[Feature Engineering] [Extended-cheatsheet]

1. Data Preprocessing

1.1 Handling Missing Values

- Check for missing values: `df.isnull().sum()`
- Drop rows with missing values: `df.dropna()`
- Fill missing values with a specific value: `df.fillna(value)`
- Fill missing values with mean: `df.fillna(df.mean())`
- Fill missing values with median: `df.fillna(df.median())`
- Fill missing values with mode: `df.fillna(df.mode().iloc[0])`
- Fill missing values with forward fill: `df.fillna(method='ffill')`
- Fill missing values with backward fill: `df.fillna(method='bfill')`

1.2 Encoding Categorical Variables

- One-hot encoding: `pd.get_dummies(df, columns=['column_name'])`
- Label encoding: `from sklearn.preprocessing import LabelEncoder; LabelEncoder().fit_transform(df['column_name'])`
- Ordinal encoding: `from sklearn.preprocessing import OrdinalEncoder; OrdinalEncoder().fit_transform(df[['column_name']])`
- Binary encoding: `df['binary_column'] = np.where(df['column_name'] == 'value', 1, 0)`
- Frequency encoding: `df['freq_encoded'] = df.groupby('column_name')['column_name'].transform('count')`
- Mean encoding: `df['mean_encoded'] = df.groupby('column_name')['target'].transform('mean')`
- Weight of Evidence (WoE) encoding: `df['woe'] = np.log(df.groupby('column_name')['target'].mean() / (1 - df.groupby('column_name')['target'].mean()))`

1.3 Scaling and Normalization

- Min-max scaling: `from sklearn.preprocessing import MinMaxScaler; MinMaxScaler().fit_transform(df[['column_name']])`
- Standard scaling (Z-score normalization): `from sklearn.preprocessing import StandardScaler; StandardScaler().fit_transform(df[['column_name']])`

By: Waleed Mousa

- Max-abs scaling: `from sklearn.preprocessing import MaxAbsScaler; MaxAbsScaler().fit_transform(df[['column_name']])`
- Robust scaling: `from sklearn.preprocessing import RobustScaler; RobustScaler().fit_transform(df[['column_name']])`
- Normalization (L1, L2, Max): `from sklearn.preprocessing import Normalizer; Normalizer(norm='l1').fit_transform(df[['column_name']])`

1.4 Handling Outliers

- Identify outliers using IQR: `Q1 = df['column_name'].quantile(0.25); Q3 = df['column_name'].quantile(0.75); IQR = Q3 - Q1; df[(df['column_name'] < Q1 - 1.5 * IQR) | (df['column_name'] > Q3 + 1.5 * IQR)]`
- Identify outliers using Z-score: `from scipy.stats import zscore; df[np.abs(zscore(df['column_name'])) > 3]`
- Remove outliers: `df = df[(df['column_name'] >= lower_bound) & (df['column_name'] <= upper_bound)]`
- Cap outliers: `df['column_name'] = np.where(df['column_name'] > upper_bound, upper_bound, np.where(df['column_name'] < lower_bound, lower_bound, df['column_name']))`

2. Feature Transformation

2.1 Mathematical Transformations

- Logarithmic transformation: `df['log_column'] = np.log(df['column_name'])`
- Square root transformation: `df['sqrt_column'] = np.sqrt(df['column_name'])`
- Exponential transformation: `df['exp_column'] = np.exp(df['column_name'])`
- Reciprocal transformation: `df['reciprocal_column'] = 1 / df['column_name']`
- Box-Cox transformation: `from scipy.stats import boxcox; df['boxcox_column'] = boxcox(df['column_name'])[0]`
- Yeo-Johnson transformation: `from scipy.stats import yeojohnson; df['yeojohnson_column'] = yeojohnson(df['column_name'])[0]`

2.2 Binning and Discretization

- Equal-width binning: `pd.cut(df['column_name'], bins=n)`
- Equal-frequency binning: `pd.qcut(df['column_name'], q=n)`
- Custom binning: `pd.cut(df['column_name'], bins=[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])`

- Discretization using KBinsDiscretizer: `from sklearn.preprocessing import KBinsDiscretizer; KBinsDiscretizer(n_bins=n, encode='ordinal').fit_transform(df[['column_name']])`

2.3 Interaction Features

- Multiplication: `df['interaction'] = df['column_1'] * df['column_2']`
- Division: `df['interaction'] = df['column_1'] / df['column_2']`
- Addition: `df['interaction'] = df['column_1'] + df['column_2']`
- Subtraction: `df['interaction'] = df['column_1'] - df['column_2']`
- Polynomial features: `from sklearn.preprocessing import PolynomialFeatures; PolynomialFeatures(degree=n).fit_transform(df[['column_1', 'column_2']])`

2.4 Date and Time Features

- Extract year: `df['year'] = df['date_column'].dt.year`
- Extract month: `df['month'] = df['date_column'].dt.month`
- Extract day: `df['day'] = df['date_column'].dt.day`
- Extract hour: `df['hour'] = df['datetime_column'].dt.hour`
- Extract minute: `df['minute'] = df['datetime_column'].dt.minute`
- Extract second: `df['second'] = df['datetime_column'].dt.second`
- Extract day of week: `df['day_of_week'] = df['date_column'].dt.dayofweek`
- Extract day of year: `df['day_of_year'] = df['date_column'].dt.dayofyear`
- Extract week of year: `df['week_of_year'] = df['date_column'].dt.weekofyear`
- Extract quarter: `df['quarter'] = df['date_column'].dt.quarter`
- Extract is_weekend: `df['is_weekend'] = df['date_column'].dt.dayofweek.isin([5, 6])`
- Extract is_holiday: `holidays = ['2023-01-01', '2023-12-25']; df['is_holiday'] = df['date_column'].isin(holidays)`
- Time since feature: `df['time_since'] = (df['date_column'] - df['reference_date']).dt.days`

3. Feature Selection

3.1 Univariate Feature Selection

- Select K best features: `from sklearn.feature_selection import SelectKBest, f_classif; SelectKBest(score_func=f_classif, k=n).fit_transform(X, y)`

- Select percentile of features: `from sklearn.feature_selection import SelectPercentile, f_classif; SelectPercentile(score_func=f_classif, percentile=p).fit_transform(X, y)`

3.2 Recursive Feature Elimination

- Recursive Feature Elimination (RFE): `from sklearn.feature_selection import RFE; from sklearn.linear_model import LinearRegression; RFE(estimator=LinearRegression(), n_features_to_select=n).fit_transform(X, y)`
- Recursive Feature Elimination with Cross-Validation (RFECV): `from sklearn.feature_selection import RFECV; from sklearn.linear_model import LinearRegression; RFECV(estimator=LinearRegression(), min_features_to_select=n).fit_transform(X, y)`

3.3 L1 and L2 Regularization

- Lasso (L1) regularization: `from sklearn.linear_model import Lasso; Lasso(alpha=a).fit_transform(X, y)`
- Ridge (L2) regularization: `from sklearn.linear_model import Ridge; Ridge(alpha=a).fit_transform(X, y)`
- Elastic Net regularization: `from sklearn.linear_model import ElasticNet; ElasticNet(alpha=a, l1_ratio=r).fit_transform(X, y)`

3.4 Feature Importance

- Feature importance using Random Forest: `from sklearn.ensemble import RandomForestClassifier; rf = RandomForestClassifier(); rf.fit(X, y); rf.feature_importances_`
- Feature importance using Gradient Boosting: `from sklearn.ensemble import GradientBoostingClassifier; gb = GradientBoostingClassifier(); gb.fit(X, y); gb.feature_importances_`
- Permutation feature importance: `from sklearn.inspection import permutation_importance; permutation_importance(model, X, y, n_repeats=n)`

4. Dimensionality Reduction

4.1 Principal Component Analysis (PCA)

- PCA: `from sklearn.decomposition import PCA; PCA(n_components=n).fit_transform(X)`

- Incremental PCA: `from sklearn.decomposition import IncrementalPCA; IncrementalPCA(n_components=n).fit_transform(X)`
- Kernel PCA: `from sklearn.decomposition import KernelPCA; KernelPCA(n_components=n, kernel='rbf').fit_transform(X)`

4.2 t-SNE (t-Distributed Stochastic Neighbor Embedding)

- t-SNE: `from sklearn.manifold import TSNE; TSNE(n_components=n).fit_transform(X)`

4.3 UMAP (Uniform Manifold Approximation and Projection)

- UMAP: `from umap import UMAP; UMAP(n_components=n).fit_transform(X)`

4.4 Autoencoders

- Autoencoder using Keras: `from keras.layers import Input, Dense; from keras.models import Model; input_layer = Input(shape=(n,)); encoded = Dense(encoding_dim, activation='relu')(input_layer); decoded = Dense(n, activation='sigmoid')(encoded); autoencoder = Model(input_layer, decoded); encoder = Model(input_layer, encoded)`

5. Text Feature Engineering

5.1 Text Preprocessing

- Lowercase: `df['text'] = df['text'].str.lower()`
- Remove punctuation: `df['text'] = df['text'].str.replace('[^a-zA-Z]', ' ')`
- Remove stopwords: `from nltk.corpus import stopwords; stop_words = set(stopwords.words('english')); df['text'] = df['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))`
- Stemming: `from nltk.stem import PorterStemmer; ps = PorterStemmer(); df['text'] = df['text'].apply(lambda x: ' '.join([ps.stem(word) for word in x.split()]))`
- Lemmatization: `from nltk.stem import WordNetLemmatizer; lemmatizer = WordNetLemmatizer(); df['text'] = df['text'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word) for word in x.split()]))`

5.2 Text Vectorization

- Bag-of-Words (BoW): `from sklearn.feature_extraction.text import CountVectorizer; CountVectorizer().fit_transform(df['text'])`

- TF-IDF: from sklearn.feature_extraction.text import TfidfVectorizer; TfidfVectorizer().fit_transform(df['text'])
- Word2Vec: from gensim.models import Word2Vec; Word2Vec(sentences=df['text'], vector_size=n, window=w, min_count=m, workers=wrk)
- GloVe: from gensim.models import KeyedVectors; KeyedVectors.load_word2vec_format('glove.6B.100d.txt', binary=False)
- FastText: from gensim.models import FastText; FastText(sentences=df['text'], vector_size=n, window=w, min_count=m, workers=wrk)
- BERT: from transformers import BertTokenizer, BertModel; tokenizer = BertTokenizer.from_pretrained('bert-base-uncased'); model = BertModel.from_pretrained('bert-base-uncased')

5.3 Text Feature Extraction

- Named Entity Recognition (NER): from nltk import word_tokenize, pos_tag, ne_chunk; ne_chunk(pos_tag(word_tokenize(text)))
- Part-of-Speech (POS) tagging: from nltk import word_tokenize, pos_tag; pos_tag(word_tokenize(text))
- Sentiment Analysis: from textblob import TextBlob; TextBlob(text).sentiment

6. Image Feature Engineering

6.1 Image Preprocessing

- Resize: from PIL import Image; img = Image.open('image.jpg'); img = img.resize((width, height))
- Convert to grayscale: from PIL import Image; img = Image.open('image.jpg'); img = img.convert('L')
- Normalize pixel values: from PIL import Image; img = Image.open('image.jpg'); img = img / 255.0
- Data augmentation (flip, rotate, etc.): from keras.preprocessing.image import ImageDataGenerator; datagen = ImageDataGenerator(rotation_range=r, width_shift_range=ws, height_shift_range=hs, shear_range=s, zoom_range=z, horizontal_flip=True)

6.2 Image Feature Extraction

- HOG (Histogram of Oriented Gradients): `from skimage.feature import hog; hog_features = hog(image, orientations=n, pixels_per_cell=(p, p), cells_per_block=(c, c))`
- SIFT (Scale-Invariant Feature Transform): `from cv2 import xfeatures2d; sift = xfeatures2d.SIFT_create(); keypoints, descriptors = sift.detectAndCompute(image, None)`
- ORB (Oriented FAST and Rotated BRIEF): `from cv2 import ORB; orb = ORB_create(); keypoints, descriptors = orb.detectAndCompute(image, None)`
- CNN features: `from keras.applications.vgg16 import VGG16; model = VGG16(weights='imagenet', include_top=False); features = model.predict(image)`

7. Audio Feature Engineering

7.1 Audio Preprocessing

- Load audio file: `import librosa; audio, sample_rate = librosa.load('audio.wav')`
- Resampling: `import librosa; audio = librosa.resample(audio, orig_sr=sample_rate, target_sr=target_sample_rate)`
- Normalize audio: `import librosa; audio = librosa.util.normalize(audio)`

7.2 Audio Feature Extraction

- MFCC (Mel-Frequency Cepstral Coefficients): `import librosa; mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate)`
- Chroma features: `import librosa; chroma = librosa.feature.chroma_stft(y=audio, sr=sample_rate)`
- Spectral contrast: `import librosa; contrast = librosa.feature.spectral_contrast(y=audio, sr=sample_rate)`
- Tonnetz: `import librosa; tonnetz = librosa.feature.tonnetz(y=audio, sr=sample_rate)`

8. Time Series Feature Engineering

8.1 Time Series Decomposition

- Decompose time series into trend, seasonality, and residuals: `from statsmodels.tsa.seasonal import seasonal_decompose; decomposition = seasonal_decompose(time_series, model='additive', period=p)`

- STL decomposition: `from statsmodels.tsa.seasonal import STL; stl = STL(time_series, period=p); res = stl.fit()`

8.2 Rolling and Expanding Statistics

- Rolling mean: `time_series.rolling(window=n).mean()`
- Rolling standard deviation: `time_series.rolling(window=n).std()`
- Expanding mean: `time_series.expanding(min_periods=n).mean()`
- Expanding standard deviation: `time_series.expanding(min_periods=n).std()`

8.3 Lag Features

- Shift/lag feature: `df['lag_1'] = df['column'].shift(1)`
- Difference feature: `df['diff_1'] = df['column'].diff(1)`
- Percentage change feature: `df['pct_change_1'] = df['column'].pct_change(1)`

8.4 Autocorrelation and Partial Autocorrelation

- Autocorrelation Function (ACF): `from statsmodels.tsa.stattools import acf; acf_values = acf(time_series, nlags=n)`
- Partial Autocorrelation Function (PACF): `from statsmodels.tsa.stattools import pacf; pacf_values = pacf(time_series, nlags=n)`

9. Geospatial Feature Engineering

9.1 Geospatial Distance and Proximity

- Haversine distance: `from math import radians, cos, sin, asin, sqrt; def haversine(lon1, lat1, lon2, lat2): lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2]); dlon = lon2 - lon1; dlat = lat2 - lat1; a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2; c = 2 * asin(sqrt(a)); r = 6371; return c * r`
- Manhattan distance: `from math import fabs; def manhattan(x1, y1, x2, y2): return fabs(x1 - x2) + fabs(y1 - y2)`
- Euclidean distance: `from math import sqrt; def euclidean(x1, y1, x2, y2): return sqrt((x1 - x2)**2 + (y1 - y2)**2)`

9.2 Geospatial Aggregation

- Spatial join: `import geopandas as gpd; gpd.sjoin(gdf1, gdf2, op='intersects')`

- Spatial groupby: `import geopandas as gpd; gdf.groupby('column').agg({'geometry': 'first', 'other_column': 'mean'})`

9.3 Geospatial Binning

- Create grid: `import geopandas as gpd; grid = gpd.GeoDataFrame(geometry=gpd.points_from_xy(x, y))`
- Spatial binning: `import geopandas as gpd; gdf['grid_id'] = gpd.sjoin(gdf, grid, op='within')['index_right']`

10. Feature Scaling and Normalization

10.1 Scaling

- Min-max scaling: `from sklearn.preprocessing import MinMaxScaler; MinMaxScaler().fit_transform(df[['column_name']])`
- Standard scaling (Z-score normalization): `from sklearn.preprocessing import StandardScaler; StandardScaler().fit_transform(df[['column_name']])`
- Max-abs scaling: `from sklearn.preprocessing import MaxAbsScaler; MaxAbsScaler().fit_transform(df[['column_name']])`
- Robust scaling: `from sklearn.preprocessing import RobustScaler; RobustScaler().fit_transform(df[['column_name']])`

10.2 Normalization

- L1 normalization: `from sklearn.preprocessing import Normalizer; Normalizer(norm='l1').fit_transform(df[['column_name']])`
- L2 normalization: `from sklearn.preprocessing import Normalizer; Normalizer(norm='l2').fit_transform(df[['column_name']])`
- Max normalization: `from sklearn.preprocessing import Normalizer; Normalizer(norm='max').fit_transform(df[['column_name']])`