AI In Healthcare, ML/DL Tutorial

Evan Jones, UT ID: ej8387

Assignment source at:

https://github.com/etjones/aihc_hw6_mldl_ej8387

Overview: Who will make it out of the ICU?

Every hospital admission carries some risk.

What factors, at admission, predict patient survival best?

Load Data From SQLite Database as Pandas Dataframes

We use a SQLite database containing Mimic-III data, built with the import.py script from the mimic-code repository.

```python
def _load_and_prepare_table(
    db: sqlite_utils.Database,
    table_name: str,
    date_cols: list[str] | None = None,
    string_cols: list[str] | None = None,
) -> pd.DataFrame:
    """Loads a table, converts specified columns, and returns a DataFrame."""
    try:
        table = db[table_name]
        # Convert table rows to DataFrame
        df = pd.DataFrame(table.rows)

        if df.empty:
            print(f"Warning: Table '{table_name}' is empty.")
            return df

        # Convert date columns
        if date_cols:
            for col in date_cols:
                if col in df.columns:
                    df[col] = pd.to_datetime(df[col], errors="coerce")

        # Convert string columns
        if string_cols:
            for col in string_cols:
                if col in df.columns:
                    df[col] = df[col].astype(str)

    except Exception as e:
        print(f"Error loading or preparing table {table_name}: {e}")
        return pd.DataFrame()  # Return empty DataFrame on error
    return df
```

Combine admissions, patients, and diagnoses tables

We're interested in data available immediately at intake: patient demographics and diagnoses.

We include human-readable SHORT_TITLE columns for diagnosis codes.

```python
def load_data(db_path: str) -> tuple[pd.DataFrame, dict]:
"""Loads data from the MIMIC-III database via SQLite, performs joins,
feature engineering, and returns the final DataFrame and ICD-9 code mapping."""
print(f"Loading data from {db_path}...")
db = sqlite_utils.Database(db_path)

# Load tables using the helper function
data = {}
data["admissions"] = _load_and_prepare_table(
    db, "admissions", date_cols=["ADMITTIME", "DISCHTIME", "DEATHTIME"]
)
data["patients"] = _load_and_prepare_table(db, "patients", date_cols=["DOB"])
data["diagnoses_icd"] = _load_and_prepare_table(
    db, "diagnoses_icd", string_cols=["ICD9_CODE"]
)

data["d_icd_diagnoses"] = _load_and_prepare_table(
    db, "d_icd_diagnoses", string_cols=["ICD9_CODE"]
)

# Check if essential tables are loaded
if data["admissions"].empty or data["patients"].empty:
    print(
        "Error: Essential tables (admissions, patients) could not be loaded. Exiting."
    )
    sys.exit(1)

# Create ICD-9 mapping; {icd_code: short_diagnosis_title}
icd9_map = {}
if not data["d_icd_diagnoses"].empty:
    icd9_map = pd.Series(
        data["d_icd_diagnoses"]["SHORT_TITLE"].values,
        index=data["d_icd_diagnoses"]["ICD9_CODE"],
    ).to_dict()
else:
    print(
        "Warning: d_icd_diagnoses table is empty. ICD-9 descriptions will be unavailable."
    )

# --- Joins and Feature Engineering --- #

# 1. Merge Admissions and Patients (Core join)
df = pd.merge(
    data["admissions"],
    data["patients"][["SUBJECT_ID", "GENDER", "DOB"]],
    on="SUBJECT_ID",
    how="left",
)

# Call helper functions for subsequent steps
df = _calculate_age(df)
df = _get_primary_diagnosis(df, data["diagnoses_icd"])
df = _calculate_survival_days(df)

print(f"Data loading and feature engineering complete. Final shape: {df.shape}")
# print("Final DataFrame columns:", df.columns.tolist())
# print(df.head())
# print(df.info())
# print(df.describe(include='all'))

return df, icd9_map
```

Calculate age, making up for Mimic-3 Obfuscation

To anonymize patients, patient DOBs and admission days are shifted by random amounts.
We generate a synthetic column, AGE from the obfuscated data.

```python
def _calculate_age(df: pd.DataFrame) -> pd.DataFrame:
"""Calculates age at admission robustly, handling implausible values and overflows."""
df["AGE"] = np.nan  # Initialize AGE column

valid_dates_mask = (
    df["ADMITTIME"].notna()
    & df["DOB"].notna()
    & (df["DOB"].dt.year >= 1900)  # Exclude likely placeholder DOBs
)

if valid_dates_mask.any():
    # Select the relevant rows
    admittime = df.loc[valid_dates_mask, "ADMITTIME"]
    dob = df.loc[valid_dates_mask, "DOB"]

    # Calculate difference in years
    years_diff = admittime.dt.year - dob.dt.year

    # Adjust age if birthday hasn't occurred yet in the admission year
    # Correction = 1 if admit_day_of_year < dob_day_of_year, else 0
    correction = (admittime.dt.dayofyear < dob.dt.dayofyear).astype(int)

    # Calculate final age in years
    calculated_age = years_diff - correction

    # Apply the calculated age to the DataFrame subset
    df.loc[valid_dates_mask, "AGE"] = calculated_age

    # Cap age at 90 (MIMIC obfuscates ages > 89)
    # Apply clipping to the entire column after calculations
    df["AGE"] = df["AGE"].clip(upper=90)

print(f"Calculated AGE. Missing values: {df['AGE'].isna().sum()}. Capped at 90.")
return df
```

Define `SurvivalPredictor` model

We use a simple two-layer Linear/ReLU PyTorch model to train on our dataset.

More complex models like convolutional neural networks or transformers wouldn't be very helpful with the slim set of data available at intake time.

```python
class SurvivalPredictor(nn.Module):
"""Simple MLP for survival time prediction."""

def __init__(self, input_dim: int):
    super().__init__()
    self.layer_1 = nn.Linear(input_dim, 64)
    self.relu = nn.ReLU()
    self.layer_2 = nn.Linear(64, 32)
    self.output_layer = nn.Linear(
        32, 1
    )  # Predicting a single value (survival days)

def forward(self, x: torch.Tensor) -> torch.Tensor:
    # TODO: Implement forward pass
    x = self.relu(self.layer_1(x))
    x = self.relu(self.layer_2(x))
    x = self.output_layer(x)
    return x
```

Train model

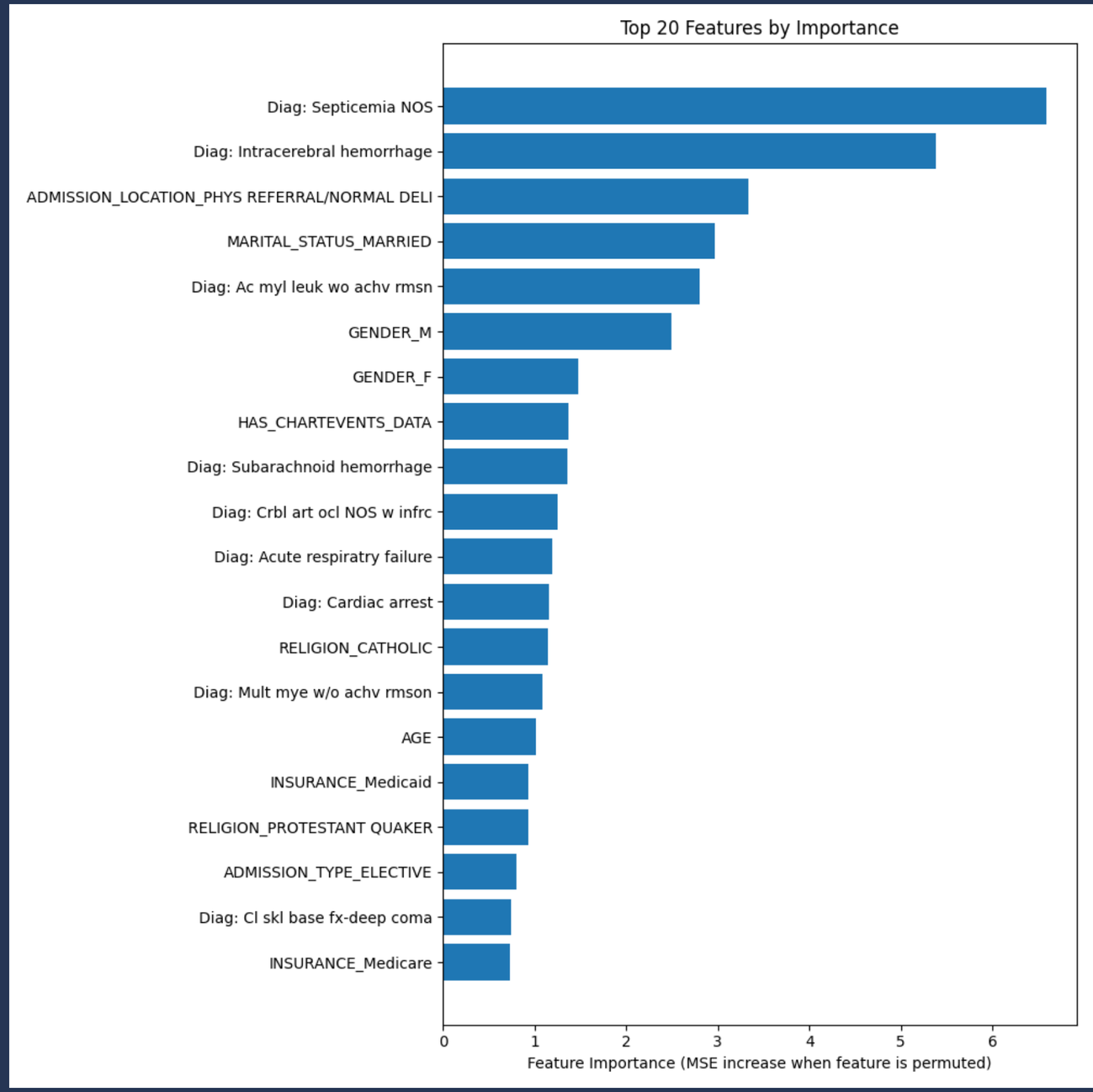
Even a simple Linear/ReLU deep learning model learns the important features.

One benefit of this approach is that we don't have to know which features are important; gradient descent identifies significant features on its own.

Identify influential features with Permutation Importance

For each feature, randomly change values several times, and record which changes alter the loss (mean-squared error) most.

This is a sizeable function. See `patient_survival.calulate_feature_importance()` for details.



Permutation Importance weaknesses

- Permutation Importance analysis **doesn't** tell us which direction a change in a given feature changes the predictive power of the model. We see both `GENDER_M` and `GENDER_F` in the chart of feature importance. What does this mean?
- Likewise, does `RELIGION_PROTESTANT_QUAKER` make a patient more or less likely to survive? I suspect the answer is that Quakerism is a proxy for high socioeconomic status, so quakers tend to have better health outcomes, but I haven't identified this conclusively.
- `RELIGION_CATHOLIC` is also an influential feature, but I suspect in the opposite direction, correlated with under-resourced Latin American immigrants.
- This phenomenon calls for some more easily interpretable analysis.

Calculate importance with SHAP values

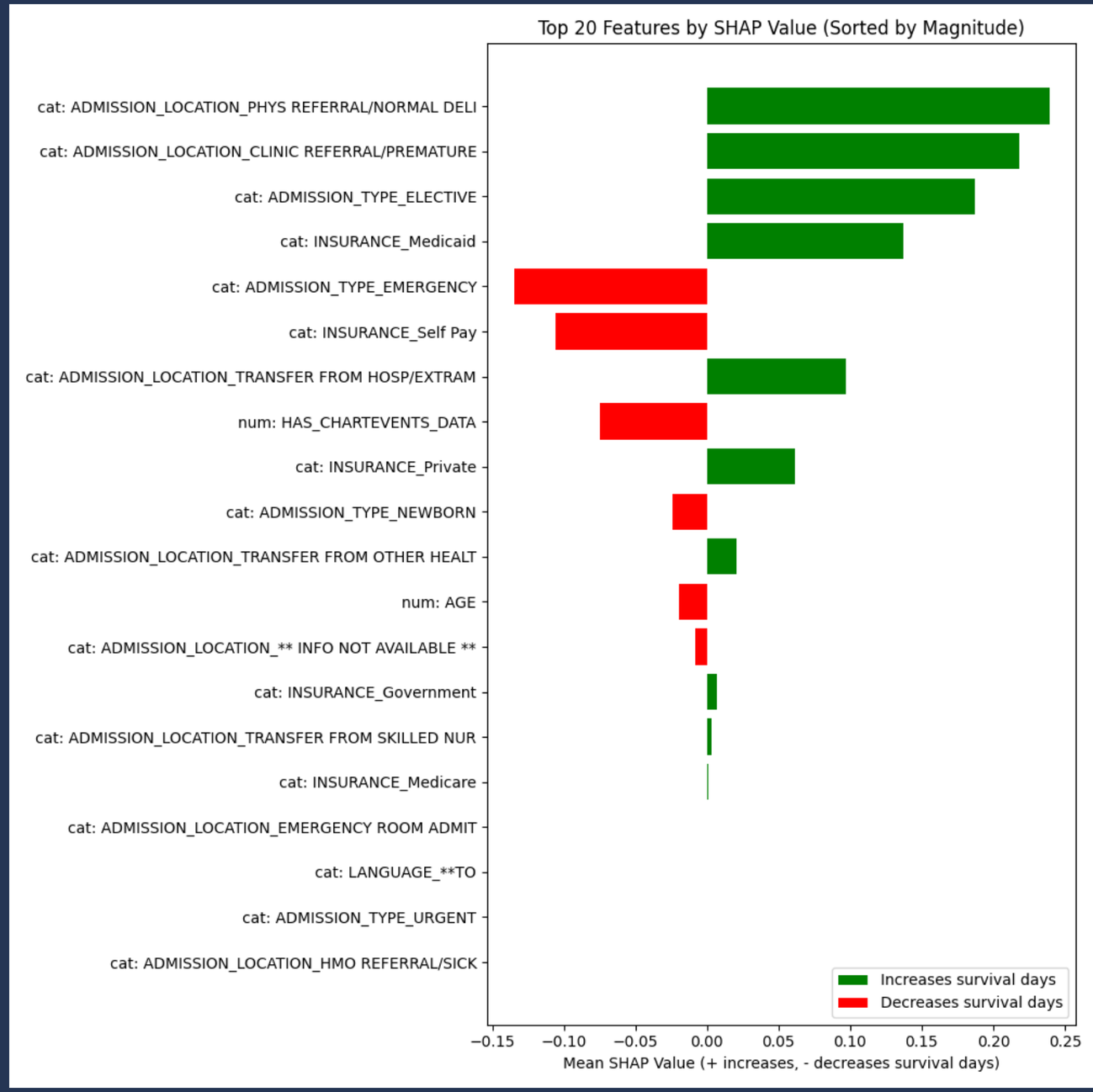
SHAP values tell us about magnitude of importance, but also *direction*.

Some features predict longer survival. Others predict shorter.

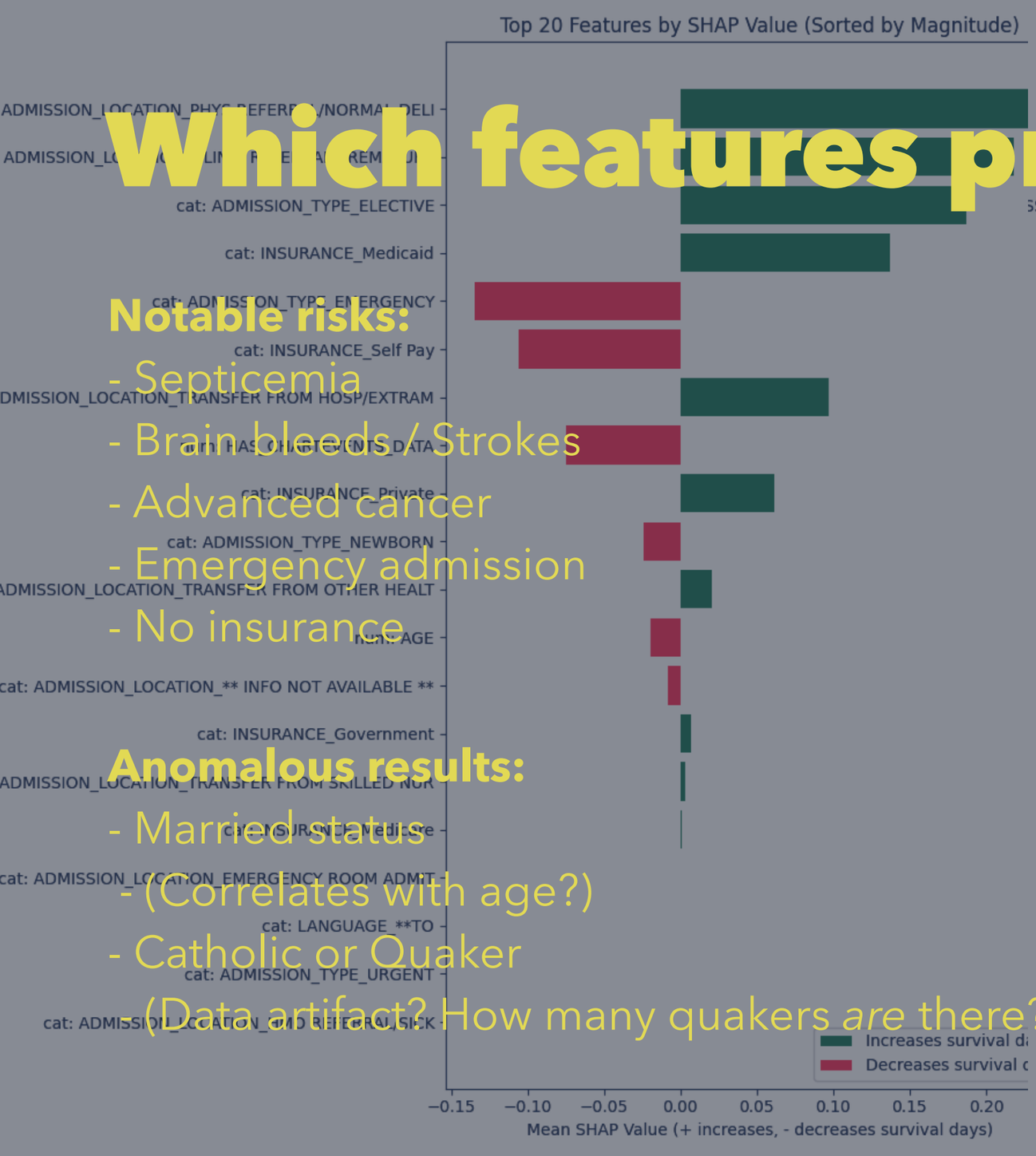
It's important to know which is which!

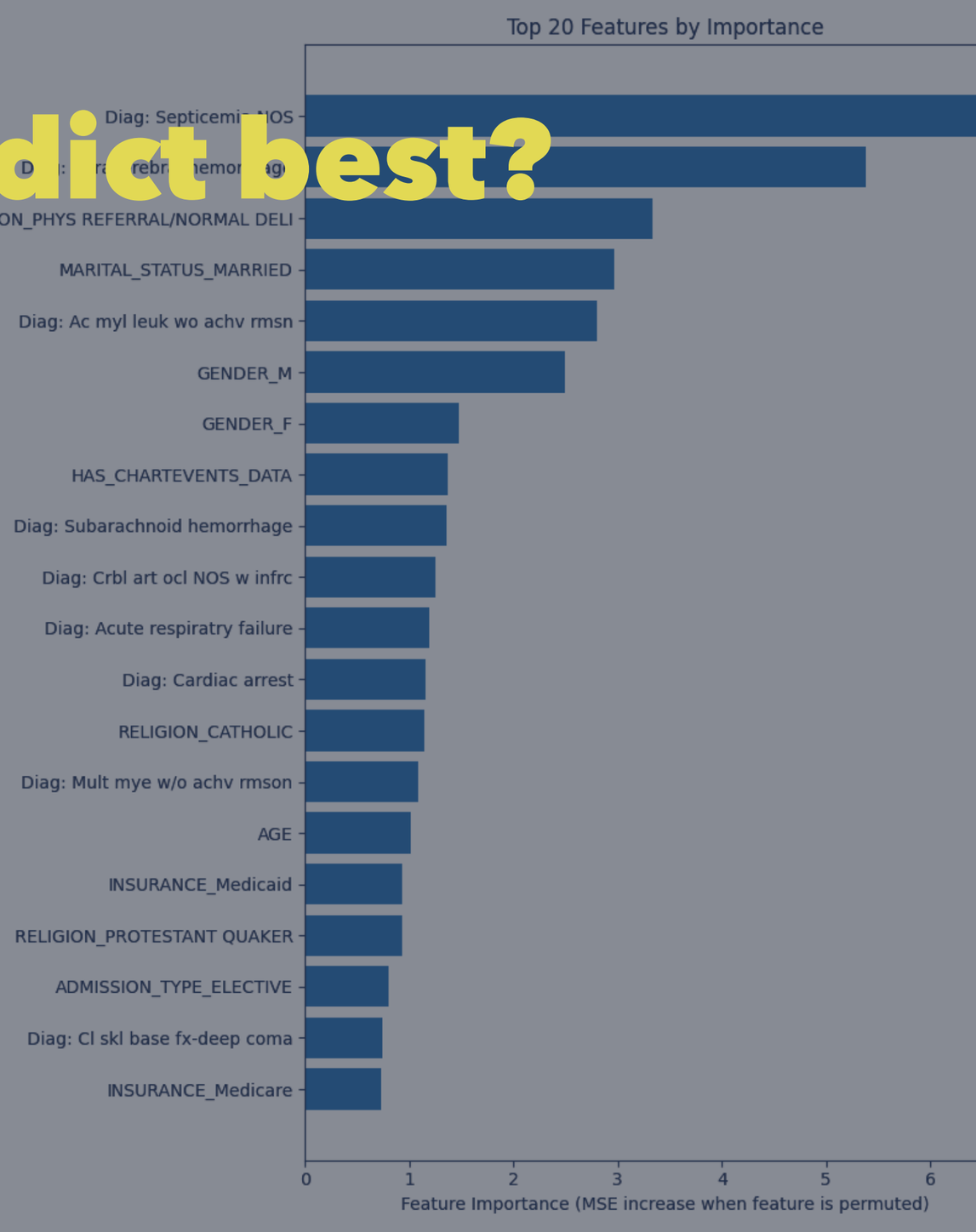
Anomolous behavior

- No diagnosis codes appear as significant SHAP features. I haven't isolated why they appear in the Permutation Importance analysis but not in the SHAP values.



Which features predict best?





Notable risks:

- Septicemia
- Brain bleeds / Strokes
- Advanced cancer
- Emergency admission
- No insurance

Anomalous results:

- Married status
  - (Correlates with age?)
- Catholic or Quaker
  - (Data artifact? How many quakers *are* there?)

Conclusion

- I trained a Linear/ReLU deep network on patient data available immediately at intake, predicting length of survival for patients who died at the hospital.
- I predict length of survival relatively accurately.
- I used two methods, Permutation Importance and SHAP Values, to identify important predictive features.
- I was unable to resolve the different results of my importance analyses.