NFL Play Caller

Machine learning model to predict run vs pass plays in NFL games based on pre-snap situational features. Built using logistic regression on 2021-2023 play-by-play data.

Project Structure

```
nflplaycaller/
├── src/nfl_run_pass/          # Core ML library
│   ├── __init__.py
│   ├── config.py              # Settings, paths, hyperparameters
│   ├── data_loading.py        # Load and filter play-by-play data
│   ├── preprocessing.py       # Train/test split, scaling
│   ├── features.py            # Feature engineering
│   ├── models.py              # Model training (logistic regression)
│   ├── evaluation.py          # Metrics and evaluation
│   ├── pipeline.py            # End-to-end training pipeline
│   └── tuning.py              # Hyperparameter tuning
│
├── api/                       # FastAPI backend
│   └── main.py                # REST API for predictions
│
├── streamlit/                 # Frontend applications
│   ├── st_app.py             # Main predictor UI
│   ├── steamlit_btm.py       # "Beat the Model" game
│   └── streamlit_app.py      # Alternative UI version
│
├── models/                    # Saved artifacts
│   ├── log_reg_model.pkl     # Trained logistic regression model
│   ├── scaler.pkl            # Fitted StandardScaler
│   └── feature_cols.json     # Feature column names (ordered)
│
├── data/                      # Data files (gitignored)
│   └── raw/
│       └── pbp_2021_2023.csv  # NFL play-by-play dataset
│
├── tests/                     # Unit and integration tests (TO BE CREATED)
│   ├── test_data_loading.py
│   ├── test_features.py
│   ├── test_models.py
│   ├── test_pipeline.py
│   └── test_api.py
│
├── run_pipeline.py            # Script to train model
├── sample_plays.csv           # Sample data for demos
├── requirements.txt           # Python dependencies
├── pyproject.toml             # Project metadata (uv/pip)
└── README.md                  # This file
```

Module Overview

Core Library (`src/nfl_run_pass/`)

The library follows a modular pipeline design:

1. **`config.py`** - Central configuration
   - Data paths and filtering rules
   - Feature definitions
   - Model hyperparameters
   - Train/test split settings
2. **`data_loading.py`** - Data ingestion
   - `load_raw_data()` - Read CSV
   - `filter_season()` - Filter to 2023 season
   - `filter_run_pass_plays()` - Keep only run/pass plays
   - `add_is_pass_target()` - Create binary target (0=run, 1=pass)
3. **`features.py`** - Feature engineering
   - `add_engineered_features()` - Create pre-snap features:
     - Field position (red zone, goal-to-go, yardline)
     - Down/distance buckets (short/medium/long)
     - Formation (shotgun, no huddle)
     - Score state (trailing, tied, leading)
     - Time context (quarter, late half)
   - `build_feature_matrix()` - Construct (X, y) for modeling
4. **`preprocessing.py`** - Data preparation
   - `handle_missing_values()` - Imputation
   - `split_train_test()` - Stratified train/test split
   - `scale_features()` - StandardScaler normalization
5. **`models.py`** - Model training
   - `create_base_log_reg_model()` - Logistic regression setup
   - `train_log_reg_model()` - Fit with optional GridSearchCV
6. **`evaluation.py`** - Model assessment
   - Accuracy, precision, recall, F1, ROC-AUC
   - Confusion matrix
   - Train vs test metrics
7. **`pipeline.py`** - Orchestration
   - `run_training_pipeline()` - End-to-end workflow:
     1. Load data
     2. Engineer features

        3. Split and scale
        4. Train model
        5. Evaluate
        6. Save artifacts
    - `save_artifacts()` / `load_artifacts()` - Persistence

8. **`tuning.py`** - Hyperparameter optimization

    - `run_log_reg_tuning()` - GridSearchCV for C and class_weight
    - `compare_models()` - Baseline vs tuned comparison

API (`api/`)

FastAPI REST endpoint for real-time predictions:

- **POST `/predict`** - Takes game situation JSON, returns prediction + probabilities
- Loads saved model/scaler on startup
- CORS-enabled for frontend integration

Frontend (`streamlit/`)

Three Streamlit applications:

1. **`st_app.py`** - Main interactive predictor with visual field
2. **`steamlit_btm.py`** - "Beat the Model" game with real plays + GIFs
3. **`streamlit_app.py`** - Alternative layout

Setup Instructions

1. Install Dependencies

Using pip:

```
pip install -r requirements.txt
```

Using uv (recommended):

```
uv pip install -e .
```

2. Prepare Data

Place your play-by-play CSV at:

```
data/raw/pbp_2021_2023.csv
```

The data should contain nflfastR-style columns including:

- `season`, `week`, `game_id`, `play_id`
- `play_type` (run/pass/etc)
- `down`, `ydstogo`, `yardline_100`
- `shotgun`, `no_huddle`, `qtr`
- `posteam_score`, `defteam_score`, `score_differential`
- `half_seconds_remaining`, `game_seconds_remaining`
- `posteam`, `home_team`, `away_team`

3. Train the Model

```
python run_pipeline.py
```

This will:

- Load and filter 2023 season data
- Engineer pre-snap features
- Split train/test (80/20, stratified)
- Scale features with StandardScaler
- Train logistic regression with GridSearchCV
- Save model artifacts to `models/`

4. Run Applications

Streamlit UI:

```
streamlit run streamlit/st_app.py
```

FastAPI Backend:

```
cd api
uvicorn main:app --reload
```

Model Details

Features (18 total)

Base numeric (4):

- `down` (1-4)
- `ydstogo` (yards to first down)
- `yardline_100` (distance to opponent end zone)
- `game_seconds_remaining` (time left in half)

Engineered binary (14):

- Field position: `is_red_zone`, `is_goal_to_go`
- Distance buckets: `short_ydstogo` (≤3), `medium_ydstogo` (4-7), `long_ydstogo` (≥8)
- Formation: `shotgun`, `no_huddle`
- Score state: `is_trailing`, `is_tied`, `is_leading`, `score_differential`
- Time: `is_fourth_qtr`, `late_half` (<2 min)
- Home/away: `is_home_offense`

Model Architecture

- **Algorithm:** Logistic Regression
- **Preprocessing:** StandardScaler on all features
- **Hyperparameter Tuning:** GridSearchCV
  - `C`: [0.01, 0.1, 1.0, 10.0]
  - `class_weight`: [None, "balanced"]
  - Scoring: F1
  - CV folds: 3

Performance

Add your metrics here after training

```
Train accuracy: X.XX%
Test accuracy:  X.XX%
Test F1:        X.XX
Test ROC-AUC:   X.XX
```

Testing (TODO - Priority #1)

We need to create comprehensive tests in `tests/`:

Unit Tests Needed

1. **`test_data_loading.py`**
   - Test CSV loading with valid/invalid paths
   - Test season filtering
   - Test run/pass filtering
   - Test target creation
2. **`test_features.py`**
   - Test each engineered feature calculation
   - Test missing value handling
   - Test feature matrix construction
   - Test feature alignment with saved feature_cols.json
3. **`test_preprocessing.py`**
   - Test train/test split stratification

   - Test scaling (mean=0, std=1)
   - Test missing value imputation

4. `test_models.py`

   - Test model creation
   - Test training with/without GridSearchCV
   - Test prediction shape and probabilities

5. `test_pipeline.py`

   - Test end-to-end pipeline
   - Test artifact saving/loading
   - Test reproducibility with random_state

6. `test_evaluation.py`

   - Test metric calculations
   - Test edge cases (all one class, etc)

7. `test_api.py`

   - Test /predict endpoint
   - Test input validation
   - Test feature reconstruction matches training

Integration Tests

- Full pipeline on `sample_plays.csv`
- API + frontend integration
- Model versioning and reproducibility

Testing Framework

Use pytest:

```
pip install pytest pytest-cov
pytest tests/ -v
pytest tests/ --cov=src/nfl_run_pass --cov-report=html
```

Configuration

All settings are in `src/nfl_run_pass/config.py`:

```python
from nfl_run_pass.config import CONFIG

# Access configuration
CONFIG.data.season           # 2023
CONFIG.train_test.test_size  # 0.2
CONFIG.log_reg.param_grid    # GridSearchCV params
```

To modify:

```
CONFIG.data.season = 2022
CONFIG.train_test.test_size = 0.25
```

Development Workflow

For New Features

1. Update `config.py` with any new settings
2. Modify relevant module (e.g., `features.py`)
3. Write tests for new functionality
4. Run pipeline to retrain model
5. Update API/frontend if needed

For Bug Fixes

1. Write a failing test that reproduces the bug
2. Fix the bug
3. Verify test passes
4. Check integration tests still pass

Before Committing

```
# Run all tests
pytest tests/ -v

# Check code formatting
black src/ tests/
isort src/ tests/

# Run type checking (optional)
mypy src/
```

API Usage

Request Format

```
{
  "down": 3,
  "ydstogo": 7,
  "yardline_100": 35,
  "offense_score": 14,
  "defense_score": 17,
  "qtr": 4,
  "seconds_remaining_half": 180,
  "shotgun": true,
  "no_huddle": false,
  "is_home_offense": true
}
```

Response Format

```
{
  "prediction": "PASS",
  "prob_pass": 0.73,
  "prob_run": 0.27
}
```

Example with curl

```
curl -X POST "http://localhost:8000/predict" \
  -H "Content-Type: application/json" \
  -d '{
    "down": 3,
    "ydstogo": 7,
    "yardline_100": 35,
    "offense_score": 14,
    "defense_score": 17,
    "qtr": 4,
    "seconds_remaining_half": 180,
    "shotgun": true,
    "no_huddle": false,
    "is_home_offense": true
  }'
```

Contributing

Getting Started

1. Clone the repository
2. Create a virtual environment: `python -m venv venv`
3. Install dependencies: `pip install -r requirements.txt`
4. Install package in editable mode: `pip install -e .`
5. Run tests: `pytest tests/`

Branching Strategy

- `main` - Production-ready code

- `develop` - Integration branch
- `feature/feature-name` - New features
- `fix/bug-description` - Bug fixes
- `test/test-description` - Test additions

Pull Request Process

1. Create feature branch from `develop`
2. Write code + tests
3. Ensure all tests pass
4. Update documentation if needed
5. Submit PR to `develop`
6. Get code review from at least one teammate
7. Merge after approval

Team Collaboration

Code Review Checklist

- [ ] Tests written for new functionality
- [ ] All tests pass
- [ ] Code follows project style (see existing modules)
- [ ] Documentation updated (docstrings, README)
- [ ] No sensitive data or credentials in code
- [ ] Configuration changes documented
- [ ] Breaking changes clearly communicated

Communication

- Use GitHub Issues for bugs and feature requests
- Use GitHub Projects for sprint planning
- Document major decisions in `/docs/decisions/`
- Update this README as project evolves

Resources

- **nflfastR data guide:** https://www.nflfastr.com/articles/beginners_guide.html
- **scikit-learn docs:** https://scikit-learn.org/stable/
- **FastAPI tutorial:** https://fastapi.tiangolo.com/tutorial/
- **Streamlit docs:** https://docs.streamlit.io/

License

[Add your license here]

Authors

[Add team member names and roles]

Questions? Open an issue or reach out to the team on [your communication channel].