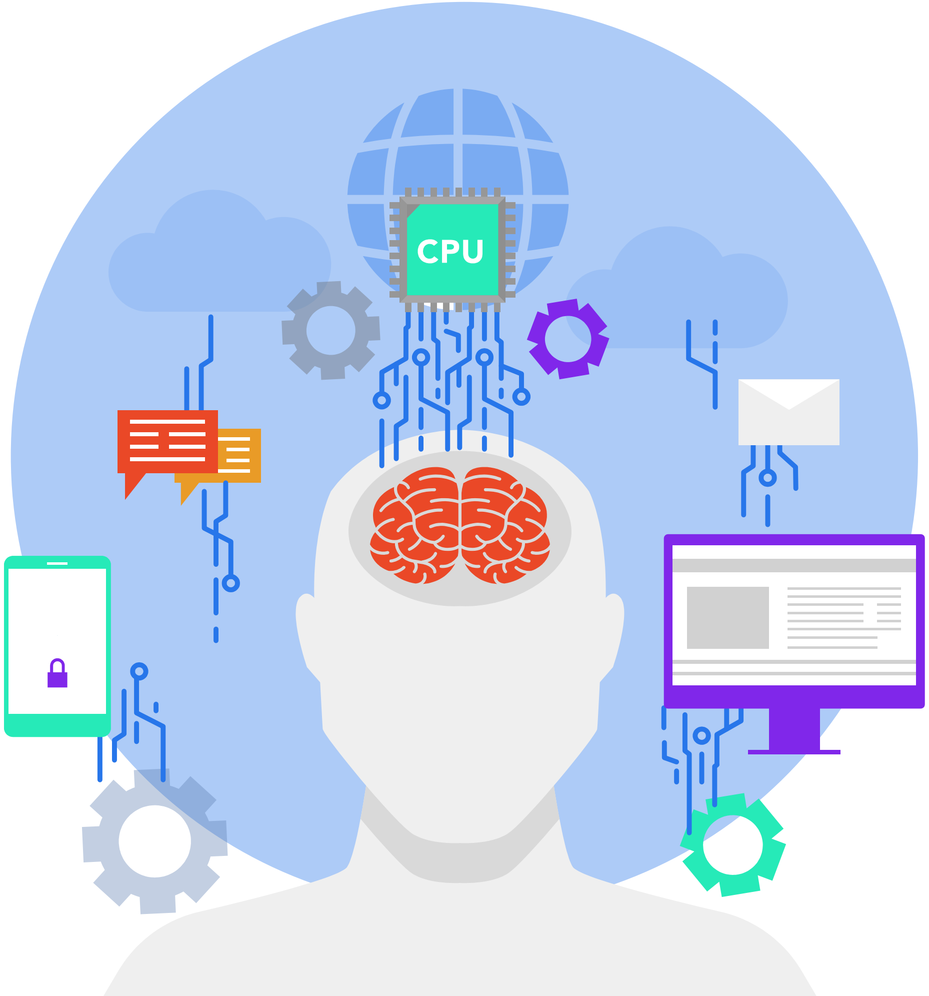
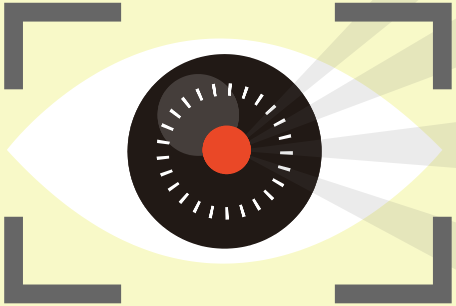


Команда “Провинция”

Этапы разработки

Разработка



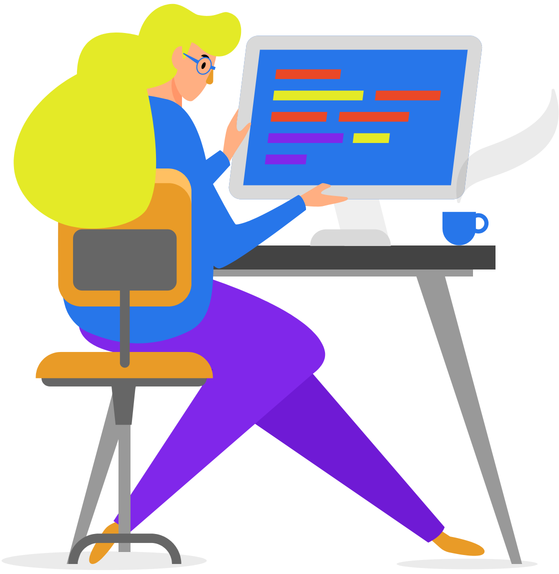
Парсинг — Предобработка данных для удобной работы

Векторизация — Перевод слов в вектор

Обучение — Обучаем модель на тестовых данных

Ранжирование — Сортировка подходящих резюме

Парсинг



01 Объединяем колонки с подходящими и не подходящими резюме

02 Выделяем нужные категории из резюме для обучения

03 Преобразовываем данные о каждом кандидате в одну строку

04 Убираем лишнюю информацию из резюме

05 Убираем аномальные данные

```python
def process_data(input_data, good_keys, data_type):
    df = input_data.copy()
    translate_dict = {'about': "О себе",
                      'experienceItem': 'Опыт работы',
                      'educationItem': 'Образование',
                      'key_skills': "Стэк",
                      'uuid': "id",
                      'first_name': "Имя",
                      'last_name': "Фамилия",
                      'country': "Страна",
                      'city': "Город",
                      }
    experience_item_good_keys = ['position', 'description']

    for i, resume_list in enumerate(df['resumes']):
        for j, resume_dict in enumerate(resume_list):
            string = ""
            for key, value in resume_dict.items():
                if key in good_keys and value:
                    string += f"{translate_dict[key]}: "
                    if key == 'experienceItem':
                        for exp_item in resume_dict['experienceItem']:
                            for k, v in exp_item.items():
                                if k in experience_item_good_keys and v:
                                    string += f"{v} "
                                if k == "starts":
                                    num = calculate_experience(v, exp_item.get('ends', ''))
                                    string += f"Время работы: {num} года "
                    elif key == 'educationItem':
                        for edu_item in resume_dict['educationItem']:
                            for k in ['faculty', 'specialty']:
                                if edu_item[k]:
                                    string += f"{edu_item[k]} "
                                    break
                            for k in ['education_type', 'education_level', 'result']:
                                if edu_item[k]:
                                    string += f"{edu_item[k]} "
                                    break
```

```python
                    else:
                        string += f"{value} "
            if data_type == "test_input":
              df['resumes'][i][j] = [resume_dict['uuid'], string, 1 if resume_dict['passed'] == 1 else 0]
            elif data_type == "result_input":
              df['resumes'][i][j] = [resume_dict['uuid'], string]

    for i, vacancy_dict in enumerate(df['vacancy']):
        string = ""
        for key, value in vacancy_dict.items():
            if key != "uuid":
                if key == 'name' and value:
                    string += f"Название вакансии: {value} "
                elif key == 'keywords' and value:
                    string += f"Стэк: {value} "
                elif value:
                    string += f"{value} "
        df['vacancy'][i] = [vacancy_dict['uuid'], string]

    df = pd.DataFrame(df)

    return df
```

Векторизация

Векторизация проходит с помощью предобученной модели SBERT-ru

SBERT-ru

03

использовали предварительно обученную модель для создания семантических связей

Токенизация

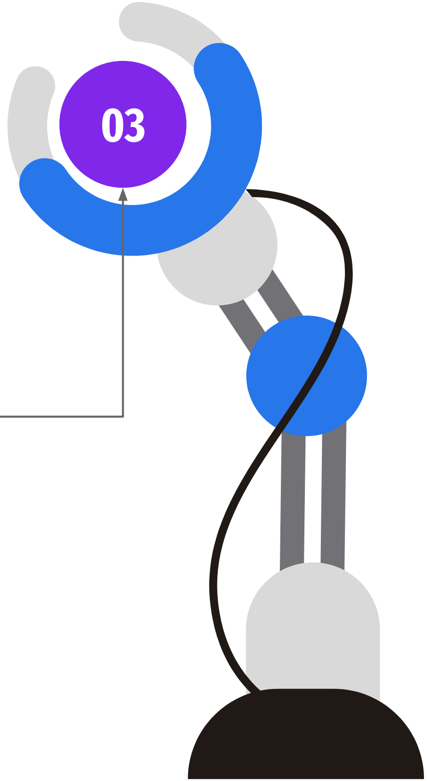
02

присваивали каждому слову уникальное место в общем словаре

Препроцессинг

01

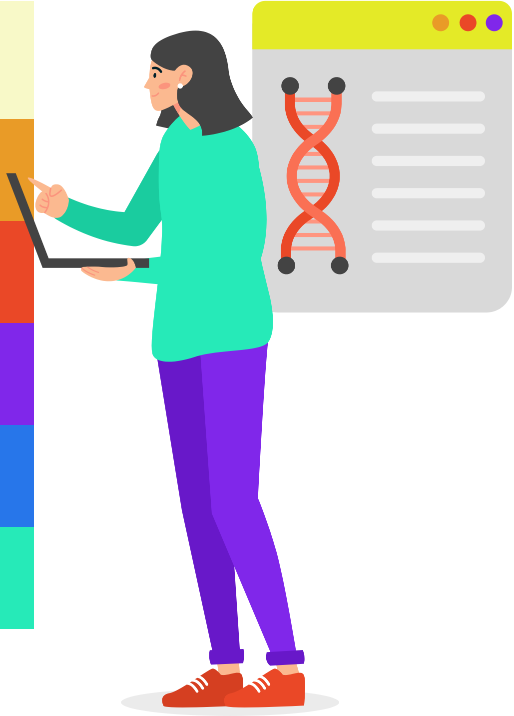
приводили входные данные в общий вид



```python
#Mean Pooling - Take attention mask into account for correct averaging
def mean_pooling(model_output, attention_mask):
    token_embeddings = model_output[0] #First element of model_output contains all token embeddings
    input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
    sum_embeddings = torch.sum(token_embeddings * input_mask_expanded, 1)
    sum_mask = torch.clamp(input_mask_expanded.sum(1), min=1e-9)
    return sum_embeddings / sum_mask
#Sentences we want sentence embeddings for
#Load AutoModel from huggingface model repository
tokenizer = AutoTokenizer.from_pretrained("ai-forever/sbert_large_mt_nlu_ru")
model = AutoModel.from_pretrained("ai-forever/sbert_large_mt_nlu_ru")
sentence_embeddings = []
for k in range(29):
#Tokenize sentences
    encoded_input = tokenizer(sentences[k], padding=True, truncation=True, max_length=512, return_tensors='pt')
    #Compute token embeddings
    with torch.no_grad():
        model_output = model(**encoded_input)
    #Perform pooling. In this case, mean pooling
    sentence_embedding = mean_pooling(model_output, encoded_input['attention_mask'])
    sentence_embeddings.append(sentence_embedding[:-1])## Добавили без вакансий
    jobs_vector.append(sentence_embedding[-1]) ## наполняем вектор вакансий
```

Обучение

Модели
Логистическая регрессия
SVM
Градиентный бустинг
Random Forest
Ансамбль моделей



```python
class LogisticRegressionModel:
    def __init__(self, X_train, y_train, X_test, y_test):
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.model = None
        self.best_params = None
        self.accuracy = None

    def train(self, param_grid):
        model = LogisticRegression()
        grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
        grid_search.fit(self.X_train, self.y_train)
        self.model = grid_search.best_estimator_
        self.best_params = grid_search.best_params_

    def evaluate(self):
        y_pred = self.model.predict(self.X_test)
        self.accuracy = accuracy_score(self.y_test, y_pred)
        print("Best parameters:", self.best_params)
        print(f'Accuracy: {self.accuracy}')
        print(classification_report(self.y_test, y_pred))

    def predict(self, X_new):
        return self.model.predict(X_new)
```

```python
# Пример использования
param_grid = {
    'penalty': ['l1', 'l2'],  # Регуляризация: l1 - Lasso, l2 - Ridge
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Обратная сила регуляризации
    'solver': ['liblinear', 'saga']  # Алгоритм оптимизации
}

# Создание экземпляра модели и обучение
log_reg_model = LogisticRegressionModel(X_train, y_train, X_test, y_test)
log_reg_model.train(param_grid)

# Оценка модели
log_reg_model.evaluate()
```

```
Best parameters: {'C': 0.001, 'penalty': 'l1', 'solver': 'liblinear'}
Accuracy: 0.7633587786259542
              precision    recall  f1-score   support

           0       0.76      1.00      0.87       100
           1       0.00      0.00      0.00        31

    accuracy                           0.76       131
   macro avg       0.38      0.50      0.43       131
weighted avg       0.58      0.76      0.66       131
```

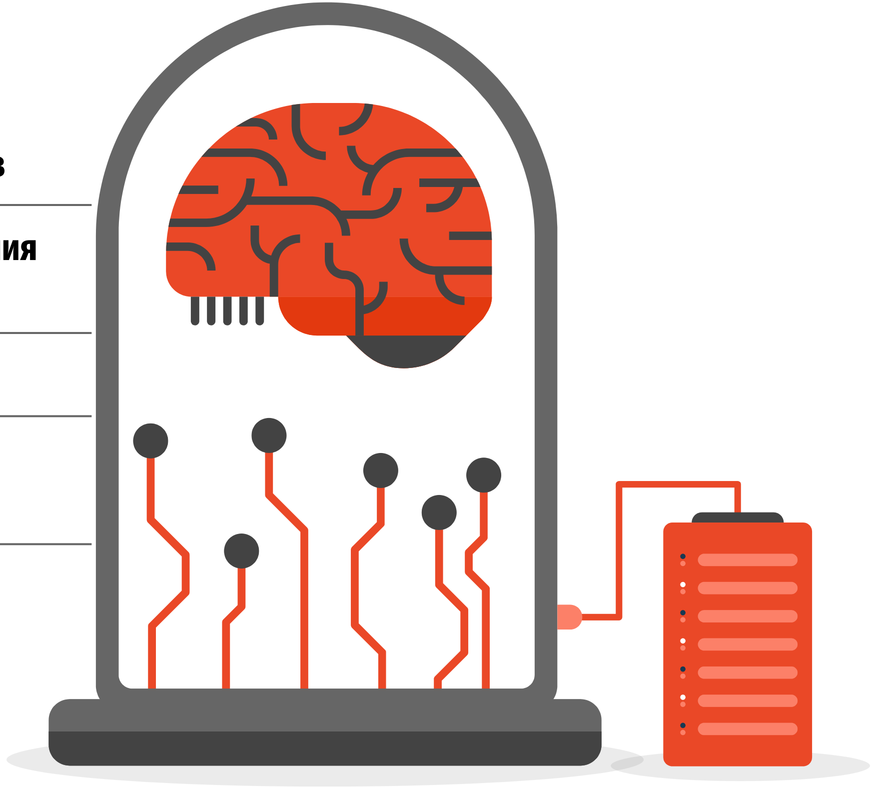
Ранжирование

01 Считаем разные метрики для подходящих векторов

02 Определяем минимальные и максимальные значения для разных метрик

03 Нормализуем значение метрик для векторов

04 Сортируем значение по убыванию объединенной метрики



```python
import numpy as np
from scipy.stats import pearsonr
from sklearn.metrics.pairwise import cosine_similarity

# Заданный вектор вакансии - задан ранее, job_vector

# Векторы резюме
resume_vectors = result_accepted_vecs['vecs']

# Определение метрик
def manhattan_distance(vec1, vec2):
    return np.sum(np.abs(vec1 - vec2))

def euclidean_distance(vec1, vec2):
    return np.sqrt(np.sum((vec1 - vec2) ** 2))

def pearson_correlation(vec1, vec2):
    corr, _ = pearsonr(vec1, vec2)
    return corr

def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    similarity = dot_product / (norm_vec1 * norm_vec2)
    return similarity
```

```python
# Вычисление минимальных и максимальных значений метрик
min_manhattan_dist = np.inf
max_manhattan_dist = -np.inf
min_euclidean_dist = np.inf
max_euclidean_dist = -np.inf
min_pearson_corr = np.inf
max_pearson_corr = -np.inf
min_cosine_sim = np.inf
max_cosine_sim = -np.inf

for resume_vector in resume_vectors:
    manhattan_dist = manhattan_distance(resume_vector, job_vector)
    euclidean_dist = euclidean_distance(resume_vector, job_vector)
    pearson_corr = pearson_correlation(resume_vector, job_vector)
    cosine_sim = cosine_similarity(resume_vector, job_vector)

    min_manhattan_dist = min(min_manhattan_dist, manhattan_dist)
    max_manhattan_dist = max(max_manhattan_dist, manhattan_dist)
    min_euclidean_dist = min(min_euclidean_dist, euclidean_dist)
    max_euclidean_dist = max(max_euclidean_dist, euclidean_dist)
    min_pearson_corr = min(min_pearson_corr, pearson_corr)
    max_pearson_corr = max(max_pearson_corr, pearson_corr)
    min_cosine_sim = min(min_cosine_sim, cosine_sim)
    max_cosine_sim = max(max_cosine_sim, cosine_sim)
```

```python
# Нормализация значений метрик
def normalize(score, min_val, max_val):
    return (score - min_val) / (max_val - min_val) if max_val != min_val else 0

# Ранжирование резюме
ranked_resumes = []
counter = 0
for resume_vector in resume_vectors:
    manhattan_dist = manhattan_distance(resume_vector, job_vector)
    euclidean_dist = euclidean_distance(resume_vector, job_vector)
    pearson_corr = pearson_correlation(resume_vector, job_vector)
    cosine_sim = cosine_similarity(resume_vector, job_vector)

    # Нормализация значений метрик
    manhattan_dist_normalized = normalize(manhattan_dist, min_manhattan_dist, max_manhattan_dist)
    euclidean_dist_normalized = normalize(euclidean_dist, min_euclidean_dist, max_euclidean_dist)
    pearson_corr_normalized = normalize(pearson_corr, min_pearson_corr, max_pearson_corr)
    cosine_sim_normalized = normalize(cosine_sim, min_cosine_sim, max_cosine_sim)

    # Объединение и нормализация значений метрик
    combined_score = (manhattan_dist_normalized + euclidean_dist_normalized +
                      pearson_corr_normalized + cosine_sim_normalized) / 4

    ranked_resumes.append((resume_vector, combined_score, counter))
    counter += 1

# Сортировка резюме по убыванию значения объединенной метрики
ranked_resumes.sort(key=lambda x: x[1], reverse=True)
```

```python
# Вывод результатов
res_ = []
for resume, score, counter in ranked_resumes:
    print(f"UUID: {result_accepted_vecs['uuid'][counter]}, Combined Score: {score}")
    res_.append(result_accepted_vecs['uuid'][counter])
    # Вывод резюме или его идентификатора
res_df = pd.DataFrame(res_)
```

```python
res_df = res_df.rename(columns={0: 'Sorted UUID'})
res_df
```

	Sorted UUID
0	37cba700-eed6-3018-bad6-f720f8217aeb
1	93d61f89-b8d0-3187-8c90-888be29e68dd
2	a71b0749-1ebc-3099-b0a1-342ca64d1575
3	ebcd86ef-6e1f-39cf-8af3-85adaec6d3b3
4	9a9b0a97-3514-3137-b8b8-129474b24528
5	fd0ccbd0-3a58-3818-8691-98f31de17527
6	73592479-12bf-38d4-84f0-91fe33518b47
7	d9847b13-dc30-36e7-8d75-c632495a7eec
8	fbab422e-7e0b-38d2-a7bc-cc7286858c10
9	5785c202-6744-3e1b-994a-d5bffc6aad14
10	c70de373-9f3a-3647-ab66-f25e98c29409
11	cc88bf96-f0b9-313a-abce-dbe60b6f1c98