²)
7. **DiabetesPedigreeFunction:** Функция родословной диабета
8. **Age:** Возраст (годы)
9. **Outcome:** Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мю Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мю Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мю Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мЮ Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мю Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мю Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мЮ Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

1. Pregnancies: Число беременностей
2. Glucose: Концентрация глюкозы в плазме крови через 2 часа при проведении перорального теста на толерантность к глюкозе.
3. BloodPressure: Диастолическое артериальное давление (мм рт. ст.).
4. SkinThickness: Толщина кожной пластины трицепса (мм).
5. Insulin: 2-часовой сывороточный инсулин (мю Ед/мл).
6. BMI: Индекс массы тела (вес в кг/(рост в м)²)
7. DiabetesPedigreeFunction: Функция родословной диабета
8. Age: Возраст (годы)
9. Outcome: Переменная класса (0 или 1) 268 из 768 - 1, остальные - 0

Все переменные либо известны сами по себе, либо доступны в простом анализе крови, а "результат"(диабет/не диабет) - это то, что нам нужно предсказать.

- ▶ Мы будем исследовать различные характеристики и выполнять различные методы предварительной обработки.
- ▶ Опробуем различные алгоритмы машинного обучения, такие как логистическая регрессия, SVM, случайный лес, Gradient Boosting и, наконец, мы также исследуем нейронные сети.
- ▶ Как только мы получим наилучшую модель, мы сохраним нашу модель с помощью Pickle.
- ▶ Разработаем приложение для прогнозирования диабета с помощью веб-фреймворка Flask.
- ▶ Затем развернем его с помощью Heroku.

Все переменные либо известны сами по себе, либо доступны в простом анализе крови, а "результат"(диабет/не диабет) - это то, что нам нужно предсказать.

- ▶ Мы будем исследовать различные характеристики и выполнять различные методы предварительной обработки.
- ▶ Опробуем различные алгоритмы машинного обучения, такие как логистическая регрессия, SVM, случайный лес, Gradient Boosting и, наконец, мы также исследуем нейронные сети.
- ▶ Как только мы получим наилучшую модель, мы сохраним нашу модель с помощью Pickle.
- ▶ Разработаем приложение для прогнозирования диабета с помощью веб-фреймворка Flask.
- ▶ Затем развернем его с помощью Heroku.

Все переменные либо известны сами по себе, либо доступны в простом анализе крови, а "результат"(диабет/не диабет) - это то, что нам нужно предсказать.

- ▶ Мы будем исследовать различные характеристики и выполнять различные методы предварительной обработки.
- ▶ Опробуем различные алгоритмы машинного обучения, такие как логистическая регрессия, SVM, случайный лес, Gradient Boosting и, наконец, мы также исследуем нейронные сети.
- ▶ Как только мы получим наилучшую модель, мы сохраним нашу модель с помощью Pickle.
- ▶ Разработаем приложение для прогнозирования диабета с помощью веб-фреймворка Flask.
- ▶ Затем развернем его с помощью Heroku.

Все переменные либо известны сами по себе, либо доступны в простом анализе крови, а "результат"(диабет/не диабет) - это то, что нам нужно предсказать.

- ▶ Мы будем исследовать различные характеристики и выполнять различные методы предварительной обработки.
- ▶ Опробуем различные алгоритмы машинного обучения, такие как логистическая регрессия, SVM, случайный лес, Gradient Boosting и, наконец, мы также исследуем нейронные сети.
- ▶ Как только мы получим наилучшую модель, мы сохраним нашу модель с помощью Pickle.
- ▶ Разработаем приложение для прогнозирования диабета с помощью веб-фреймворка Flask.
- ▶ Затем развернем его с помощью Heroku.

Все переменные либо известны сами по себе, либо доступны в простом анализе крови, а "результат"(диабет/не диабет) - это то, что нам нужно предсказать.

- ▶ Мы будем исследовать различные характеристики и выполнять различные методы предварительной обработки.
- ▶ Опробуем различные алгоритмы машинного обучения, такие как логистическая регрессия, SVM, случайный лес, Gradient Boosting и, наконец, мы также исследуем нейронные сети.
- ▶ Как только мы получим наилучшую модель, мы сохраним нашу модель с помощью Pickle.
- ▶ Разработаем приложение для прогнозирования диабета с помощью веб-фреймворка Flask.
- ▶ Затем развернем его с помощью Heroku.

1. Постановка задачи

2. Обработка данных

3. Разработка модели

4. Интерфейс

5. Развертывание

Импортируем необходимые библиотеки:

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 df.head()
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.shape
```

```
(768, 9)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 768 entries, 0 to 767  
Data columns (total 9 columns):  
#   Column                Non-Null Count  Dtype    
---  ---                  
0   Pregnancies            768 non-null   int64    
1   Glucose                768 non-null   int64    
2   BloodPressure          768 non-null   int64    
3   SkinThickness          768 non-null   int64    
4   Insulin                768 non-null   int64    
5   BMI                    768 non-null   float64   
6   DiabetesPedigreeFunction 768 non-null   float64   
7   Age                    768 non-null   int64    
8   Outcome                768 non-null   int64    
dtypes: float64(2), int64(7)  
memory usage: 54.1 KB
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Видим, что BMI/Skin Thickness/Age имеют минимальное значение 0, что не имеет смысла.

Посмотрим сколько нулей в данных по признакам:

```
print('Total zero Glucose values: ' + str(df[df['Glucose']==0].shape[0]))
print('Total zero BloodPressure values: ' + str(df[df['BloodPressure']==0].shape[0]))
print('Total zero SkinThickness values: ' + str(df[df['SkinThickness']==0].shape[0]))
print('Total zero Insulin values: ' + str(df[df['Insulin']==0].shape[0]))
print('Total zero BMI values: ' + str(df[df['BMI']==0].shape[0]))
print('Total zero DiabetesPedigreeFunction values: ' + str(df[df['DiabetesPedigreeFunction']==0].shape[0]))
print('Total zero Age values: ' + str(df[df['Age']==0].shape[0]))
```

```
Total zero Glucose values: 5
Total zero BloodPressure values: 35
Total zero SkinThickness values: 227
Total zero Insulin values: 374
Total zero BMI values: 11
Total zero DiabetesPedigreeFunction values: 0
Total zero Age values: 0
```

Заменим нули на Null, чтобы затем провести интерполяцию и заполнить пропущенные значения:

```
def replace_zero(df):  
    df_nan = df.copy(deep=True)  
    cols = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]  
    df_nan[cols] = df_nan[cols].replace({0:np.nan})  
    return df_nan  
  
df_nan=replace_zero(df)
```


Посчитаем медиану по категориям:

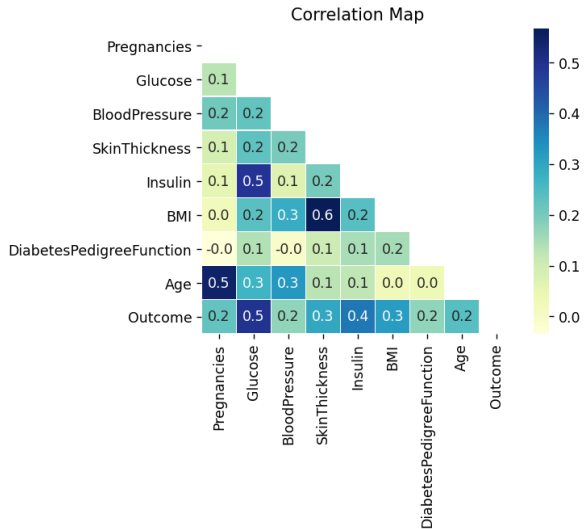
```
#finds the median of different attribute based on Outcome  
def find_median(df, col):  
  
    df_nondiab = df[df['Outcome']==0].reset_index(drop = True)  
    df_diab = df[df['Outcome']==1].reset_index(drop = True)  
    return(df_nondiab[col].median(), df_diab[col].median())
```

Интерполируем данные по медиане:

```
def replace_null(df, var):  
  
    median_tuple = find_median(df, var)  
    var_0 = median_tuple[0]  
    var_1 = median_tuple[1]  
  
    df.loc[(df['Outcome'] == 0) & (df[var].isnull()), var] = var_0  
    df.loc[(df['Outcome'] == 1) & (df[var].isnull()), var] = var_1  
  
    return df[var].isnull().sum()
```

Посмотрим за зависимость признаков друг от друга по карте корреляции и гистограмме:

```
: plt.figure(dpi = 125,figsize= (5,4))
mask = np.triu(df_nan.corr())
sns.heatmap(df_nan.corr(),mask = mask, fmt = ".1f",annot=True,lw=0.1,cmap = 'YlGnBu')
plt.title('Correlation Map')
plt.show()
```



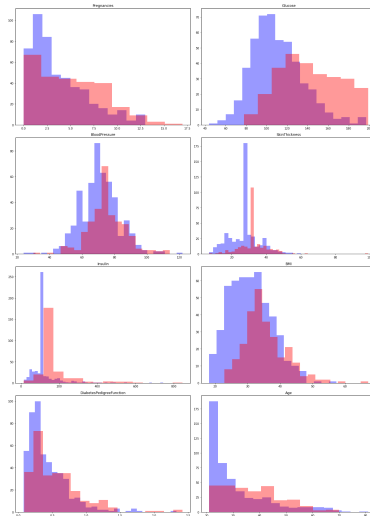
```
fig, ax = plt.subplots(ncols=2, nrows=4, figsize=(18, 25))

target_0 = df.loc[df['Outcome'] == 0]
target_1 = df.loc[df['Outcome'] == 1]

for i, subplot in zip(range(8), ax.flatten()):

    sns.distplot(target_0[[column_list[i]]], hist=True, color = 'blue', ax=subplot, norm_hist = False, kde = False)
    sns.distplot(target_1[[column_list[i]]], hist=True, color = 'red', ax=subplot, norm_hist = False, kde = False)
    subplot.title.set_text(column_list[i])
    plt.tight_layout()

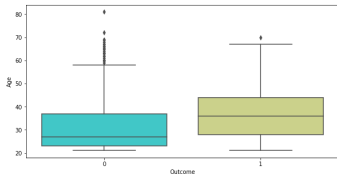
plt.show()
```



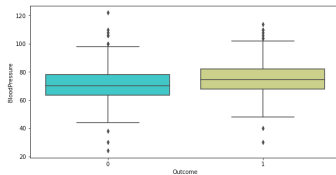
Мы видим, что по большинству признаков распределение для людей с диабетом (красная часть) смещено вправо по сравнению с распределением недиабетиков (синяя часть). Это, по сути, говорит нам о том, что человек с диабетом с большей вероятностью будет пожилым человеком с более высоким ИМТ, толщиной кожи и уровнем глюкозы.

Далее мы построим график для каждого из этих признаков, чтобы четко увидеть разницу в распределении каждого из признаков для обоих исходов (диабетического и недиабетического).

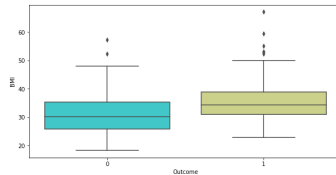
Age vs Diabetes



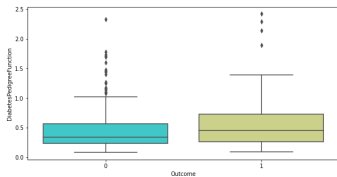
BloodPressure vs Diabetes



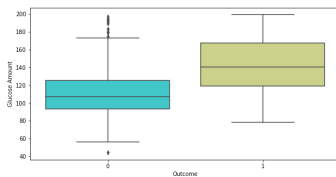
BMI vs Diabetes



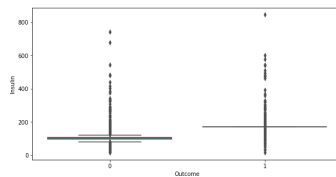
DiabetesPedigreeFunction vs Diabetes



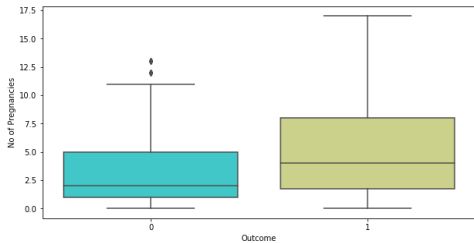
Glucose vs Diabetes



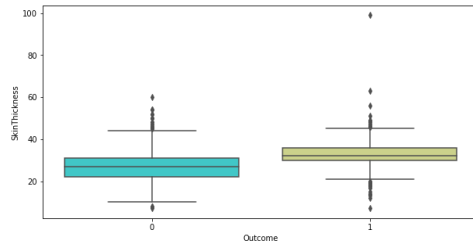
Insulin vs Diabetes



Pregnancies vs Diabetes



SkinThickness vs Diabetes



Далее визуализируем данные в двумерной плоскости методами PCA и t-SNE:

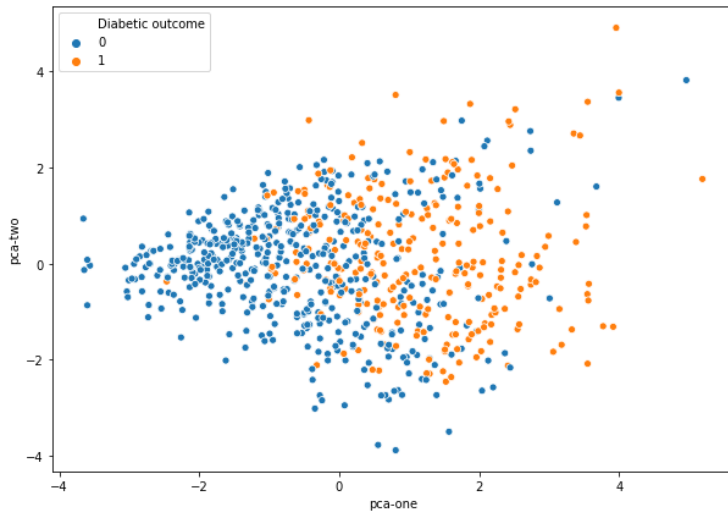
```
pca_df = pd.DataFrame(df['Outcome'])
```

```
#visualizing using PCA
```

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(pca_x)
pca_df['pca-one'] = pca_result[:,0]
pca_df['pca-two'] = pca_result[:,1]
```

```
pca_df
```

	Outcome	pca-one	pca-two
0	1	1.575769	-0.613465
1	0	-1.575930	0.079637
2	1	0.734548	-0.543459
3	0	-2.131111	0.267265



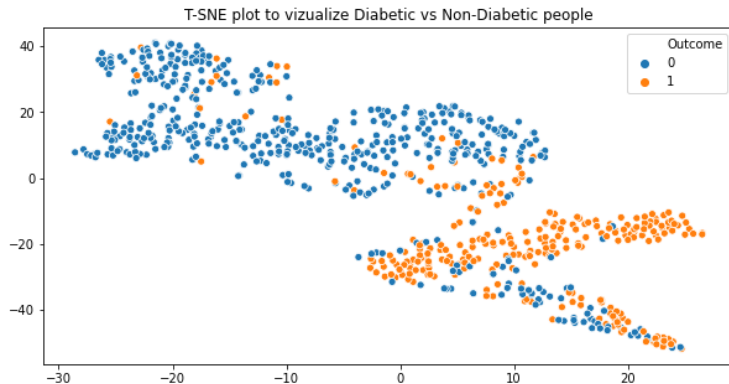
```
y = df_tsne['Outcome']  
df_tsne = df_tsne.drop(columns = 'Outcome')
```

```
df_tsne.head()
```

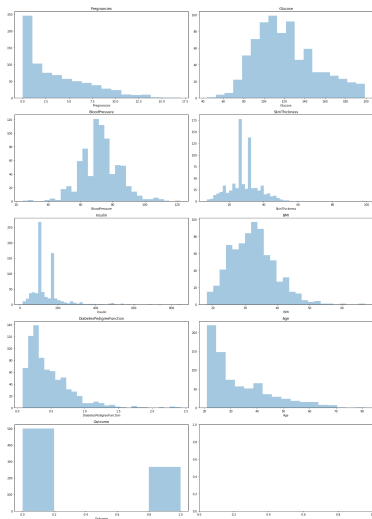
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33

```
from sklearn.manifold import TSNE  
model = TSNE(learning_rate=50)  
tsne_features = model.fit_transform(df_tsne)
```

```
plt.figure(figsize=(10, 5))  
plt.title('T-SNE plot to visualize Diabetic vs Non-Diabetic people')  
sns.scatterplot(x = tsne_features[:,0], y = tsne_features[:,1], hue = y)  
plt.show()
```



- ▶ Теперь, перед моделированием, мы должны масштабировать данные, поскольку все атрибуты находятся в разных масштабах.
 - ▶ Нормализация обычно означает изменение масштаба значений в диапазоне $[0, 1]$.
 - ▶ Стандартизация обычно означает изменение масштаба данных, чтобы среднее значение было равно 0, а стандартное отклонение - 1 (дисперсия единицы).
- ▶ Хорошей практикой является настройка скалера на обучающих данных, а затем использование его для преобразования тестовых данных. Это позволит избежать утечки данных в процессе тестирования модели. Кроме того, масштабирование целевых значений обычно не требуется.



- ▶ Только показатели глюкозы, артериального давления и ИМТ имеют гауссово распределение, где нормализация может иметь смысл, но поскольку нет жесткого и быстрого правила, мы попробуем три варианта и сравним их эффективность.
- ▶ Используем стандартизацию.

1. Постановка задачи

2. Обработка данных

3. Разработка модели

4. Интерфейс

5. Развертывание

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
std = scaler.fit(X_train)

#save
import pickle
pickle.dump(std, open('std.pkl', 'wb'))

X_train = std.transform(X_train)
X_test = std.transform(X_test)
X_train = pd.DataFrame(X_train, columns = columns)
X_test = pd.DataFrame(X_test, columns = columns)
X_train.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	-0.531978	-0.706880	-0.186385	2.489469	-0.956673	1.157236	0.631014	-0.693698
1	0.057083	-0.113681	-0.186385	-0.237913	-0.454062	1.742852	1.334750	-0.608986
2	1.824269	-0.673925	0.318498	2.053088	0.402033	0.044565	-0.937667	2.525347
3	-0.826509	-0.838703	-0.691267	-0.237913	-0.625281	0.088486	-0.571848	-1.032545
4	1.235207	-0.871658	-0.018090	-0.237913	-0.454062	0.615541	0.035783	2.017077

```
#Building Logistic Regression model on the Standardized variables  
  
from sklearn.linear_model import LogisticRegression  
lr_std = LogisticRegression(random_state = 20,penalty='l2')  
lr_std.fit(X_train, y_train)  
y_pred = lr_std.predict(X_test)  
print('Accuracy of logistic regression on test set with standardized features: {:.2f}'.format(lr_std.score(X_test, y_test)))
```

Accuracy of logistic regression on test set with standardized features: 0.83

```
: from sklearn.neighbors import KNeighborsClassifier
test_scores = []
train_scores = []
for i in range(5,15):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train, y_train)
    train_scores.append(neigh.score(X_train,y_train))
    test_scores.append(neigh.score(X_test,y_test))
```

```
: print('Max train_scores is ' + str(max(train_scores)*100) + ' for k = '+
      str(train_scores.index(max(train_scores))+5))
```

Max train_scores is 85.50488599348535 for k = 11

```
: print('Max test_scores is ' + str(max(test_scores)*100) + ' for k = '+
      str(test_scores.index(max(test_scores))+5))
# K=13 has generalized well for our data.
```

Max test_scores is 87.01298701298701 for k = 13

```
from sklearn.model_selection import GridSearchCV
clf = RandomForestClassifier(random_state = 20)

# use a full grid over all parameters
param_grid = {'n_estimators' : [50,100,150], "max_depth": [3, 5, 7],
              "max_features": [1, 3, 5, 7, 8]
              }

# run grid search
grid = GridSearchCV(clf, param_grid, cv=5,scoring='accuracy')

grid.fit(X_train, y_train)
print("Grid-Search with accuracy")
print("Best parameters:", grid.best_params_)
print("Best cross-validation score (accuracy): {:.3f}".format(grid.best_score_))
```

```
Grid-Search with accuracy
Best parameters: {'max_depth': 7, 'max_features': 3, 'n_estimators': 100}
Best cross-validation score (accuracy): 0.880
```

```
clf = GradientBoostingClassifier(random_state=20)
# use a full grid over all parameters
param_grid = {'n_estimators': [50,100,150], "max_depth": [2,5,7,8],
              "max_features": [2,3,5, 7], 'learning_rate': [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]}
# run grid search
grid = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
print("Grid-Search with accuracy")
print("Best parameters:", grid.best_params_)
print("Best cross-validation score (accuracy): {:.3f}".format(grid.best_score_))
```

Grid-Search with accuracy

Best parameters: {'learning_rate': 0.075, 'max_depth': 5, 'max_features': 2, 'n_estimators': 50}

Best cross-validation score (accuracy): 0.888

```
#Support Vector Machines
from sklearn import svm
svm_model = svm.SVC(probability=True).fit(X_train, y_train)
svm_pred=svm_model.predict(X_test)
svm_model.score(X_test, y_test)
# Almost 89% Accuracy
```

0.8831168831168831

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
def build_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(8, activation='relu', input_shape=[len(X_train.keys())]),
        #tf.keras.layers.Dense(4, activation='relu'),
        tf.keras.layers.Dense(4, activation='relu'),
        tf.keras.layers.Dense(2, activation='relu'),

        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    optimizer = tf.keras.optimizers.Adam(learning_rate=0.005, beta_1=0.9, beta_2=0.999, epsilon=1e-07)

    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

neural_model = build_model()

neural_model.summary()
```

```
neural_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 8)	72
dense_1 (Dense)	(None, 4)	36
dense_2 (Dense)	(None, 2)	10
dense_3 (Dense)	(None, 1)	3
=====		

Total params: 121

Trainable params: 121

Non-trainable params: 0

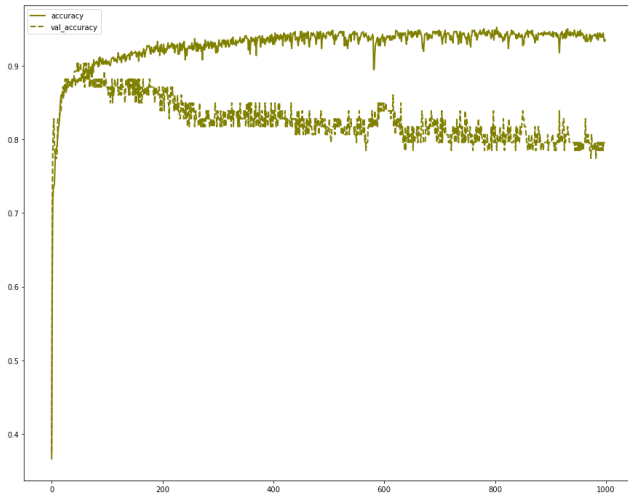

```
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
checkpoint_name = 'Weights\Weights-{epoch:03d}--{val_accuracy:.5f}.hdf5'  
checkpoint = ModelCheckpoint(checkpoint_name, monitor='val_loss', verbose = 1, save_best_only = False, mode = 'auto')  
callbacks_list = [checkpoint]
```

```
# Keeping EPOCHs high as dataset is small.
```

```
EPOCHS = 1000
```

```
neural_pred = neural_model.fit(X_train, y_train, epochs=EPOCHS, validation_split=0.15, verbose=2, callbacks = callbacks_list )
```



Точность:

- ▶ Логистическая регрессия - 83%
- ▶ К-ближайших соседей - 87%
- ▶ SVM - 88–89%
- ▶ Случайные деревья - 88–89%
- ▶ Gradient Boosting - 87–88%
- ▶ Нейронная сеть - 85%

Теперь, в качестве последнего шага, мы сохраним нашу SVM-модель для прогнозирования в файл .h5 или .bin с помощью библиотеки типа pickle.

```
import pickle
# Lets dump our SVM model
pickle.dump(svm_model, open('svm_model.pkl', 'wb'))
```

1. Постановка задачи
2. Обработка данных
3. Разработка модели
- 4. Интерфейс**
5. Развертывание

- ▶ Следующим шагом будет упаковка этой модели в веб-сервис, который, получив данные через POST-запрос, вернет вероятность диабетического прогноза в качестве ответа.
- ▶ Для этого мы будем использовать веб-фреймворк Flask, широко используемый легкий фреймворк для разработки веб-сервисов на языке Python.

Код ниже в файле app.py, по сути, устанавливает домашнюю страницу и предоставляет пользователю файл index.html:

```
1 #import relevant libraries for flask , html rendering and loading the #  
   ML model  
2 from flask import Flask , request , url_for , redirect , render_template  
3 import pickle  
4 import pandas as pd  
5  
6 app = Flask(__name__)  
7  
8 #loading the SVM model and the preprocessor  
9 model = pickle.load(open("svm_model.pkl", "rb"))  
10 std = pickle.load(open('std.pkl', 'rb'))  
11  
12  
13 #Index.html will be returned for the input
```

```
14 @app.route("/")
15 def hello_world():
16     return render_template("index.html")
17
18
19 #predict function, POST method to take in inputs
20 @app.route('/predict', methods=['POST', 'GET'])
21 def predict():
22     #take inputs for all the attributes through the HTML form
23     pregnancies = request.form[1]
24     glucose = request.form[2]
25     bloodpressure = request.form[3]
26     skinthickness = request.form[4]
27     insulin = request.form[5]
28     bmi = request.form[6]
29     diabetespedigreefunction = request.form[7]
30     age = request.form[8]
```



```
31
32     #form a dataframe with the input and run the preprocessor as used
    in the training
33     row_df = pd.DataFrame([pd.Series([pregnancies, glucose,
    bloodpressure, skinthickness, insulin, bmi, diabetespedigreefunction
    , age])])
34     row_df = pd.DataFrame(std.transform(row_df))
35
36     print(row_df)
37
38     #predict the probability and return the probability of being a
    diabetic
39     prediction=model.predict_proba(row_df)
40     output='{0:.{1}f}'.format(prediction[0][1], 2)
41     output_print = str(float(output)*100)+"%"
42
43     if float(output) > 0.5:
```

```
44         return render_template('(result.'html,pred='fYou have a chance of
having diabetes.\nProbability of you being a diabetic is {
output_print}.\nEat clean and exercise 'regularly)
45     else:
46         return render_template('(result.'html,pred='fCongratulations , you
are safe.\n Probability of you being a diabetic is {output_print}')
47
48
49 if __name__ == "__main__":
50     app.run(debug=True)
```

Отрывок из index.html:

```
1      <form action = '/predict ' method="post" class="col s12">
2      ...
3          <div class="input-field col s4">
4              <label for="first_name"><b>Pregnancies (0-15)</b></
label>
5              <br>
6              <input placeholder="No. of Pregnancies" name="1" id
="first_name" type="text" class="validate">
7              </div>
8      ...
9          <button type="submit" class="btn-large waves-effect
waves-light light-green">Submit and predict probability</button>
10     ...
11     </form>
```

```
(base) C:\Users\saket.garodia\Kaggle datasets\India diabetes prediction\diabetes>python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 231-534-753
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Are you worried that you might be Diabetic?

[Return to home](#)

Predict diabetes (using AI in the background)

Predict the probability that you may be diabetic

Pregnancies (0-15)

6

Glucose (40-250)

40

BloodPressure (20-140)

72

SkinThickness (5-80)

35

Insulin (0-1000)

10

BMI(10-100)

20

DiabetesPedigreeFunction (0-2.5)

0.627

Age (10-120)

50

SUBMIT AND PREDICT PROBABILITY

© Saket Garodia

saketgarodia1@gmail.com

1. Постановка задачи
2. Обработка данных
3. Разработка модели
4. Интерфейс
- 5. Развертывание**

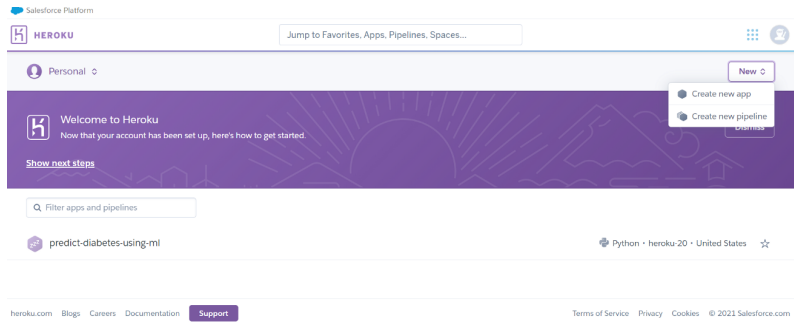
Воспользуемся Github, а затем развернем репозиторий Github в Heroku.

Структура проекта:


```
1 diabetes(root)
2 |___templates
3 |   |___index.html #main html page to enter the data
4 |   |___result.html #Page returned after pressing submit
5 |___static
6 |   |___css #code for the look and feel of the web app
7 |   |___js
8 |___app.py #main file with flask and prediction code
9 |___svm_model.pkl #model
10 |___std.pkl #preprocessor
11 |___requirements.txt #Library list with versions
12 |___Procfile
```

Procfile: Здесь содержится команда для запуска приложения на сервере.

```
1 web: gunicorn app:app
```




Salesforce Platform

 HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...



Create New App

App name




predict-diabetes-usingml is available

Choose a region



 United States 

Add to pipeline...



Create app

 **HEROKU**



Jump to Favorites, Apps, Pipelines, Spaces...

Create a new pipeline or choose an existing one and add this app to a stage in it.

Pipelines let you connect multiple apps together and **promote** code between them.
[Learn more](#)

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests.
[Learn more](#)

Choose a pipeline

 Heroku Git
Use Heroku CLI GitHub
Connect to GitHub Container Registry
Use Heroku CLI

Search for a repository to connect to

 garodisk

repo-name

Search

Diabetes Prediction

[Home](#)

Congratulations, you are safe. Probability of you being a diabetic is 13.0%

© Saket Garodia

saketgarodia1@gmail.com

