ML with graphs ideas

Open Data Events v4.0.0.pdf
PDF

Go through the description of this dataset and give me some project ideas that can be implemented using ML with Graphs , Don't give the code just give the ideas and workflow

Thanks for sharing the dataset spec. To help me tailor project ideas using machine learning and graphs, could you let me know:

1. Are you interested in a specific type of graph structure? (e.g., player-passing networks, event transition graphs, possession graphs, etc.)
2. Do you have a preferred type of ML task? (e.g., prediction, clustering, anomaly detection, recommendation)
3. Is your focus more on team-level analysis, player-level analysis, or match-level dynamics?
4. Are there any constraints or objectives you want to prioritize? (e.g., interpretability, real-time performance, visualization)

This will help me give you more targeted and practical project workflows.

1) Open to ideas
2) No
3) All
4) no

Thanks! I'll explore a range of project ideas using machine learning with graph-based approaches, all grounded in the structure of the StatsBomb event data. I'll include ideas at the player, team, and match levels across different ML tasks (prediction, clustering, anomaly detection, etc.).

I'll get back to you with a list of concrete project concepts and workflow outlines shortly.

Graph-based Machine Learning Projects with StatsBomb Data

StatsBomb's open event data (v4.0.0) contains detailed logs of soccer actions – passes, shots, dribbles, tackles, carries, pressures, etc. – each with rich spatiotemporal context . Graph-based models can leverage this by treating players or events as nodes and interactions (passes, spatial proximity, sequences) as edges. Below are several project ideas at **player**, **team**, and **match** levels. Each idea outlines how to build the graph from StatsBomb events, the ML task, possible models, evaluation metrics, and extensions.

Player-Level Projects

1. Player Role Classification via Passing Networks

- **Graph Construction:** Build a *passing network* for each team or match. Nodes represent players (identified by StatsBomb IDs), and a directed edge from player A to B exists if A passes to B. Edge weights can be the total count or success rate of passes. Node features can include a player's average pitch location, playing position, or event counts. (This follows common practice: players as nodes, passes as edges .)
- **ML Problem: Node classification** – predict each player's position/role (e.g. defender, midfielder, forward) or playing style category (playmaker, stopper, etc.) based on the network.
- **Model:** A **Graph Neural Network (GNN)** such as a Graph Convolutional Network (GCN) or Graph Attention Network (GAT) that takes the team's pass graph as input. Node embeddings from the GNN can be fed to a softmax classifier for roles.
- **Metrics:** Accuracy or F1-score on labeled roles; confusion matrix. If positions are hierarchical (e.g. left/right mid), use hierarchical or multi-label metrics.
- **Extensions:** Incorporate other events (e.g. dribbles or carries) as additional edges or node attributes. Use **dynamic graphs** by aggregating passes over time or multiple matches, to capture evolving roles. One could also use **graph clustering** on the learned embeddings to discover new role categories (e.g. "metronomic midfielders" vs "box-to-box midfielders").

2. Player Similarity Clustering

- **Graph Construction:** Aggregate each player's interactions over a season or set of matches. Form a *player–player graph* across the league: nodes are players, and an undirected edge connects two players on the same team if they pass to each other, with weight = number of passes between them. (Or build a bipartite player–team graph.) Node features can include total passes, shot counts, etc.
- **ML Problem: Clustering** – group players into similar "styles" or performance tiers (e.g. prolific passers vs dribblers).
- **Model:** Learn node embeddings using a GNN (e.g. GraphSAGE, Graph Autoencoder) or graph embedding methods (Node2Vec), then apply k-means or spectral clustering on the embeddings. Alternatively, use **community detection** (e.g. Louvain) directly on the pass graph.
- **Metrics:** Silhouette score or Davies–Bouldin index for cluster quality; inspect cluster "profiles" (e.g. average stats per cluster). Evaluate interpretability by how well clusters align with known categories (positions, tiers).
- **Extensions:** Add multi-relational edges (passes vs dribbles vs carries) or temporal context (e.g. build separate graphs for different match phases). One could also perform **co-clustering** of players and actions by building a player–action bipartite graph.

3. Player Performance Regression (Value/Impact Prediction)

- **Graph Construction:** Represent each on-ball action (pass/dribble/tackle) as a graph of players on the field. For example, at the moment of a pass, create a graph of all 22 players as nodes (plus possibly a "ball" node). Edges can connect players on the same team (complete subgraph) and link to the ball node, similar to prior work . Node features include the player's position coordinates and motion; edge features include spatial distances or pass vectors.
- **ML Problem: Regression** – predict a continuous measure of player value or impact. For instance, predict the change in team's *Expected Threat (xT)* or *goal probability* resulting from an action, or predict a player's overall contribution rating.

- **Model:** A GNN (e.g. a Graph Transformer or GAT) over the constructed graph. This is similar to the "GoalNet" system, which uses a GNN over each event's player graph to predict expected threat . The model outputs a numeric score per node or per action.
- **Metrics:** Mean Squared Error (MSE) or R^2 on the target value (e.g. xT change), and rank correlation between predicted and true impact. Could also measure error per player or position.
- **Extensions:** Incorporate **temporal context** by constructing a sequence of graphs (one per action) and using a Temporal GNN (TGN) to capture action sequences. Additional node features (e.g. fatigue, recent touches) or edge features (e.g. event type encoding) can improve accuracy. One could also adapt this into a **classification** task by binning value (e.g. "high-impact" vs "low-impact" actions).

4. Anomalous Player Behavior Detection

- **Graph Construction:** Build a player–event bipartite graph or a player network as above over one match or season. Nodes are players, edges encode interactions (passes, challenges, pressures). Optionally, attach event nodes (shots, dribbles) to the players who performed them.
- **ML Problem: Anomaly detection** – identify players whose behavior is unusually different from their peers. For example, detect a player whose passing pattern or involvement differs significantly from past games (could indicate injury, fatigue, or malicious match-fixing).
- **Model:** An unsupervised graph-based anomaly detector. For example, train a **Graph Autoencoder (GAE)** or **Graph GAN** on typical player graphs; players whose reconstructed embeddings or edges have high error are anomalies. Alternatively, compute node-level statistics (degree, centrality) and flag outliers.
- **Metrics:** Precision/recall if labeled anomalies are available (rare); otherwise, use domain-knowledge checks (e.g. does the flagged player correspond to known outliers). Use the reconstruction error distribution as an "anomaly score".
- **Extensions:** Incorporate temporal analysis: compare a player's current graph neighborhood to an average over their historical networks. Combine with video or physiological data if available for more context.

Team-Level Projects

5. Team Style Classification

- **Graph Construction:** For each team in a match, build its passing network as before (nodes = players, edges = passes). Optionally combine both teams into one graph by labeling nodes with team affiliation. Include spatial info by positioning nodes at average pitch location.
- **ML Problem: Graph classification** – classify the *type of play* or tactical style of the team. For example, label matches or teams as "possession-oriented" vs "counterattacking", or classify the match outcome from the network.
- **Model:** A GNN (e.g. GraphSAGE or GAT) that takes the entire team network (or both teams' networks) as input and outputs a label. For match outcome, use both teams' graphs (as in Lee *et al.*) .
- **Metrics:** Accuracy/F1 of classification (e.g. ability to predict win/draw/loss or style category). For style, human expert evaluation of the clusters or labels can be used.
- **Extensions:** Use **dynamic graph modeling**: treat the passing network as evolving over the match (sliding windows). Incorporate other events (tackles, pressures) as additional edges to capture defense/offense mix. Include edge attributes such as pass success or distance. One could also apply **graph pooling** to summarize each team's graph into a vector for downstream classification.

6. Team Clustering by Passing Patterns

- **Graph Construction:** Create a graph-of-teams: each node is a team (or a team in a particular match), with edges encoding similarity of their passing networks. For example, compute a feature vector per team (e.g. centrality measures, clustering coefficient) and connect teams whose feature similarity is high. Alternatively, build a multipartite graph linking teams to players or events.
- **ML Problem: Clustering** – group teams into archetypes (e.g. "high-possession teams" vs "long-ball teams").
- **Model:** Compute graph embeddings for the team nodes (using GNNs or spectral embedding) and apply clustering (k-means, hierarchical). Or use community detection on the similarity graph.
- **Metrics:** Evaluate cluster coherence (silhouette score) and alignment with known categorizations (e.g. comparing clusters to leagues or coach styles).

- **Extensions:** Let teams have multiple graphs (e.g. one per match); use a multilayer graph where each layer is a match. Explore clustering on the bipartite player–team graph (teams clustered by having similar key players).

7. Pass Link Prediction in Team Networks

- **Graph Construction:** For a given team's passing network (for a match or over several matches), represent it as a graph as before. Use directed edges (A→B if A passed to B). Optionally include negative edges or weights for pass frequency.
- **ML Problem: Link prediction** – predict potential passes that did *not* occur in the data. For example, given partial pass data from the first half, predict which passes will appear in the second half (next-pass prediction).
- **Model:** A graph autoencoder or GNN configured for link prediction. For instance, train a GraphSAGE or VGAE to reconstruct adjacency: node embeddings are learned so that the inner product of embeddings predicts edge existence. For dynamic link prediction, use a Temporal GNN (TGN).
- **Metrics:** AUC-ROC or Average Precision for predicting held-out edges; precision@k for top-k predicted links. Can also report accuracy of next-pass receiver prediction.
- **Extensions:** Incorporate player attributes or context (e.g. current score, time) into edge features. Use spatial proximity: edges more likely between nearby players. Combine with imitation learning: simulate passes and compare predicted vs actual flow of play.

8. Team Anomaly Detection

- **Graph Construction:** Aggregate a team's play over a match or season into a pass-and-events network (players as nodes, passes/dribbles/pressures as edges). Compute graph statistics (degree, motif counts) for each match.
- **ML Problem: Anomaly detection** – identify matches or teams whose network deviates from the norm. For example, detect a match where a normally possession-heavy team suddenly plays very direct.
- **Model:** Fit a GNN-based autoencoder on many team graphs; flag those with high reconstruction error. Or use community-detection on a similarity graph of matches to find outliers. Could also apply an LSTM/GNN hybrid on graph metrics over time.
- **Metrics:** If anomalies are known (e.g. upsets), use precision/recall. Otherwise, inspect unsupervised anomaly scores (e.g. z-scores of network features).

- **Extensions:** Combine with game context (e.g. unusual weather, injuries). Use subgraph analysis to pinpoint which players or interactions cause the anomaly (subgraph outlier detection).

Match-Level Projects

9. Match Outcome Prediction via Dynamic Passing Graphs

- **Graph Construction:** Construct a *dynamic graph* over the course of the match. At each timestamp or action index, form a graph of all players (nodes) with edges representing passes that have occurred so far (or recent actions). Alternatively, use a *snapshot* at time *T*: nodes are players, edges are all passes up to time *T* (weighted by count). Include edges across teams only through the ball or for possession switches. Node features may include cumulative stats so far; edge features include total passes or last-pass timestamp.
- **ML Problem: Graph (sequence) classification** – predict the final result (home win/draw/away win) or in-game probabilities given the evolving graph. Similar to real-time win-probability models.
- **Model:** A **Temporal GNN** or sequence-of-graphs model. For example, use a GraphSAGE per snapshot and feed its embedding into an RNN; or use a Temporal Graph Network (TGN) directly on the event stream. Lee *et al.* demonstrate that a GAT-based model on passing graphs can accurately predict match outcome .
- **Metrics:** Classification accuracy or cross-entropy at various match times (e.g. at 45′, 60′). Measure calibration of win-probabilities (Brier score). Compare to baselines (pre-match odds or traditional statistical models).
- **Extensions:** Fuse non-pass events: integrate shots, tackles, and pressures as additional edges or node signals to improve prediction (the cited study reported 5–20% improvement by adding non-pass information). Use attention to focus on critical events. Also try regressing final goal difference or future goal probability.

10. Phase-of-Play Identification (e.g. Counterattacks vs Build-up)

- **Graph Construction:** Consider each sequence of actions (e.g. possession chain) as a graph. For example, build a graph whose nodes are the players involved in a possession and whose edges represent passes, dribbles, or carries in that possession (possibly temporal edges connecting events). Node features include positions and distances moved; edge features include event type and length. Alternatively, create a graph embedding by pooling node features for the possession.
- **ML Problem: Clustering/Classification** – cluster or classify possessions into phases (e.g. “build-up”, “fast break”, “set play”). This is like determining phases of play.
- **Model:** First, learn vector **embeddings** of each possession graph using a GNN (e.g. GCN or Graph Transformer) . Then apply k-means or another classifier on the embeddings. Juan Camilo Campos’ work shows that GCN-based pass embeddings capture semantic context and can be labeled for phase identification .
- **Metrics:** If ground-truth labels for phases exist, use clustering metrics (ARI, NMI) or classification accuracy. Otherwise, qualitatively assess if clusters correspond to known phases. For a continuous embedding space, measure how well simple labels (counterattack vs non-counter) separate.
- **Extensions:** Integrate pitch control or opponent formation information (e.g. distance to nearest defenders). Use a **sequence-to-sequence** GNN to capture how phases transition over time. Apply **graph motif analysis**: e.g. identify common subgraph patterns unique to counterattacks.

11. Anomalous Match/Event Detection

- **Graph Construction:** Form a **multigraph** of the entire match. Nodes are players, edges include all on-ball interactions (passes, tackles, fouls) labeled by type. Add a temporal dimension by ordering edges. Optionally include a “possession” node linking all players who touch the same possession.
- **ML Problem: Anomaly detection** – flag unusual match situations, such as a sudden strategic shift or an outlier sequence of events. For example, find a possession with an extremely unusual passing loop.

- **Model:** A graph-based anomaly detector. One approach is to slide a window over the match (creating many small graphs) and train a GNN autoencoder; windows with high reconstruction loss are anomalous. Alternatively, use **Graph Neural Markov Models** to detect unlikely transitions.
- **Metrics:** Since labeled anomalies are rare, one can validate by injected synthetic anomalies (e.g. randomize one player's actions) or by known events (e.g. VAR stoppages, weather delays). Compute anomaly scores over events and compare to baseline (random graphs).
- **Extensions:** Combine with unstructured data (commentary) to label anomalies. Use **reinforcement learning** to simulate "normal" vs "abnormal" strategies and see which current patterns deviate.

12. Reinforcement Learning for Strategy Optimization

- **Graph Construction:** Represent each game state as a graph: nodes are players (plus perhaps a "ball" node), edges encode team membership and recent passes (as in). Node features include positions, speeds, and recent event involvement; edge features include relative distances or pass frequencies. The **state** can be the current possession subgraph.
- **ML Problem: Reinforcement Learning (RL)** – learn an optimal policy for a task, e.g. maximizing expected goals or threatening play. The agent's actions could be to choose which player to pass to, whether to dribble, or to shoot.
- **Model:** A *graph-based RL* setup. For example, use a Deep Q-Network whose input is the player graph (node embeddings from a GNN) and whose outputs are actions (possible passes). Offline RL (from historical data) can be used; for instance, the Conservative Q-Learning (CQL) algorithm has been applied to event data to learn policies . Alternatively, a policy-gradient method could be used in a simulated environment with a GNN as the policy network.
- **Metrics:** Average reward (e.g. probability of scoring, as in expected goals) achieved by the policy. Compare to a baseline (historical average). If learning offline, use held-out historical sequences to test if the policy would have chosen higher-value actions (e.g. higher xT) compared to actual play.
- **Extensions:** Use a **multi-agent** RL framework where each player is an agent, and the graph captures interactions. Incorporate opponent modeling by including defensive players in the graph. One could also use *inverse RL* to infer latent team objectives from observed graphs of play.

Sources: The above ideas draw on known methods in soccer analytics and graph ML. In particular, on-ball event data (passes, shots, etc.) is often represented as a **passing network** (players as nodes, passes as edges) . Recent research applies Graph Neural Networks for tasks like match outcome prediction , embedding passes to detect phases , and player evaluation . The StatsBomb dataset has been used in these contexts; for example, a GNN model trained on Premier League StatsBomb data (758,426 events including passes, shots, tackles) demonstrates the feasibility of these graph approaches . In short, we can leverage the rich spatiotemporal event data of StatsBomb v4.0.0 to construct player/team graphs and apply ML (classification, clustering, link-prediction, anomaly detection, RL) to answer tactical and predictive questions in soccer.

ok so i am thinking of designing a problem statement where we can check the compatibility between 2 players . Like designing a formula which uses graph based calculations to arrive at a conclusion how well a combination of 2 or more players fit. Is there some way?

Absolutely—this is a great fit for graph thinking. Here's a concrete, ML-friendly framework to quantify **player compatibility** (for a pair or a small group) from StatsBomb Events. No code—just the blueprint and formulas/workflows you can implement.

Compatibility as a Graph Problem

Think of a match (or set of matches) as a **multi-relational, temporal player graph**:

- **Nodes:** players (with attributes like position, average locations, event rates). Positions and average pitch coordinates come straight from the event spec (e.g., `position`, `location`).

- **Edges (typed):** passes, carries handoffs, press/duel chains, interceptions, etc., with features like success, distance, angle, start/end locations (`pass.recipient`, `pass.end_location`, `carry.end_location`, `interception.outcome`, etc.).
- **Temporal aspect:** sequences by `timestamp` / `index` within possessions; possessions are explicitly tracked (`possession`, `possession_team`).
- **Pressure context:** “under_pressure” and “counterpress” tags enable linking defensive actions to on-ball events within 5s windows.

We’ll score a pair (A,B) with a **Compatibility Score** built from components that each measure a different kind of fit. You can compute per-match and aggregate/weight by minutes played together.

The Compatibility Score (for a pair A,B)

$$\textbf{Compat}(A, B) \;=\; w_1\mathrm{CQ} + w_2\mathrm{CS} + w_3\mathrm{SC} + w_4\mathrm{RC} + w_5\mathrm{DI} + w_6\mathrm{TS}$$

Weights can be (a) learned to predict outcomes you care about (e.g., xG differential when both are on) or (b) set by domain judgement (equal or tuned). Each component below is on [0,1] after normalization.

---

1) CQ – Connection Quality (direct synergy)

How reliably and productively do they connect?

- **Graph view:** directed pass edges A→B and B→A with features: count, success rate, distance, angle, height, pressure context (from pass attributes and `under_pressure`).
- **Compute:**
  - **Quality:** $q_{A\to B}$ = weighted success rate (weight by pass difficulty—distance/angle/height—and pressure flags).
  - **Volume-normalized:** scale by opportunities (minutes together, proximity in average locations).
  - **Bidirectional CQ:** $\mathrm{CQ} = \frac{q_{A\to B}+q_{B\to A}}{2}$.
- **Graph-ML twist:** train a simple **edge existence/quality model** (link prediction) from team graphs; **CQ is the residual** (actual − expected). Positive residual ⇒ they outperform what the system expects.

2) CS – Chain Synergy (indirect, sequence value)

Do A and B make each other more dangerous across sequences, not just direct passes?

- **Graph view:** a **possessions hypergraph** (nodes: players, hyperedges: possession sequences). Include pass+carry+dribble edges; shots as terminal nodes with `statsbomb_xg` (if available) or your xT proxy derived from shot/location features .
- **Compute:** for sequences that include both A and B in any order within N steps:
  - **Lift in value:** $\Delta = \text{xT/xG of sequence} - \text{baseline xT/xG of similar sequences without that pair.}$
  - **CS:** mean positive Δ clipped to [0,1].
- **Notes:** Use `carry` and `dribble` events to credit ball progression between the pair even without a pass .

---

3) SC – Spatial Complementarity

Do their preferred spaces **overlap (bad)** or **interlock (good)**?

- **Graph view:** discretize the pitch into zones (as per spec's locations framework) and build a **player↔zone bipartite graph** weighted by time on ball / events per zone (using `location` and `end_location`) .
- **Compute:**
  - Build two zone distributions $P_A(z), P_B(z)$.
  - **Overlap penalty:** $\text{OL} = \sum_z \min(P_A(z), P_B(z))$.
  - **Adjacency bonus:** reward when their peaks are in **adjacent** lanes (facilitates triangles and safe passing lanes).
  - **SC:** $1 - \text{OL} + \text{AdjacencyBonus}$, normalized to [0,1].

---

4) RC – Role Complementarity

Do their on-ball profiles fit (e.g., creator + runner, progressor + finisher)?

- **Graph view:** per player, a **role vector** from event mix: pass types (`switch` , `cross` , `cut-back` , through-like "technique"), carries, dribbles, pressures, duels, interceptions, shot involvement; all available in event objects .

- **Compute:** learn **player embeddings** via a graph model (e.g., team pass graph with multi-relational edges). Define a **compatibility kernel** where *complementary* roles score higher than identical ones (e.g., via a learned bilinear form $e_A^\top M e_B$). Start simple: cosine distance inverted + bonuses for creator→finisher pairs (based on labels like frequent `shot-assist` vs `first_time` shots) .
- **RC:** scale to [0,1].

---

5) DI – Defensive Interlock

How well do they act as a **press-trap** pair or cover for each other?

- **Graph view:** chains linking **pressure → (opponent on-ball event) → recovery/interception** within the 5s "counterpress" window (available as `counterpress` / `under_pressure`) .
- **Compute:** count A's pressures leading to B's recoveries/interceptions (and vice-versa), weighted by outcome quality (win ball high, create shot within k actions).
  - $\text{DI} = \frac{\text{success chains involving both}}{\text{all joint press chains}}$ (minutes-adjusted).
- **Include tackling/duel outcomes** (`duel.outcome` , `interception.outcome`) for success credit .

---

6) TS – Triangle Stability (third-man patterns)

Do they form robust **triangles** with others?

- **Graph view:** on the pass graph, compute **closed triads** that contain A and B; weight by edge quality and zone diversity.
- **Compute:** clustering coefficient restricted to A,B subgraph neighborhood, plus a **third-man run** frequency (A→X→B within t seconds; include `carry` edges as A→X handoffs) .
- **TS:** normalize by minutes/opportunities.

Workflow (end-to-end)

1. **Assemble on-pitch segments** where A and B both play. Slice events by `timestamp` , `period` , and player presence.
2. **Build graphs per segment:**

- **Pass multigraph** (directed, weighted, with features like distance/angle/height/pressure) .
- **Possession hypergraphs** for chain synergy (include pass, carry, dribble, shot nodes/edges; shots have outcomes/xG) .
- **Press-chains graph** (pressure→opponent on-ball→recovery/interception within counterpress window) .
- **Zone bipartite** for spatial distributions (from `location`, `end_location`) .

3. **Feature computation:** compute CQ, CS, SC, RC, DI, TS as above; normalize to [0,1].
4. **Weighting/Scoring:**
   - **Supervised option:** learn w_i to maximize correlation with a target when A+B are on (e.g., team xG per 90, goal diff per 90, entries into final third).
   - **Unsupervised option:** equal weights or principal-components weighting (more stable when data is sparse).
5. **Validation:**
   - **Back-test:** do top-10 compatible pairs improve your target metric vs randomly sampled pairs with similar positions?
   - **Ablations:** recompute score with each component dropped; check which components drive predictive signal.
   - **Robustness:** control for opponent quality, game state, minutes, sample size (e.g., ignore pairs < 300 shared minutes).
6. **Deployment:** use rolling windows to keep scores current; expose pair and small-group (triplets) scores to scouting or lineup tools.

Extensions (optional but powerful)

- **Small-group compatibility:** generalize to **hyperedges** for triplets/units (e.g., CB-LB-DM triangle) and re-use CS/TS.
- **Context awareness:** compute **situation-specific** scores (vs low/mid/high block; leading vs trailing; first vs second half).
- **Uncertainty:** bootstrap possessions to add confidence intervals around each component.
- **Explainability:** show top sequences and zones that most increased each component (e.g., top A→B passes under pressure).

Why this works with StatsBomb Events

- Rich pass features (recipient, angle, distance, height, crosses/cut-backs) let you rate connection difficulty and intent for **CQ/CS** .
- Carry/dribble and their end locations capture non-pass progression between the pair for **CS/TS** .
- `under_pressure` and `counterpress` flags allow tight **defensive chain** modeling for **DI** within the 5-second window .
- Positions and `location` fields support **SC** (zone maps) and role vectors for **RC** .

If you want, I can help you pick target outcomes and sketch a minimal data schema for these components (so you can implement the score and start validating it on your matches).