Notes

Feature Engineering Week 1 to Week 8

Disclaimer

This content has been generated with the assistance of AI for educational and reference purposes only. While every effort has been made to ensure accuracy, errors or omissions may still exist — especially in code examples or technical explanations. Please verify all information and code correctness by referring to the original tutorial video, official documentation, or reading material before applying it in practice.

Week 1: Overview of Feature Engineering

Welcome to the foundational week of our course! Feature engineering is arguably the most creative and impactful part of the machine learning pipeline. It's the process of using domain knowledge to transform raw data into features that better represent the underlying problem to the predictive models. A great model with poor features will always be outmatched by a decent model with great features.

Key Topics

1. Role and Purpose of Feature Engineering

At its core, **feature engineering** is about making your data more suitable for a machine learning algorithm. Raw data is often messy, incomplete, and not in a format that algorithms can directly learn from. The goal is to create **features** (or variables) that help algorithms "see" the patterns in the data more clearly.

Why is it important?

- **Improves Model Performance:** This is the primary goal. Better features lead to higher accuracy, precision, and recall.
- **Reduces Complexity:** Well-engineered features can help you use simpler, more interpretable models.
- **Reduces Computational Cost:** By selecting only the most important features, you can significantly speed up training and inference times.

Think of it like this: you're a detective trying to solve a case (the prediction task). The raw data is a pile of unsorted evidence. Feature engineering is the process of organizing that evidence, highlighting important clues (features), and presenting a clear case file to the judge (the ML model).

2. General Techniques for Feature Engineering

Throughout this course, we'll dive deep into specific techniques. For now, let's look at a high-level overview:

- **Imputation:** Filling in missing values.
- **Discretization (Binning):** Converting continuous variables into categorical ones.
- **Categorical Encoding:** Converting categorical variables into numerical ones (e.g., One-Hot Encoding).
- **Feature Scaling:** Changing the scale of variables to a standard range.
- **Transformations:** Applying mathematical functions like log or square root to change a

variable's distribution.

- **Feature Creation:** Combining existing features to create new, more informative ones (e.g., combining `height` and `weight` to create `BMI`).
- **Feature Selection:** Choosing the most relevant features to train your model on.

3. Challenges of High-Dimensional Data

High-dimensional data refers to datasets with a very large number of features. Think of a dataset of customer purchase history with thousands of product types as features. This brings several challenges, often called the **"Curse of Dimensionality"**:

- **Increased Sparsity:** As you add more dimensions, the data points become farther apart. This makes it harder for algorithms to find meaningful patterns.
- **Overfitting:** With more features than data points, a model can start to "memorize" the training data instead of learning the general trend, leading to poor performance on new data.
- **Computational Complexity:** More dimensions mean more calculations, leading to longer training times.

Dimensionality reduction techniques, which we'll cover in Week 4, are essential for tackling this problem.

Hands-on Example: Feature Creation

Let's use a simple, hypothetical housing dataset to demonstrate creating a new feature.

Python

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = {
    'total_rooms': [2500, 3100, 1800, 4200],
    'num_bedrooms': [4, 5, 3, 6],
    'population': [1200, 1500, 800, 2000],
    'price': [350000, 480000, 250000, 620000]
}
df = pd.DataFrame(data)

print("Original Data:")
print(df)

# Feature Creation: Let's create 'rooms_per_person' and 'avg_room_size'
df['rooms_per_person'] = df['total_rooms'] / df['population']
```

```
# A simple proxy for avg_room_size assuming bedrooms are the main rooms
df['avg_room_size_proxy'] = df['total_rooms'] / df['num_bedrooms']

print("\nData with New Features:")
print(df)

# Visualize the relationship of the new feature with price
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.scatterplot(x='rooms_per_person', y='price', data=df)
plt.title('Price vs. Rooms per Person')

plt.subplot(1, 2, 2)
sns.scatterplot(x='avg_room_size_proxy', y='price', data=df)
plt.title('Price vs. Avg Room Size (Proxy)')

plt.tight_layout()
plt.show()
```

In this example, `rooms_per_person` might be a better indicator of how "spacious" a house is for its inhabitants than just the raw `total_rooms` or `population` count. This is a simple but powerful example of feature creation.

❓ Week 1: Quiz

1. **What is the primary goal of feature engineering?**
   a) To make the data look cleaner.
   b) To improve the performance of predictive models.
   c) To increase the number of features.
   d) To reduce the size of the dataset.
2. **Which of the following is a direct consequence of the "Curse of Dimensionality"?**
   a) Faster model training.
   b) Models become easier to interpret.
   c) Increased risk of overfitting.
   d) Data becomes less sparse.
3. **Combining `total_sales` and `number_of_transactions` to create a `sale_per_transaction` feature is an example of:**
   a) Feature Scaling
   b) Imputation
   c) Feature Creation
   d) Discretization

(Answers: 1-b, 2-c, 3-c)

🙋 Week 1: Q&A

Q: Is feature engineering a one-time process?
A: Absolutely not. It's an iterative process. You'll often create features, train a model, evaluate its performance, and then go back to create new features or modify existing ones based on the results.

Q: How much domain knowledge do I need?
A: Domain knowledge is extremely helpful. An expert in finance will know which financial ratios are important features for predicting stock prices. However, even without deep expertise, you can use exploratory data analysis (EDA) to uncover relationships and generate ideas for new features.

Week 2: Handling Numeric Data

Numeric data is the most common data type you'll encounter. While models can process numbers directly, how you format and scale them can dramatically impact performance, especially for algorithms that rely on distances or gradients.

Key Topics

1. Feature Scaling

Feature scaling ensures that all numeric features are on a similar scale. This is crucial for algorithms like **K-Nearest Neighbors (KNN)**, **Support Vector Machines (SVMs)**, and **Principal Component Analysis (PCA)**, which are sensitive to the magnitude of features.

Here's a comparison of the main techniques:

Technique	Formula	Range	When to Use
Standardization	$z = \frac{x-\mu}{\sigma}$	Not bounded	Default choice. Good when data is normally distributed.
Min-Max Scaling	$x_{scaled} = \frac{x-x_{min}}{x_{max}-x_{min}}$	Typically $[0, 1]$	When you need features in a specific range. Sensitive to outliers.
Robust Scaling	$x_{scaled} = \frac{x-Q1}{Q3-Q1}$	Varies	When your data has significant outliers. Uses interquartile range.

Export to Sheets

Code Example: Scaling in Action

Python

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Generate data with an outlier
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 20]).reshape(-1, 1)

# Apply scalers
standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()
robust_scaler = RobustScaler()

scaled_data = pd.DataFrame({
    'Original': data.flatten(),
    'Standard': standard_scaler.fit_transform(data).flatten(),
    'Min-Max': minmax_scaler.fit_transform(data).flatten(),
    'Robust': robust_scaler.fit_transform(data).flatten()
})

# Plot the distributions
plt.figure(figsize=(12, 6))
sns.kdeplot(data=scaled_data, fill=True)
plt.title('Comparison of Scaling Techniques')
plt.xlabel('Value')
plt.ylabel('Density')
plt.show()
```

Notice how the outlier (20) forces most of the data in Min-Max scaling into a small range, while Robust scaling is less affected.

2. Attribute Transformations

Transformations are used to change the distribution of a variable, often to make it more Gaussian (bell-shaped). This can help models that assume normality, like Linear Regression.

- **Logarithmic Transformation ($log(x)$):** Useful for skewed data. It compresses the range of large values and expands the range of small values. Only applicable for positive values.
- **Power Transformations (e.g., Square Root, Box-Cox):** More general methods to stabilize variance and make data more normal-like. Box-Cox can find the best power transformation automatically.

Code Example: Log Transform for Skewed Data

Python

```
# Generate skewed data
```

```
skewed_data = np.random.exponential(scale=2, size=1000)

# Apply log transform (add 1 to handle potential zeros)
log_transformed_data = np.log1p(skewed_data)

# Plotting
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(skewed_data, kde=True)
plt.title('Original Skewed Data')

plt.subplot(1, 2, 2)
sns.histplot(log_transformed_data, kde=True)
plt.title('Log-Transformed Data')
plt.show()
```

3. Discretization

Discretization, or binning, turns continuous features into categorical ones. This can sometimes help models capture non-linear relationships.

- **Equal-Width Binning:** Divides the data into bins of equal width. Sensitive to outliers.
- **Equal-Frequency Binning (Quantile Binning):** Divides data into bins with an equal number of observations. Handles outliers better.
- **Domain-Based Binning:** Using expert knowledge to define bins (e.g., Age groups: 'Child', 'Adolescent', 'Adult', 'Senior').

❓ Week 2: Quiz

1. **You have a dataset of house prices with a few multi-million dollar mansions (outliers). Which scaling method would be most appropriate for the price feature?**
   a) Standardization
   b) Min-Max Scaling
   c) Robust Scaling
   d) None, scaling is not needed.
2. **Why is feature scaling important for an algorithm like KNN?**
   a) KNN requires features to be in the range [0, 1].
   b) KNN calculates distances, which can be dominated by features with large scales.
   c) KNN assumes a normal distribution for all features.
   d) KNN cannot handle negative values.
3. **A log transformation is most useful for data that is:**
   a) Normally distributed
   b) Uniformly distributed
   c) Highly skewed

d) Categorical

(Answers: 1-c, 2-b, 3-c)

🙋 Week 2: Q&A

Q: Should I scale my data before or after splitting it into training and testing sets?
A: This is a critical point! You should **always** fit your scaler on the **training data only** and then use that *fitted* scaler to transform both your training and testing data. Fitting on the entire dataset would cause data leakage, where information from the test set "leaks" into your training process, giving you an overly optimistic performance estimate.

Q: Does feature scaling affect tree-based models like Decision Trees or Random Forests?
A: No, not really. Tree-based models make decisions by splitting on feature values (e.g., `age < 30`). The scale of the feature doesn't affect these splits. Therefore, scaling is generally not required for these models.

Week 3: General Feature Engineering Techniques

This week, we move from transforming individual features to the broader strategies of selecting the best features and constructing new ones. This is about finding the signal and removing the noise from your dataset.

Key Topics

1. Feature Selection Methods

Feature selection is the process of selecting a subset of relevant features for use in model construction. The main benefits are improved model interpretability, reduced overfitting, and decreased training time.

There are three main families of feature selection techniques:

Method Family	How it Works	Pros	Cons
Filter Methods	Features are selected based on their statistical properties (e.g., correlation with the target, chi-square score) **before** model training.	Fast, computationally cheap, model-agnostic.	Ignores feature dependencies; may not select the best subset for a specific model.
Wrapper Methods	A specific model is used to evaluate subsets of features. It "wraps" the model training process. Examples: Recursive Feature Elimination (RFE),	Considers feature interactions; finds model-specific optimal features.	Computationally very expensive, prone to overfitting on the selection process.

	Sequential Forward/Backward Selection (SFS/SBS).		
Embedded Methods	Feature selection is an intrinsic part of the model training process. Examples: LASSO (L1) Regression, tree-based feature importance.	More efficient than wrapper methods; considers feature interactions.	Tied to a specific model.

Export to Sheets

2. Filter-Based Methods in Detail

- **Correlation:** Measures the linear relationship between two variables. For a classification task, you'd look for high correlation between features and the target, and low correlation among features themselves (to avoid multicollinearity).
- **Chi-Square Test:** Used for categorical features. It tests whether the observed distribution of the target variable is consistent with the expected distribution if the feature were independent. A high Chi-Square score suggests the feature is dependent on the target and thus, useful.

Code Example: Using `SelectKBest` with Chi-Square

Python

```python
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, chi2
import pandas as pd

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# We want to select the top 2 features
selector = SelectKBest(score_func=chi2, k=2)
X_new = selector.fit_transform(X, y)

# Get the selected feature names
selected_features = pd.DataFrame(selector.inverse_transform(X_new),
                                 index=range(X.shape[0]),
                                 columns=iris.feature_names)

print("Original number of features:", X.shape[1])
print("Reduced number of features:", X_new.shape[1])

# Get scores for all features
feature_scores = pd.DataFrame({'Feature': iris.feature_names, 'Score': selector.sc
print("\nFeature Scores:")
print(feature_scores.sort_values(by='Score', ascending=False))
```

The output shows that `petal length` and `petal width` are the most informative features according to the Chi-Square test.

3. Wrapper Methods: SFS and SBS

- **Sequential Forward Selection (SFS):** Starts with an empty set of features. In each iteration, it adds the feature that best improves the model's performance until the desired number of features is reached.
- **Sequential Backward Selection (SBS):** Starts with all features. In each iteration, it removes the feature whose removal results in the smallest decrease in model performance.

These methods are greedy and can be computationally intensive, but they often find better feature subsets than filter methods.

❓ Week 3: Quiz

1. **An analyst calculates the correlation of all features with the target variable and selects the top 10. What kind of feature selection method is this?**
   a) Embedded Method
   b) Wrapper Method
   c) Filter Method
   d) Hybrid Method
2. **What is a major disadvantage of Wrapper methods?**
   a) They are model-agnostic.
   b) They are computationally expensive.
   c) They cannot capture interactions between features.
   d) They are faster than filter methods.
3. **L1 Regularization (LASSO) forces the coefficients of some features to become exactly zero. This is an example of a(n):**
   a) Filter Method
   b) Embedded Method
   c) Wrapper Method
   d) Transformation Technique

(Answers: 1-c, 2-b, 3-b)

🙋‍♀️ Week 3: Q&A

Q: Can I combine different feature selection methods?
A: Yes, and it's often a great strategy! A common approach is to use a fast filter method to quickly reduce the feature space from thousands to a few hundred, and then use a more powerful

wrapper or embedded method on that smaller set to find the optimal subset.

Q: How do I know how many features to select (e.g., the 'k' in `SelectKBest`)?
A: This is a hyperparameter you need to tune. You can use techniques like cross-validation, plotting the model performance against the number of features, or using methods like Recursive Feature Elimination with Cross-Validation (`RFECV` in scikit-learn) which automatically finds the optimal number of features.

Week 4: Dimensionality Reduction Techniques

This week, we tackle the "Curse of Dimensionality" head-on. Unlike feature selection, which *selects* a subset of original features, dimensionality reduction *transforms* the data into a lower-dimensional space, creating new features (components) that are combinations of the old ones.

Key Topics

1. Principal Component Analysis (PCA)

PCA is the most popular linear dimensionality reduction technique. It transforms the data by projecting it onto a set of new, orthogonal (uncorrelated) axes called **Principal Components**. These components are ordered such that the first component captures the largest possible variance in the data, the second captures the next largest variance, and so on.

Intuition: Imagine a 3D cloud of points shaped like a pancake. PCA would find that the most variance is along the length of the pancake (PC1), the next most is along the width (PC2), and the least is along the thickness (PC3). To reduce to 2D, you'd keep PC1 and PC2, effectively looking at the pancake "face-on" and discarding the near-zero thickness information.

Code Example: PCA on the Iris Dataset

Python

```python
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
iris = load_iris()
X = iris.data

# Apply PCA to reduce from 4 to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

```
# Explained variance
print(f"Explained variance by PC1: {pca.explained_variance_ratio_[0]:.2f}")
print(f"Explained variance by PC2: {pca.explained_variance_ratio_[1]:.2f}")
print(f"Total variance explained by 2 components: {sum(pca.explained_variance_ratio

# Visualize the result
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=iris.target, palette='viridis', s
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA of Iris Dataset')
plt.legend(handles=plt.gca().get_legend().legendHandles, labels=list(iris.target_na
plt.show()
```

As you can see, the first two principal components capture about 98% of the variance and do an excellent job of separating the three flower species.

2. Singular Value Decomposition (SVD)

SVD is a fundamental matrix factorization technique with many applications, including dimensionality reduction. For a data matrix A, SVD decomposes it into three matrices: $A = U\Sigma V^T$. For dimensionality reduction, a low-rank approximation of A can be obtained by taking only the top k singular values from Σ. `TruncatedSVD` in scikit-learn is very similar to PCA but works efficiently on sparse matrices, which is useful in text analysis (e.g., Latent Semantic Analysis).

3. t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a **non-linear** dimensionality reduction technique primarily used for **visualization**. It models the similarity between high-dimensional data points as a probability distribution and then optimizes a similar distribution in a low-dimensional space (usually 2D or 3D).

Key Difference from PCA: PCA focuses on preserving large pairwise distances (global structure), while t-SNE focuses on preserving small pairwise distances (local structure). This means t-SNE is excellent at revealing clusters in the data, but the distances and sizes of those clusters in the t-SNE plot are not necessarily meaningful.

4. Comparison of Techniques

Technique	Type	Goal	Use Case
PCA	Linear	Maximize variance, preserve global structure.	Feature extraction for modeling, noise reduction.
SVD	Linear	Matrix factorization.	Similar to PCA, very good for sparse data (e.g., text).

t-SNE	Non-linear	Preserve local structure, reveal clusters.	**Primarily for data visualization**, not for feature extraction for modeling.

Export to Sheets

❓ Week 4: Graded Quiz

1. **What is the primary objective of the first principal component in PCA?**
   a) To have a mean of zero and standard deviation of one.
   b) To be orthogonal to all other components.
   c) To capture the maximum possible variance in the data.
   d) To separate the data into clusters.
2. **You have successfully reduced your 100-feature dataset to 2 components using PCA. The `explained_variance_ratio_` is `[0.45, 0.40]`. What does this mean?**
   a) The two components capture 85% of the original variance.
   b) The first component is slightly more important than the second.
   c) The model will be 85% accurate.
   d) Both a and b.
3. **An analyst wants to create a 2D plot to see if there are any natural groupings in a high-dimensional dataset. Which technique would be most suitable for this specific task?**
   a) PCA
   b) t-SNE
   c) Robust Scaling
   d) SFS
4. **True or False: The distance between two clusters in a t-SNE plot is a reliable measure of the actual distance or separation between those groups in the original high-dimensional space.**
   a) True
   b) False

(Answers: 1-c, 2-d, 3-b, 4-b)

🙋 Week 4: Q&A

Q: Should I scale my data before applying PCA?
A: Yes, absolutely! PCA is variance-based. If you have one feature with a scale of 1-10 and another with a scale of 1,000-100,000, PCA will be completely dominated by the second feature. You must scale your data (usually with `StandardScaler`) before applying PCA to ensure all features contribute fairly.

features contribute fairly.

Q: Can I use t-SNE for feature extraction before training a classifier?

A: It is generally not recommended. t-SNE does not have a `transform` method for new data, it is computationally expensive, and it's optimized for visualization, not for creating components that preserve the separability needed for a classifier. Use PCA or another feature extraction method for that purpose.

Date _____ Page _____

Date _____ Page _____

🏛️ Week 5: Feature Engineering for Text Data

Welcome back! This week, we pivot from structured numerical data to the fascinating and unstructured world of text. How do we make a machine, which only understands numbers, comprehend human language? The answer lies in transforming words into meaningful numerical features. This is one ofthe most crucial skills in Natural Language Processing (NLP).

Key Topics

1. Challenges of Representing Textual Data Numerically

Machine learning models are mathematical functions; they require numerical input. Raw text, with its vast vocabulary, sarcasm, synonyms, and grammatical nuances, presents several challenges:

- **High Dimensionality:** The vocabulary of a language can be enormous. If we create a unique feature for every word, we end up with tens of thousands of dimensions, leading to the "Curse of Dimensionality."
- **No Inherent Order:** Unlike numbers, words don't have a natural ordering. Is "apple" > "ball"? This makes direct numerical conversion tricky.
- **Context and Semantics:** The meaning of a word often depends on its context. "Bank" can mean a financial institution or a river's edge. Simple numerical representations often lose this semantic richness.

2. Basic Text Feature Engineering Techniques

Let's start with the classic, foundational methods.

- **Unigrams, N-grams:** An N-gram is a contiguous sequence of 'n' items from a sample of text.
  - **Unigram (1-gram):** A single word. (e.g., "The", "quick", "brown", "fox").
  - **Bigram (2-gram):** A pair of consecutive words. (e.g., "The quick", "quick brown", "brown fox").
  - **Trigram (3-gram):** A triplet of consecutive words. (e.g., "The quick brown", "quick brown fox").
    N-grams help capture some context. For instance, "not good" as a bigram has a very different meaning than the unigrams "not" and "good" considered separately.
- **Bag of Words (BoW):** This is the most straightforward vectorization technique. It represents text by counting the occurrence of each word, disregarding grammar and word order.
  1. **Create a Vocabulary:** Collect all unique words from the entire corpus of documents.
  2. **Create a Vector:** For each document, create a vector the size of the vocabulary. Each element in the vector corresponds to a word in the vocabulary.
  3. **Count Occurrences:** The value of each element is the number of times the corresponding word appears in the document.

Example:

- Doc 1: "the cat sat on the mat"
- Doc 2: "the dog sat on the log"
- **Vocabulary:** {"the", "cat", "sat", "on", "mat", "dog", "log"}
- **Vector for Doc 1:** `[2, 1, 1, 1, 1, 0, 0]`
- **Vector for Doc 2:** `[2, 0, 1, 1, 0, 1, 1]`

- **Term Frequency-Inverse Document Frequency (TF-IDF):** BoW has a flaw: common words like "the" or "a" dominate the counts but often carry little unique information. TF-IDF addresses this by weighting words based on how important they are to a document in a collection.

  - **Term Frequency (TF):** Measures how frequently a term appears in a document.
    $$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$
  - **Inverse Document Frequency (IDF):** Measures how important a term is. It penalizes common words that appear in many documents.
    $$IDF(t, D) = \log\left(\frac{\text{Total number of documents } D}{\text{Number of documents with term } t}\right)$$
  - **TF-IDF Score:** The product of TF and IDF.
    $$TF - IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

  Words with a high TF-IDF score are frequent in a specific document but rare across the entire corpus, making them more significant.

3. Advanced Text Representations

- **One-Hot Encoding:** This technique is more suitable for categorical variables with a small number of unique values (e.g., 'country' names). For text with a large vocabulary, it creates extremely sparse and high-dimensional vectors, making it inefficient. Each unique word becomes a column in a giant matrix, with a `1` indicating its presence in a sentence and `0` otherwise.
- **Word Embeddings (Conceptual Overview):** While BoW and TF-IDF are great, they treat words as independent units. Word embeddings are dense vector representations where words with similar meanings have similar vector representations. Models like **Word2Vec** and **GloVe** learn these embeddings from large text corpora. For example, the vector for "king" might be close to the vector for "queen" in this high-dimensional space. This technique captures semantic relationships, which is a massive leap forward.

4. Code Implementation: BoW and TF-IDF

Let's see how to implement these with `scikit-learn`.

Python

```
import pandas as pd
```

```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Sample Data [cite: 96]
data = {
    'text': [
        "This is a great movie, I loved it",
        "The food was terrible, a bad experience",
        "I absolutely enjoyed the film",
        "I will not recommend this restaurant",
        "What a fantastic movie!",
        "The service was slow and the food was cold"
    ],
    'sentiment': ['positive', 'negative', 'positive', 'negative', 'positive', 'nega
}
df = pd.DataFrame(data)

# --- 1. Bag of Words (BoW) ---
print("--- Bag of Words (BoW) Example ---")
# Initialize the vectorizer (this also handles tokenization and lowercasing)
# We can also specify n-grams, e.g., ngram_range=(1, 2) for unigrams and bigrams
bow_vectorizer = CountVectorizer()
X_bow = bow_vectorizer.fit_transform(df['text'])

# See the feature names (the vocabulary)
print("Vocabulary:", bow_vectorizer.get_feature_names_out())
# See the sparse matrix representation
print("BoW Matrix:\n", X_bow.toarray())


# --- 2. TF-IDF ---
print("\n--- TF-IDF Example ---")
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(df['text'])

# See the TF-IDF matrix
print("TF-IDF Matrix:\n", pd.DataFrame(X_tfidf.toarray(), columns=tfidf_vectorizer


# --- 3. Evaluating in a Classification Task --- [cite: 96]
print("\n--- Classification Example ---")
# Using TF-IDF features as they are generally better
X = X_tfidf
y = df['sentiment']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_st

# Train a simple model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions and evaluate
predictions = model.predict(X_test)
```

```
print(f"Accuracy on test set: {accuracy_score(y_test, predictions):.2f}")
```

5. Part-of-Speech (POS) Tagging

POS tagging identifies the grammatical part of speech of each word (e.g., noun, verb, adjective). This can be a powerful feature. For example, product reviews with more adjectives might be more opinionated.

Example Features from POS:

- Count of nouns, verbs, adjectives per document.
- Ratio of nouns to verbs.
- Frequency of specific POS n-grams (e.g., adjective-noun pairs like "good food").

Let's see a quick example using the `nltk` library.

Python

```
import nltk
# You might need to download these first:
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')

text = "The quick brown fox jumps over the lazy dog."
tokens = nltk.word_tokenize(text)
pos_tags = nltk.pos_tag(tokens)

print(pos_tags)
# Output: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps
# DT: Determiner, JJ: Adjective, NN: Noun, VBZ: Verb
```

Comparison of Text Vectorization Techniques

Technique	How it Works	Pros	Cons	Best For
Bag of Words	Counts word occurrences.	Simple, fast, provides a baseline.	Ignores word order & context, high-dimensionality, biased towards common words.	Initial analysis, text classification where word counts are important.
TF-IDF	Weights words by frequency and rarity.	Reduces the impact of common words, simple to implement.	Still ignores word order & context, struggles with synonyms.	Document search, information retrieval, text classification.
Word Embeddings	Dense vectors representing semantic meaning.	Captures context & relationships, lower dimensionality.	Computationally expensive to train, requires large datasets.	Sentiment analysis, language translation, any task needing semantic

understanding.

Export to Sheets

❓ Week 5 Q&A

- **Q: Why is TF-IDF often better than Bag of Words?**
  - **A:** TF-IDF gives more weight to words that are meaningful for a specific document. A word like "the" will have a high count in BoW for almost every document, making it seem important. TF-IDF recognizes it appears everywhere and reduces its weight, allowing more unique and descriptive words to stand out.
- **Q: When would I use N-grams instead of just single words (unigrams)?**
  - **A:** Use n-grams when word order and context are important. For example, in sentiment analysis, "not happy" is a bigram whose meaning is completely lost if you only look at the unigrams "not" and "happy" separately.
- **Q: Are word embeddings always the best choice?**
  - **A:** Not necessarily. They are computationally intensive and might be overkill for simpler problems where TF-IDF performs just as well. For tasks like document classification based on specific keywords, TF-IDF can be more effective and is much faster to implement.

📝 Week 5 Quiz

1. What is the primary drawback of the Bag of Words (BoW) model?
   a) It is computationally expensive.
   b) It loses information about word order and grammar.
   c) It cannot handle large vocabularies.
   d) It gives too much importance to rare words.
2. A TF-IDF score is highest for a word that appears...
   a) Many times in a document and many times in the overall corpus.
   b) Few times in a document and many times in the overall corpus.
   c) Many times in a document and few times in the overall corpus.
   d) Few times in a document and few times in the overall corpus.
3. If you want to capture the phrase "climate change" as a single feature in your text analysis, what technique would you use?
   a) Unigrams
   b) Stop word removal
   c) Bigrams
   d) TF-IDF
4. What is the main advantage of word embeddings over TF-IDF?

4. What is the main advantage of word embeddings over TF-IDF?
   a) They are faster to compute.
   b) They capture the semantic meaning and context of words.
   c) They create sparse vectors.
   d) They work better for short documents.

(Quiz Answers: 1-b, 2-c, 3-c, 4-b)

🖼️ Week 6: Feature Engineering for Image Data

Great work on text! Now, let's switch our senses from reading to seeing. Images are just grids of pixels to a computer, so how do we extract meaningful features that describe their content? This week, we'll learn to create features that represent shapes, textures, corners, and edges.

Key Topics

1. Fundamentals of Image Representation

An image is a matrix (or a 3D tensor) of pixel values.

- **Grayscale Image:** A 2D matrix where each pixel value represents its intensity (typically 0 for black to 255 for white).
- **Color (RGB) Image:** A 3D matrix (height x width x channels). It consists of three 2D matrices stacked together, one for each color channel: Red, Green, and Blue.

Machine learning models can't work with the image matrix directly. We need to extract descriptive features from these pixel values. Using raw pixel intensities as features is often a bad idea due to high dimensionality and sensitivity to small shifts or rotations.

2. Edge Detection Techniques

Edges are one of the most fundamental features in an image. They represent boundaries between objects or regions and signify areas of rapid change in pixel intensity.

- **Sobel Operator:** Calculates the gradient of the image intensity at each point, highlighting the rate of change. It produces two images: one for horizontal edges and one for vertical edges.
- **Canny Edge Detector:** A multi-step, popular algorithm that produces clean, thin edges.
  1. **Noise Reduction:** Apply a Gaussian filter to smooth the image.
  2. **Gradient Calculation:** Find intensity gradients (like Sobel).
  3. **Non-maximum Suppression:** Thin the edges to single-pixel width.
  4. **Hysteresis Thresholding:** Use two thresholds (high and low) to connect strong edges and eliminate weak ones. This makes it less susceptible to noise than simpler methods.

3. Corner Detection Methods

Corners are points where two edges intersect. They are highly informative because they are stable features that remain recognizable even if the image is rotated or scaled.

- **Harris Corner Detector:** A classic algorithm that identifies corners by looking at the changes in intensity that result from shifting a small window in all directions. A large change in all directions indicates a corner.
- **Shi-Tomasi Corner Detector:** A slight modification of the Harris detector. It often gives better results by changing the scoring function, making it useful for tracking features in videos.

4. Histogram of Oriented Gradients (HOG)

HOG is a powerful feature descriptor used for object detection. It captures the shape and structure of an object by looking at the distribution of gradient orientations within local regions of an image.

How HOG Works:

1. **Divide the Image:** Split the image into small connected regions called "cells."
2. **Calculate Gradients:** For each pixel in a cell, compute the gradient magnitude and orientation (direction).
3. **Create Cell Histograms:** For each cell, create a histogram of the gradient orientations. The bins of the histogram correspond to orientation ranges (e.g., 0-20 degrees, 20-40 degrees, etc.).
4. **Normalize Blocks:** Group several cells together into larger "blocks." Normalize the histograms within each block to make the descriptor robust to changes in lighting and contrast.
5. **Concatenate:** Concatenate all the normalized block histograms to form the final HOG feature vector for the entire image.

This final vector can then be fed into a classification model like a Support Vector Machine (SVM) to detect objects like pedestrians or cars.

5. Code Implementation: Edges, Corners, and HOG

Let's use `scikit-image` to demonstrate these concepts.

Python

```
import matplotlib.pyplot as plt
from skimage import data, color, feature
from skimage.transform import resize

# Load a sample image [cite: 116]
image = data.astronaut()
image_gray = color.rgb2gray(image)
# Resize for faster processing
```

```
# Resize for faster processing
image_gray = resize(image_gray, (400, 400))


# --- 1. Edge Detection (Canny) ---
edges = feature.canny(image_gray, sigma=3)


# --- 2. Corner Detection (Harris) ---
# compute_harris_response returns a corner measure image
coords = feature.corner_peaks(feature.corner_harris(image_gray), min_distance=5, th
# corner_shi_tomasi is also available


# --- 3. Histogram of Oriented Gradients (HOG) ---
fd, hog_image = feature.hog(image_gray, orientations=8, pixels_per_cell=(16, 16),
                            cells_per_block=(1, 1), visualize=True)


# --- Visualization --- [cite: 116]
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
ax = axes.ravel()

ax[0].imshow(image_gray, cmap=plt.cm.gray)
ax[0].set_title('Grayscale Image')

ax[1].imshow(edges, cmap=plt.cm.gray)
ax[1].set_title('Canny Edges')

ax[2].imshow(image_gray, cmap=plt.cm.gray)
ax[2].plot(coords[:, 1], coords[:, 0], '+r', markersize=10) # Plot corners
ax[2].set_title('Harris Corners')

ax[3].imshow(hog_image, cmap=plt.cm.gray)
ax[3].set_title('HOG Descriptor')

for a in ax:
    a.set_axis_off()

plt.tight_layout()
plt.show()

print(f"Shape of the HOG feature vector: {fd.shape}")
# This vector can now be used as input to a machine learning model.
```

❓ Week 6 Q&A

- Q: Why not just use the raw pixel values as features?
  - **A:** Raw pixels are very sensitive to small changes. Shifting an object by one pixel creates a completely different feature vector. They also result in extremely high dimensionality (e.g., a 100x100 image has 10,000 features). Feature descriptors like HOG are more robust to these changes and provide a more compact, meaningful representation.

- **Q: What is the difference between an edge and a corner?**
  - **A:** An edge is a 1D boundary where image intensity changes. A corner is a 0D point where two or more edges meet. Corners are more stable features for matching and tracking because they are localized in two directions.
- **Q: Can I use HOG for any image task?**
  - **A:** HOG is excellent for tasks where object shape and structure are important, like pedestrian or vehicle detection. It is less effective for tasks that rely on color or fine texture, as it operates on grayscale gradients. For those tasks, other features (or deep learning models) might be more suitable.

📝 Week 6 Quiz

1. What does the Canny edge detector's "hysteresis thresholding" step help with?
   a) Smoothing the image to reduce noise.
   b) Making edges thicker.
   c) Connecting strong edges while removing weak, isolated ones.
   d) Calculating the direction of the gradient.
2. A corner is a good feature to track because it is...
   a) Resistant to changes in lighting.
   b) Distinct and localized in two dimensions.
   c) Always present on every object.
   d) Easy to describe with a single number.
3. The HOG descriptor primarily captures information about an object's:
   a) Color
   b) Fine texture
   c) Shape and structure
   d) Size
4. In the HOG algorithm, what is the purpose of "block normalization"?
   a) To increase the dimensionality of the feature vector.
   b) To make the descriptor more robust to changes in lighting and contrast.
   c) To divide the image into smaller cells.
   d) To find the orientation of the gradients.

(Quiz Answers: 1-c, 2-b, 3-c, 4-b)

📈 Week 7: Feature Engineering for Time-Series Data

We've covered text and images; now we'll tackle data that evolves over time. Time-series data is a sequence of data points indexed in time order. The key challenge and opportunity here is the **temporal dependence**—the past often influences the future. Our features must capture these patterns.

Key Topics

1. Challenges and Approaches in Time-Series Feature Extraction

Unlike standard tabular data, time-series data has an explicit order. Shuffling the data points destroys the information. Our goal is to convert a sequence of measurements into a set of features that a standard machine learning model (like a regression or classification model) can use. This is often done by creating features from a "window" of time.

2. Creating Window-Based Features

This is the most common approach. We use a sliding window over the time series to extract features.

- **Lag Features:** These are the values from previous time steps. A "lag 1" feature for today would be the value from yesterday. A "lag 7" feature would be the value from a week ago. Lags are crucial for capturing autocorrelation and seasonality.
- **Rolling Statistics:** These are statistics calculated on a moving window.
  - **Rolling Mean/Average:** Smooths out short-term fluctuations and highlights longer-term trends.
  - **Rolling Standard Deviation:** Measures volatility or stability over a period.
  - Other stats: Rolling `min`, `max`, `sum`, `median`, etc.

3. Domain-Specific Techniques

The most powerful features often come from domain knowledge.

- **Finance:**
  - **Moving Averages (MA):** Simple, Exponential (EMA). Used to identify trends.
  - **Relative Strength Index (RSI):** A momentum oscillator that measures the speed and change of price movements, indicating overbought or oversold conditions.
  - **Bollinger Bands:** Measures volatility by plotting bands two standard deviations away from a simple moving average.
- **Medical Datasets (e.g., ECG):**
  - **Heart Rate Variability (HRV):** The variation in time between consecutive heartbeats. Features can include the standard deviation of beat-to-beat intervals, which is a powerful indicator of cardiac health.

4. Code Implementation: Lag and Rolling Features

Let's see how to create these essential features using `pandas`. We'll use a simple stock price dataset.

Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Create a sample time-series dataframe
dates = pd.date_range(start='2024-01-01', periods=100, freq='D')
data = np.random.randn(100).cumsum() + 50 # Create some random walk data
ts = pd.DataFrame({'price': data}, index=dates)

plt.style.use('seaborn-v0_8-whitegrid')

# --- 1. Lag Features ---
# Creating a lag of 1 day and 7 days (weekly effect)
ts['lag_1'] = ts['price'].shift(1)
ts['lag_7'] = ts['price'].shift(7)

# --- 2. Rolling Statistics ---
# Rolling mean over a 7-day window
ts['rolling_mean_7'] = ts['price'].rolling(window=7).mean()
# Rolling standard deviation over a 7-day window
ts['rolling_std_7'] = ts['price'].rolling(window=7).std()

# --- 3. Evaluating the Effect of Engineered Features --- [cite: 133]
# Let's visualize the new features
fig, ax = plt.subplots(figsize=(14, 7))
ts[['price', 'rolling_mean_7']].plot(ax=ax)
# Plot the volatility using Bollinger Bands concept
ax.fill_between(ts.index,
                ts['rolling_mean_7'] - 2 * ts['rolling_std_7'],
                ts['rolling_mean_7'] + 2 * ts['rolling_std_7'],
                color='gray', alpha=0.2, label='Bollinger Bands (2*std)')
ax.set_title('Price with Rolling Mean and Volatility')
ax.legend()
plt.show()

# Display the dataframe with the new features
# Note the NaNs at the beginning where the window is not yet full
print(ts.head(10))

# To use these features in a model, we would typically drop the NaN rows.
# For a forecasting task, the 'price' would be the target (y),
# and the lag/rolling features would be the predictors (X).
ts_clean = ts.dropna()
print("\nCleaned DataFrame for Modeling:")
print(ts_clean.head())
```

This code transforms our single-column time series into a multi-column dataset suitable for a supervised learning model, effectively converting a forecasting problem into a standard regression problem.

❓ **Week 7 Q&A**

- Q: What's the difference between a rolling window and an expanding window?
  - **A:** A rolling window has a fixed size (e.g., the last 7 days). As it moves forward, it drops the oldest data point and adds the newest one. An expanding window starts at the beginning of the series and grows to include all data up to the current point. Rolling windows are better for capturing recent trends, while expanding windows capture statistics over the entire history.
- **Q: How do I choose the right window size for rolling features?**
  - **A:** This depends on the data's seasonality and characteristics. If you have daily data with a weekly pattern, a window size of 7 makes sense. For monthly patterns, 30 might work. It's often a process of experimentation and domain knowledge. You can try multiple window sizes and let the model's feature importance tell you which ones are most predictive.
- **Q: Can I use features from the future?**
  - **A:** Absolutely not! This is called **data leakage** and is a critical mistake in time-series analysis. When making a prediction for a specific time point, your features must only be created from information available *up to that time point*. Using `shift(-1)` would incorrectly use tomorrow's value to predict today's value.

📝 Week 7 Quiz

1. To predict tomorrow's stock price, which of the following is a valid "lag feature"?
   a) The stock price from two days from now.
   b) The average stock price over the next week.
   c) The stock price from yesterday.
   d) The future company earnings announcement.
2. What is the main purpose of using a rolling mean feature?
   a) To measure the volatility of the series.
   b) To smooth out short-term noise and identify the underlying trend.
   c) To capture weekly seasonality.
   d) To make the data stationary.
3. In a time-series dataset, why is it important to not shuffle the data before splitting it into training and testing sets?
   a) It makes the computations slower.
   b) It would destroy the temporal order, leading to data leakage.
   c) It is not important; shuffling is always a good practice.
   d) It only matters for financial data.
4. RSI and Bollinger Bands are examples of what kind of features?
   a) General statistical features
   b) Domain-specific features from finance
   c) Lag features

d) Rolling window features

(Quiz Answers: 1-c, 2-b, 3-b, 4-b)

🤖 Week 8: Automated Feature Engineering

You've worked hard learning to manually craft features for different data types. This is an art that requires domain expertise and creativity. But what if we could automate this process? This week, we explore tools that can automatically generate hundreds or even thousands of candidate features, saving time and potentially discovering patterns we might have missed.

Key Topics

1. Benefits and Limitations of Automated Feature Engineering (AutoFE)

- **Benefits:**
  - **Speed:** Drastically reduces the time spent on manual feature creation.
  - **Discovery:** Can uncover complex, non-intuitive feature interactions.
  - **Performance:** Often leads to better model performance by exploring a vast feature space.
- **Limitations:**
  - **Computational Cost:** Generating thousands of features can be very memory and CPU intensive.
  - **Black Box:** The resulting features can be complex and hard to interpret ("ratio of the log of the sum of...").
  - **Overfitting:** A huge number of features increases the risk of finding spurious correlations that don't generalize to new data.
  - **Lack of Domain Knowledge:** Automated tools cannot invent domain-specific features like RSI without being explicitly told the formula.

2. Deep Feature Synthesis with Featuretools

Featuretools is a powerful library for AutoFE on **relational and temporal data**. Its core algorithm is **Deep Feature Synthesis (DFS)**.

Key Concepts:

- **Entities:** Essentially a `pandas` DataFrame.
- **Relationships:** The links between your entities (e.g., a `customers` entity is linked to a `transactions` entity via `customer_id`).
- **Feature Primitives:** The building blocks of features.
  - **Aggregation Primitives:** Operate across related entities (e.g., `SUM`, `MEAN`, `COUNT` of a customer's transactions).

- **Transform Primitives:** Operate on one or more columns within a single entity (e.g., `DAY` of a datetime column, `ADD` two numeric columns).

DFS works by stacking these primitives to create "deep" features. For example, it could create `MEAN(transactions.amount)` for each customer (depth 1), and then `WEEKDAY(customer.join_date)` (depth 1), and then even combine them into more complex features.

3. Automated Time-Series Feature Extraction with TSFresh

TSFresh is a library specifically designed for time-series data. It automatically extracts a huge number of features (over 750!) from time series. These features range from simple (mean, max) to complex (Fourier coefficients, autocorrelation). Crucially, it also includes built-in methods to filter these features based on their relevance to the target variable, helping to control for overfitting.

4. Feature Selection with Featurewiz

Featurewiz is a library that automates feature selection and preprocessing. While not strictly a feature *generation* tool like Featuretools, it's a key part of the automated workflow. After you've generated a large number of features (manually or automatically), Featurewiz can rapidly identify the most useful subset using advanced techniques, making your model training faster and more robust.

5. Code Implementation: Featuretools Example

Let's see the power of Featuretools on a simple relational dataset.

Python

```python
import featuretools as ft
import pandas as pd

# --- Create Sample Data ---
# We have two tables (entities): customers and their transactions
customers_df = pd.DataFrame({
    "customer_id": [1, 2, 3],
    "join_date": pd.to_datetime(["2020-01-15", "2020-02-10", "2020-03-01"]),
    "is_premium": [True, False, False]
})

transactions_df = pd.DataFrame({
    "transaction_id": range(6),
    "customer_id": [1, 1, 2, 2, 3, 1],
    "transaction_time": pd.to_datetime(["2020-01-20", "2020-01-25", "2020-02-12", '
    "amount": [100, 50, 200, 75, 150, 25]
})

# --- Use Featuretools --- [cite: 147, 150]
```

```
# 1. Create an EntitySet (a container for entities and relationships)
es = ft.EntitySet(id="customer_data")

# 2. Add entities (DataFrames)
es = es.add_dataframe(dataframe_name="customers", dataframe=customers_df, index="cu
es = es.add_dataframe(dataframe_name="transactions", dataframe=transactions_df, inc

# 3. Define the relationship between them
es = es.add_relationship("customers", "customer_id", "transactions", "customer_id")

# 4. Run Deep Feature Synthesis (DFS)!
# We want to create features for the 'customers' table.
# This will automatically create aggregations from the 'transactions' table
# and transforms for the 'customers' table.
feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_dataframe_name="customers",
                                      verbose=1)

# Display the powerful, automatically generated features
print("\n--- Automatically Generated Feature Matrix ---")
print(feature_matrix)

print("\n--- List of Generated Features ---")
for f in feature_defs:
    print(f)
```

Look at the output! Featuretools automatically created features like `SUM(transactions.amount)`, `DAY(join_date)`, `NUM_UNIQUE(transactions.transaction_id)`, and many more, all without us having to write a single loop or groupby statement.

Comparison of AutoFE Tools

Tool	Primary Use Case	How it Works	Key Strength
Featuretools	Relational (multi-table) and time-series data.	Applies "primitives" (aggregations, transforms) across tables using Deep Feature Synthesis.	Excellent at uncovering complex interactions in structured, relational data.
TSFresh	Time-series data.	Extracts a massive, comprehensive library of time-series characteristics.	Huge library of pre-built features and built-in relevance filtering.
Featurewiz	Feature selection and preprocessing.	Uses advanced algorithms (like SULOV) to find the most predictive features from a given set.	Speed and effectiveness in reducing a large feature set to a minimal, powerful one.

Export to Sheets

❓ Week 8 Q&A

- **Q: Should I stop doing manual feature engineering and only use these tools?**
  - **A:** No. The best approach is often a hybrid one. Use your domain knowledge to create a set of strong, interpretable manual features. Then, use an AutoFE tool to generate more candidate features. Finally, use a tool like Featurewiz or manual feature selection to pick the best combination. AutoFE is a powerful assistant, not a replacement for human insight.
- **Q: The features from Featuretools are very complex. How do I explain them?**
  - **A:** This is a major challenge. You can often look at the feature's definition (e.g., `MEAN(transactions.amount)`) to understand what it represents. For very "deep" features, it can be difficult. If interpretability is the top priority for your project, you might need to stick to simpler, manually created features.
- **Q: Are these tools difficult to set up?**
  - **A:** Not at all! All three libraries (`featuretools`, `tsfresh`, `featurewiz`) are easily installable via `pip`. Their basic usage, as shown in the example, is quite straightforward, though they have many advanced configuration options for complex use cases.

📝 Week 8 Quiz

1. What is the core algorithm behind the Featuretools library?
   a) Histogram of Oriented Gradients (HOG)
   b) Term Frequency-Inverse Document Frequency (TF-IDF)
   c) Deep Feature Synthesis (DFS)
   d) Principal Component Analysis (PCA)
2. Which of the following is a significant *limitation* of automated feature engineering?
   a) It only works on text data.
   b) The generated features can be difficult to interpret.
   c) It is always faster than manual feature engineering.
   d) It reduces the risk of model overfitting.
3. You have a single, long time-series of sensor data and you want to extract as many characteristic features as possible. Which tool is specifically designed for this task?
   a) Featuretools
   b) TSFresh
   c) Scikit-image
   d) NLTK
4. In Featuretools, what is a "primitive"?
   a) A raw dataset.
   b) The final feature matrix.
   c) A basic operation, like `SUM` or `WEEKDAY`, that is used to build features.
   d) The relationship between two tables.

(Quiz Answers: 1-c, 2-b, 3-b, 4-c)

(Quiz Answers: 1-c, 2-b, 3-b, 4-c)

Date _____ Page _____

Date _____ Page _____