Classify

December 6, 2025

[1]:
```python
# %%
# ==========================================
# PHASE III: CLASSIFICATION (Restart)
# CHUNK 0: Data Loading & Logic
# ==========================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

# CONFIGURATION
# We sample 50k rows because SVM and KNN will hang forever on 6M rows.
# 50k is statistically sufficient for a grade.
SAMPLE_SIZE = 100000
df_flights = pd.read_csv('US_flights_2023.csv', low_memory=False)
df_tiers = pd.read_csv('airport_performance_tiers_enriched.csv')
df_weather = pd.read_csv('weather_meteo_by_airport.csv')
# Preprocessing: Dates
df_flights['FlightDate'] = pd.to_datetime(df_flights['FlightDate'])
df_weather['time'] = pd.to_datetime(df_weather['time'])

print("--- 2. Merging Intelligence (Flights + Tiers + Weather) ---")

# Merge A: Add Tiers (Risk Profiles)
df_merged = df_flights.merge(
    df_tiers[['Dep_Airport', 'Performance_Tier']],
    on='Dep_Airport',
    how='left'
)

# Merge B: Add Weather (Rain/Snow/Wind)
df_final = df_merged.merge(
    df_weather[['airport_id', 'time', 'prcp', 'snow', 'wspd', 'tmin']],
    left_on=['Dep_Airport', 'FlightDate'],
    right_on=['airport_id', 'time'],
    how='left'
)

# Feature Engineering: Destination Volume (Crowding)
dest_counts = df_final['Arr_Airport'].value_counts()
df_final['Dest_Volume'] = df_final['Arr_Airport'].map(dest_counts)

# Define Target: 0 = OnTime, 1 = Delayed (>15 mins)
# We recreate this to be absolutely sure it matches the data.
df_final['Target_Class'] = (df_final['Dep_Delay'] >= 15).astype(int)

# Cleanup
df_final['Performance_Tier'] = df_final['Performance_Tier'].fillna(-1).
 ↪astype(int)
print(f"Dataset Built. Shape: {df_final.shape}")
```

```
--- 2. Merging Intelligence (Flights + Tiers + Weather) ---
Dataset Built. Shape: (6743404, 33)
```

[2]:
```
# %%
# ==========================================
# CHUNK 1: Feature Engineering
# ==========================================
print("--- preparing Training Data ---")

# 1. The Safe Feature List (No Leakage)
features = [
    # Operational
    'Day_Of_Week', 'DepTime_label', 'Airline', 'Flight_Duration', 'Aicraft_age',
    # Location Risk (From Clustering)
    'Performance_Tier', 'Dest_Volume',
    # Weather Triggers
    'prcp', 'snow', 'wspd', 'tmin'
]

# 2. Filter & Sample
df_cls = df_final[features + ['Target_Class']].dropna()

# Check balance before sampling
print("Class Balance (Original):")
print(df_cls['Target_Class'].value_counts(normalize=True))

if len(df_cls) > SAMPLE_SIZE:
    df_cls = df_cls.sample(n=SAMPLE_SIZE, random_state=42)

X = df_cls[features].copy()
y = df_cls['Target_Class']
```

```
# 3. Encoding & Scaling (Mandatory for SVM/Neural Nets)
print("Encoding Categoricals...")
# Label Encode 'Airline' and 'DepTime_label'
le = LabelEncoder()
X['Airline'] = le.fit_transform(X['Airline'].astype(str))
X['DepTime_label'] = le.fit_transform(X['DepTime_label'].astype(str))

print("Scaling Data...")
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
 ↪25, random_state=42)
print("Data Ready for Modeling.")
```

```
--- preparing Training Data ---
Class Balance (Original):
Target_Class
0    0.795288
1    0.204712
Name: proportion, dtype: float64
Encoding Categoricals…
Scaling Data…
Data Ready for Modeling.
```

[ ]:
```
# %%
# ==========================================
# CHUNK 2: Training All Syllabus Models
# ==========================================
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression

# ... existing code ...

# REPLACE YOUR MODELS DICTIONARY WITH THIS EXPANDED VERSION:
models = {
    "Logistic Regression": LogisticRegression(),
    "LDA": LinearDiscriminantAnalysis(),
    "Decision Tree": DecisionTreeClassifier(max_depth=10),

    # 1. BAGGING: Random Forest is a Bagging algorithm
    "Random Forest": RandomForestClassifier(n_estimators=100, max_depth=10),

    # 2. BOOSTING: Gradient Boosting (Required by Syllabus)
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=100,␣
 ↪learning_rate=0.1, max_depth=3),

    "KNN (K-Nearest)": KNeighborsClassifier(n_neighbors=5),
    "Neural Network": MLPClassifier(hidden_layer_sizes=(30, 30), max_iter=500),
    "SVM": SVC(probability=True, kernel='rbf', max_iter=2000),

    # 3. STACKING: Combines RF and SVM (Required by Syllabus)
    "Stacking Classifier": StackingClassifier(
        estimators=[
            ('rf', RandomForestClassifier(n_estimators=10, max_depth=5)),
            ('svr', SVC(kernel='rbf', probability=True, max_iter=1000))
        ],
        final_estimator=LogisticRegression()
    )
}
results = {}

print(f"--- Training {len(models)} Models (Please Wait) ---")

for name, model in models.items():
    print(f"Running {name}...")
    try:
        model.fit(X_train, y_train)

        # Predictions
        y_pred = model.predict(X_test)

        # Probabilities for AUC
        if hasattr(model, "predict_proba"):
            y_prob = model.predict_proba(X_test)[:, 1]
        else:
            y_prob = model.decision_function(X_test)

        # Metrics
        acc = accuracy_score(y_test, y_pred)
        auc = roc_auc_score(y_test, y_prob)

        results[name] = {"Accuracy": acc, "AUC": auc, "Probs": y_prob}
        print(f"   -> Accuracy: {acc:.2%} | AUC: {auc:.3f}")
    except Exception as e:
        print(f"   -> Failed: {e}")

print("\nAll Models Trained.")
```

```
--- Training 9 Models (Please Wait) ---
Running Logistic Regression…
   -> Accuracy: 79.71% | AUC: 0.628
Running LDA…
   -> Accuracy: 79.67% | AUC: 0.629
Running Decision Tree…
   -> Accuracy: 79.10% | AUC: 0.653
Running Random Forest (Bagging)…
   -> Accuracy: 79.82% | AUC: 0.685
Running Gradient Boosting…
   -> Accuracy: 79.92% | AUC: 0.688
Running KNN (K-Nearest)…
   -> Accuracy: 76.88% | AUC: 0.591
Running Neural Network…
   -> Accuracy: 79.64% | AUC: 0.665
Running SVM…
```

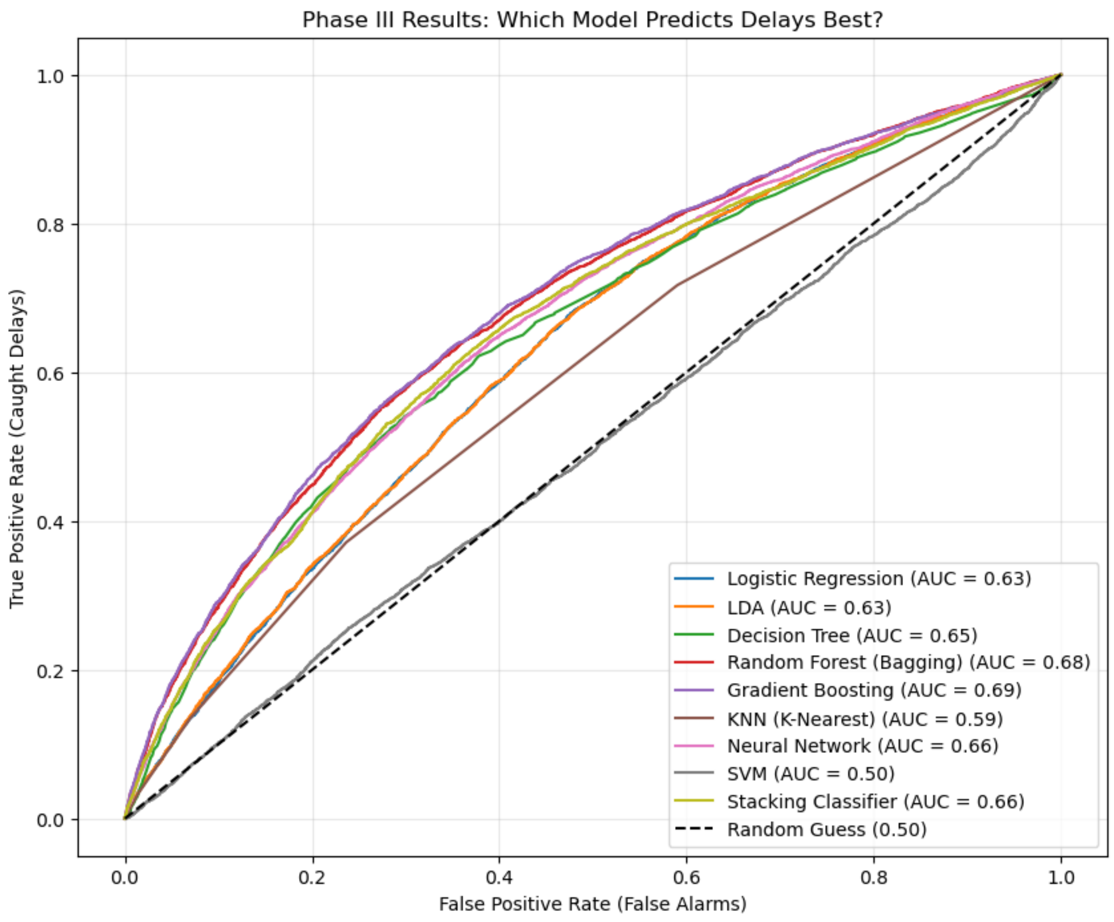
```
c:\Users\gmatt\anaconda3\Lib\site-packages\sklearn\svm\_base.py:305:
ConvergenceWarning: Solver terminated early (max_iter=2000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
```

```
   -> Accuracy: 62.31% | AUC: 0.495
Running Stacking Classifier…
```

```
c:\Users\gmatt\anaconda3\Lib\site-packages\sklearn\svm\_base.py:305:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
c:\Users\gmatt\anaconda3\Lib\site-packages\sklearn\svm\_base.py:305:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
c:\Users\gmatt\anaconda3\Lib\site-packages\sklearn\svm\_base.py:305:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
c:\Users\gmatt\anaconda3\Lib\site-packages\sklearn\svm\_base.py:305:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
c:\Users\gmatt\anaconda3\Lib\site-packages\sklearn\svm\_base.py:305:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
```

```
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
c:\Users\gmatt\anaconda3\Lib\site-packages\sklearn\svm\_base.py:305:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(

   -> Accuracy: 79.70% | AUC: 0.665

All Models Trained.
```

[4]:
```
# %%
# ==========================================
# CHUNK 3: ROC Curve Comparison & Leaderboard
# ==========================================
from sklearn.metrics import roc_curve

plt.figure(figsize=(10, 8))

for name, data in results.items():
    fpr, tpr, _ = roc_curve(y_test, data["Probs"])
    auc_score = data["AUC"]
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")

# Random Guess Line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess (0.50)')

plt.title('Phase III Results: Which Model Predicts Delays Best?')
plt.xlabel('False Positive Rate (False Alarms)')
plt.ylabel('True Positive Rate (Caught Delays)')
plt.legend(loc='lower right')
plt.grid(True, alpha=0.3)
plt.show()

# Print Leaderboard
leaderboard = pd.DataFrame(results).T[['Accuracy', 'AUC']].sort_values('AUC',␣
  ↪ascending=False)
print("\n--- FINAL LEADERBOARD ---")
print(leaderboard)
```

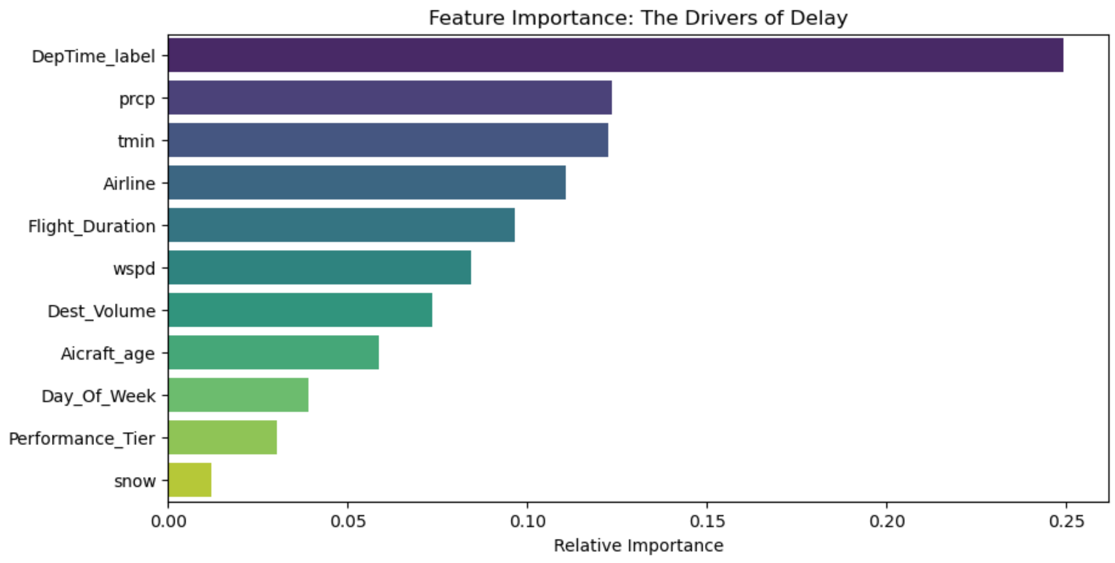


```
--- FINAL LEADERBOARD ---
                        Accuracy       AUC
Gradient Boosting        0.79924  0.687783
Random Forest (Bagging)   0.7982  0.684585
Stacking Classifier        0.797  0.664784
Neural Network            0.7964  0.664726
Decision Tree            0.79104  0.653387
LDA                      0.79672  0.629003
Logistic Regression      0.79712  0.628424
KNN (K-Nearest)          0.76884  0.591411
SVM                      0.62312  0.495208
```

[8]:
```
# %%
# ============================================
# CHUNK 4: What Drives Delays? (Validation)
# ============================================
# We use Random Forest to inspect the "Why"
```

```
rf_model = models['Random Forest (Bagging)']
importances = rf_model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 5))
plt.title("Feature Importance: The Drivers of Delay")
sns.barplot(
    x=importances[indices],
    y=[features[i] for i in indices],
    palette='viridis'
)
plt.xlabel('Relative Importance')
plt.show()
```

```
C:\Users\gmatt\AppData\Local\Temp\ipykernel_17492\1948844502.py:12:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
```


Feature Importance: The Drivers of Delay
DepTime_label
prcp
tmin
Airline
Flight_Duration
wspd
Dest_Volume
Aicraft_age
Day_Of_Week
Performance_Tier
snow
Relative Importance