05_simulated_feature_drift_ai4i

You are helping with an academic ML research experiment. This notebook simulates controlled feature drift on AI4I 2020 and quantifies impact on model performance.

In [6]:
```
# Standard imports and plotting configuration
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from sklearn.model_selection import train_test_split
from scipy.stats import ks_2samp
import warnings
warnings.filterwarnings('ignore')
sns.set(style='whitegrid')
```

In [7]:
```
# 1) Load dataset
path = os.path.join('..','data','ai4i','ai4i2020.csv')
df = pd.read_csv(path)
print('Loaded', df.shape)
df.head()
```

```
Loaded (10000, 14)
```

Out[7]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0	0

In [8]:
```
# 2) Clean column names and keep numeric features
# normalize to snake_case: lower, replace non-alnum with underscore
df.columns = df.columns.str.strip().str.lower().str.replace(r'[^0-9a-z]+','_', regex=True).str.strip('_')
# keep numeric columns + identifier columns for ordering
numeric_df = df.select_dtypes(include=[np.number]).copy()
print('Numeric shape:', numeric_df.shape)
numeric_df.columns[:10]
```

```
Numeric shape: (10000, 12)
```

Out[8]:
```
Index(['udi', 'air_temperature_k', 'process_temperature_k',
       'rotational_speed_rpm', 'torque_nm', 'tool_wear_min', 'machine_failure',
       'twf', 'hdf', 'pwf'],
      dtype='object')
```

In [9]:
```
# 3) Use `machine_failure` as the target variable
if 'machine_failure' not in df.columns:
    raise RuntimeError('Expected column `machine_failure` not found after cleaning')
# Guard against duplicate column names which can cause sort_values errors
if df.columns.duplicated().any():
    # keep first occurrence of duplicated column names to avoid ambiguous labels
    dup = df.columns[df.columns.duplicated()].tolist()
    print('Warning: duplicate columns found and removed (keeping first):', dup)
    df = df.loc[:, ~df.columns.duplicated()]
# build dataset with only numeric features + target + ordering id (udi)
data = df.copy()
# ensure target is numeric
data['machine_failure'] = data['machine_failure'].astype(int)
# drop non-numeric columns except UDI (for time-based split)
if 'udi' in data.columns:
    order_col = 'udi'
else:
    order_col = data.columns[0]
# keep only numeric features and the target and order_col
numeric_cols = data.select_dtypes(include=[np.number]).columns.tolist()
keep_cols = [c for c in numeric_cols if c != 'machine_failure'] + ['machine_failure', order_col]
```

```
data = data[keep_cols].dropna(subset=['machine_failure'])
print('Prepared data shape:', data.shape)
```

```
Prepared data shape: (10000, 13)
```

In [10]:
```
# 4) Time-based split: first 60% train, last 40% test using `order_col`
# Ensure there are no duplicate column labels on `data` (can occur after selects); keep first occurrence
if data.columns.duplicated().any():
    dup = data.columns[data.columns.duplicated()].tolist()
    print('Warning: duplicate columns in `data` found and removed (keeping first):', dup)
    data = data.loc[:, ~data.columns.duplicated()]
data = data.sort_values(order_col).reset_index(drop=True)
n = len(data)
train_end = int(0.6 * n)
train = data.iloc[:train_end].copy()
test = data.iloc[train_end:].copy()
X_train = train.drop(columns=['machine_failure', order_col])
y_train = train['machine_failure']
X_test = test.drop(columns=['machine_failure', order_col])
y_test = test['machine_failure']
print('Train/test sizes', X_train.shape, X_test.shape)
```

```
Warning: duplicate columns in `data` found and removed (keeping first): ['udi']
Train/test sizes (6000, 10) (4000, 10)
```

In [11]:
```
# 5) Train baseline RandomForestClassifier with fixed hyperparameters
RANDOM_STATE = 42
clf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=RANDOM_STATE, n_jobs=-1)
clf.fit(X_train, y_train)
# 6) Evaluate baseline metrics: Accuracy, F1, ROC-AUC
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)[:,1] if hasattr(clf, 'predict_proba') else None
baseline_acc = accuracy_score(y_test, y_pred)
baseline_f1 = f1_score(y_test, y_pred, zero_division=0)
baseline_roc = roc_auc_score(y_test, y_proba) if y_proba is not None and len(np.unique(y_test))==2 else np.nan
print('Baseline — Accuracy: {:.4f}, F1: {:.4f}, ROC-AUC: {:.4f}'.format(baseline_acc, baseline_f1, baseline_roc))
```

```
Baseline — Accuracy: 0.9992, F1: 0.9818, ROC-AUC: 0.9837
```

In [12]:
```
# 7) Identify top 2 drifting features from prior KS analysis (expected names after cleaning)
drift_feats = []
for f in ['air_temperature_k','process_temperature_k']:
    if f in X_train.columns:
        drift_feats.append(f)
if len(drift_feats) < 2:
    raise RuntimeError('Expected drift features not found in numeric columns: {}'.format(X_train.columns[:20]))
print('Drift features:', drift_feats)
```

```
Drift features: ['air_temperature_k', 'process_temperature_k']
```

In [13]:
```
# 8) Drift scenarios on the test set
def apply_gaussian(X, feats, sigma_factor, random_state=RANDOM_STATE):
    Xp = X.copy()
    rng = np.random.RandomState(random_state)
    for f in feats:
        # use relative sigma to preserve units: sigma_factor * feature_std
        sigma = Xp[f].std() * sigma_factor
        Xp[f] = Xp[f] + rng.normal(0, sigma, size=len(Xp))
    return Xp

def apply_scale(X, feats, scale_factor):
    Xp = X.copy()
    for f in feats:
        Xp[f] = Xp[f] * scale_factor
    return Xp

# define scenarios and human-readable magnitude for plotting
scenarios = []
# mild and moderate gaussian (sigma factors)
scenarios.append({'name':'mild_gauss','type':'gauss','param':0.5})
scenarios.append({'name':'moderate_gauss','type':'gauss','param':1.5})
# severe scaling factors
for s in [1.1,1.2,1.3]:
    scenarios.append({'name':f'scale_{s}','type':'scale','param':s})

results = []
for sc in scenarios:
    if sc['type']=='gauss':
        X_test_d = apply_gaussian(X_test, drift_feats, sigma_factor=sc['param'])
        magnitude = sc['param']
    else:
        X_test_d = apply_scale(X_test, drift_feats, scale_factor=sc['param'])
```

```
        # interpret magnitude as relative change (e.g., 1.2 -> 0.2)
        magnitude = sc['param'] - 1.0
    # Evaluate model on drifted test set
    y_pred_d = clf.predict(X_test_d)
    y_proba_d = clf.predict_proba(X_test_d)[:,1] if hasattr(clf, 'predict_proba') else None
    acc_d = accuracy_score(y_test, y_pred_d)
    f1_d = f1_score(y_test, y_pred_d, zero_division=0)
    roc_d = roc_auc_score(y_test, y_proba_d) if y_proba_d is not None and len(np.unique(y_test))==2 else np.nan
    # KS statistics for each drift feature between original test and drifted test
    ks_stats = {}
    for f in drift_feats:
        ks = ks_2samp(X_test[f].values, X_test_d[f].values).statistic
        ks_stats[f] = ks
    avg_ks = np.mean(list(ks_stats.values()))
    results.append({
        'scenario': sc['name'], 'type': sc['type'], 'param': sc['param'], 'magnitude': magnitude,
        'acc': acc_d, 'f1': f1_d, 'roc_auc': roc_d,
        'avg_ks': avg_ks, 'ks_details': ks_stats, 'acc_drop': baseline_acc - acc_d
    })

res_df = pd.DataFrame(results)
res_df
```

Out[13]:

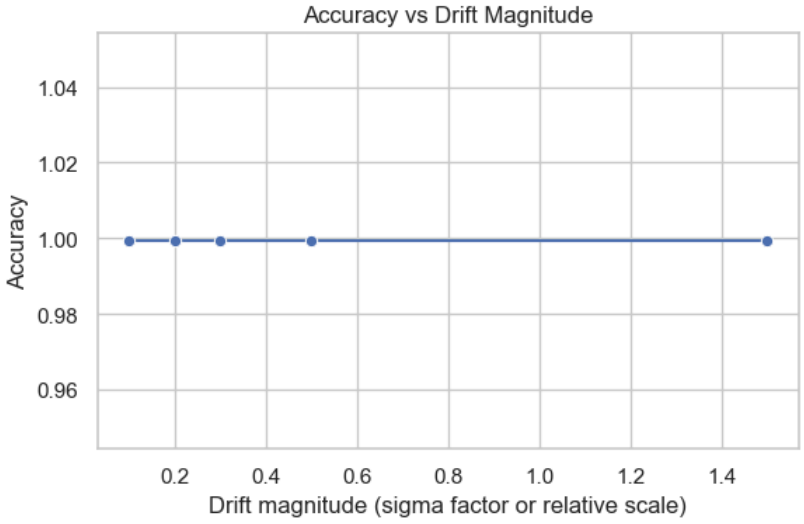
	scenario	type	param	magnitude	acc	f1	roc_auc	avg_ks	ks_details	acc_drop
0	mild_gauss	gauss	0.5	0.5	0.99925	0.981818	0.985487	0.06425	{'air_temperature_k': 0.0845, 'process_tempera...	0.0
1	moderate_gauss	gauss	1.5	1.5	0.99925	0.981818	0.988551	0.17425	{'air_temperature_k': 0.19775, 'process_temper...	0.0
2	scale_1.1	scale	1.1	0.1	0.99925	0.981818	0.989203	1.00000	{'air_temperature_k': 1.0, 'process_temperatur...	0.0
3	scale_1.2	scale	1.2	0.2	0.99925	0.981818	0.989203	1.00000	{'air_temperature_k': 1.0, 'process_temperatur...	0.0
4	scale_1.3	scale	1.3	0.3	0.99925	0.981818	0.989203	1.00000	{'air_temperature_k': 1.0, 'process_temperatur...	0.0

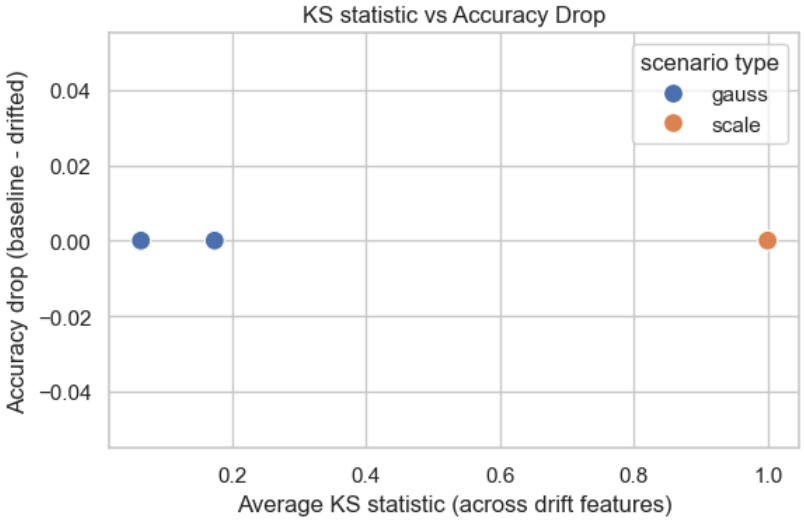
In [14]:

```
# 10) Analysis & Plots — Accuracy vs drift magnitude, KS vs accuracy drop
os.makedirs(os.path.join('..','results','figures'), exist_ok=True)
# prepare plotting dataframe: use magnitude and avg_ks as x-axes depending on scenario type
plot_df = res_df.copy()
# For plotting combine gaussian and scale magnitudes into a single numeric column
plot_df['drift_magnitude'] = plot_df['magnitude']
# Accuracy vs drift magnitude
plt.figure(figsize=(6,4))
sns.lineplot(data=plot_df, x='drift_magnitude', y='acc', marker='o')
plt.xlabel('Drift magnitude (sigma factor or relative scale)')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Drift Magnitude')
plt.tight_layout()
plt.savefig(os.path.join('..','results','figures','accuracy_vs_drift_magnitude.png'), dpi=300)
plt.show()

# KS statistic vs accuracy drop
plt.figure(figsize=(6,4))
sns.scatterplot(data=plot_df, x='avg_ks', y='acc_drop', hue='type', s=100)
plt.xlabel('Average KS statistic (across drift features)')
plt.ylabel('Accuracy drop (baseline - drifted)')
plt.title('KS statistic vs Accuracy Drop')
plt.legend(title='scenario type')
plt.tight_layout()
plt.savefig(os.path.join('..','results','figures','ks_vs_accuracy_drop.png'), dpi=300)
plt.show()

# Save result table for paper use
os.makedirs(os.path.join('..','results','tables'), exist_ok=True)
res_df.to_csv(os.path.join('..','results','tables','05_feature_drift_results.csv'), index=False)
```





Notes and research-quality comments

- We add noise or scale features to emulate distributional shift: when model relies on feature statistics learned during training, changing those statistics can systematically bias predictions and reduce performance.
- KS statistic quantifies distributional shift between original and drifted feature; correlating KS with accuracy drop demonstrates why distributional drift detection is valuable.
- Random seeds (`RANDOM_STATE`) ensure reproducible experiments for paper-quality results.