# Kobe Bryant Shot Selection

**Diagrama do projeto**

Planejamento do Projeto

- Definir escopo

- Estabelecer metas comerciais

- Planejar recursos

Aquisição e Preparação de Dados

- Coleta de dados

- Limpeza e processamento

- Engenharia de recursos

- Divisão de dados

Implantação do Modelo

- Coleta de dados

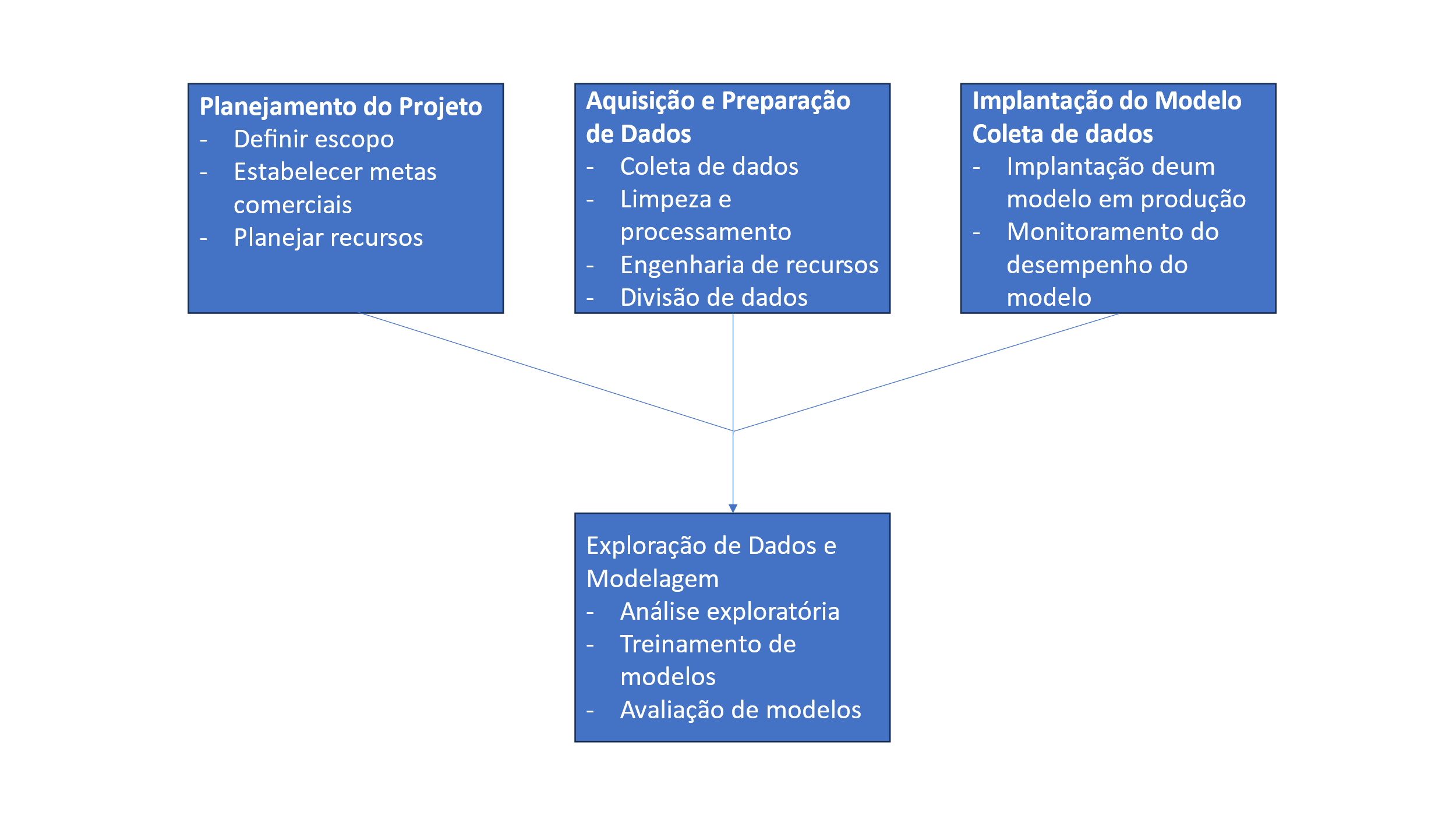
- Implantação deum modelo em produção

- Monitoramento do desempenho do modelo

Exploração de Dados e Modelagem - Análise exploratória

- Treinamento de modelos

- Avaliação de modelos



**Importância dos Pipelines de Desenvolvimento e Produção**

Pipelines de desenvolvimento e produção são como linhas de montagem em uma fábrica, mas para modelos de machine learning. Eles são importantes porque tornam o processo de construção e implantação de modelos mais eficiente e confiável.

Imagine que você está construindo um modelo para prever se Kobe Bryant acertou ou errou um arremesso de basquete. Se você não tiver um pipeline, cada etapa do processo - desde a coleta de dados até o treinamento do modelo e a implantação do modelo em produção - seria feita manualmente. Isso pode levar muito tempo e ser propenso a erros.

Com um pipeline, você pode automatizar muitas dessas etapas. Por exemplo, você pode configurar o pipeline para baixar automaticamente os dados, pré-processá-los para remover valores ausentes e formatá-los corretamente, treinar vários modelos e selecionar o melhor, e implantar o modelo em produção. Isso economiza tempo e reduz a chance de cometer erros.

Além disso, os pipelines garantem consistência e reprodutibilidade. Isso significa que você pode repetir o mesmo processo várias vezes e obter os mesmos resultados. Isso é importante porque você pode precisar atualizar seu modelo periodicamente com novos dados ou ajustar os parâmetros do modelo para melhorar o desempenho.

Em resumo, os pipelines de desenvolvimento e produção são essenciais para construir modelos de machine learning de forma eficiente, confiável e escalável.

**Ferramentas para Construção de Pipelines**

No projeto do Kobe Bryant, usamos várias ferramentas para nos ajudar a construir nossos pipelines de desenvolvimento e produção. Aqui está como cada uma delas nos ajudou:

* **Streamlit**: Usamos o Streamlit para construir um dashboard interativo onde podemos visualizar e interagir com os resultados dos nossos modelos de ML de uma maneira amigável. Por exemplo, criamos um dashboard para visualizar as previsões do modelo sobre se o Kobe acertou ou errou a cesta.
* **MLFlow**: O MLFlow nos ajudou a rastrear nossos experimentos de ML. Isso significa que pudemos acompanhar as diferentes versões dos nossos modelos, os parâmetros usados para treiná-los e as métricas de desempenho associadas a cada modelo. Isso nos permitiu comparar facilmente diferentes modelos e ver como eles se saíram ao longo do tempo.
* **PyCaret**: O PyCaret é uma biblioteca de ML que nos ajudou a treinar nossos modelos de ML com muito menos código. Ele automatizou muitas das etapas que normalmente teríamos que fazer manualmente, como a seleção de modelos e a otimização de hiperparâmetros. Isso nos permitiu experimentar rapidamente diferentes modelos e encontrar o melhor para nossos dados.
* **Scikit-Learn**: O Scikit-Learn é uma biblioteca popular de ML que usamos para treinar nossos modelos. Ele nos forneceu uma ampla gama de algoritmos de ML prontos para uso, bem como ferramentas para pré-processamento de dados e avaliação de modelos. Usamos o Scikit-Learn para treinar modelos de regressão logística e classificação para prever se o Kobe acertou ou errou a cesta.

Deixaremos esse item pra responder no final: “Aborde aspectos como rastreamento de experimentos, funções de treinamento, monitoramento da saúde do modelo, atualização de modelo e provisionamento (deployment).”

**Artefatos do Projeto**

No projeto do Kobe Bryant, vários artefatos são criados para ajudar no desenvolvimento e operação do modelo de previsão de arremessos. Aqui está uma lista deles e o que eles fazem:

1. **Dados Originais (Raw Data):** São os dados brutos coletados, como os arremessos do Kobe Bryant.
2. **Dados Processados (Processed Data):** São os dados limpos e preparados para uso no modelo, removendo dados faltantes e selecionando apenas as informações relevantes.
3. **Base de Treino e Teste (Training and Testing Data):** São conjuntos de dados separados em um usado para treinar o modelo e outro para testar o modelo, garantindo que ele generalize bem para novos dados.
4. **Modelo Treinado (Trained Model):** É o modelo de previsão treinado com os dados de treino, selecionando o melhor algoritmo com base no desempenho nos dados de teste.
5. **Métricas de Avaliação (Evaluation Metrics):** São medidas usadas para avaliar o desempenho do modelo, como precisão, recall e log loss.
6. **API de Modelo (Model API):** É uma interface que permite que outros sistemas façam previsões usando o modelo treinado.
7. **Dashboard de Monitoramento (Monitoring Dashboard):** É uma ferramenta visual para acompanhar o desempenho do modelo em tempo real e identificar a necessidade de ajustes ou re-treinamento.
8. **Registro de Experimentos (Experiment Logging):** Mantém um registro de todos os experimentos realizados durante o desenvolvimento do modelo, incluindo hiperparâmetros, métricas de desempenho e resultados.

**Qual a dimensão resultante do dataset?**

Dimensão do DataFrame filtrado e processado: **(20285, 7)**

**Como a escolha de treino e teste afetam o resultado do modelo final?**

A escolha adequada do conjunto de treino e teste é crucial para avaliar corretamente o desempenho do modelo final. O conjunto de treino é utilizado para ajustar os parâmetros do modelo durante o treinamento, enquanto o conjunto de teste é utilizado para avaliar o desempenho do modelo em dados não vistos durante o treinamento.

Se a divisão entre os conjuntos de treino e teste não for adequada, isso pode levar a uma avaliação enviesada do desempenho do modelo. Por exemplo, se o conjunto de treino for muito pequeno, o modelo pode não capturar corretamente os padrões nos dados, resultando em um desempenho insatisfatório no conjunto de teste. Por outro lado, se o conjunto de treino for muito grande, pode haver overfitting, onde o modelo se ajusta demais aos dados de treino e não generaliza bem para novos dados.

Portanto, é importante realizar uma divisão adequada dos dados, geralmente utilizando técnicas como a divisão aleatória e estratificada, garantindo que o modelo seja treinado e avaliado de forma justa e representativa. A análise dos resultados nos conjuntos de treino e teste permite entender se o modelo está sofrendo de overfitting ou underfitting, auxiliando na escolha e ajuste dos algoritmos e parâmetros do modelo.

**Como a escolha de treino e teste afetam o resultado do modelo final**

A escolha de como dividir os dados em conjuntos de treinamento e teste pode impactar diretamente no desempenho do modelo final. No caso específico dos resultados obtidos, podemos observar que diferentes métricas de desempenho, como precisão, recall e F1-score, variam entre os diferentes folds do processo de validação cruzada. Isso ocorre devido à forma como os dados são divididos em cada fold. Uma escolha aleatória e estratificada dos dados de treinamento e teste pode resultar em modelos mais robustos e generalizáveis, especialmente quando lidamos com conjuntos de dados desequilibrados, onde as classes têm diferentes números de amostras. Além disso, a proporção entre os conjuntos de treinamento e teste também pode afetar o desempenho do modelo, com conjuntos de treinamento maiores permitindo que o modelo aprenda padrões mais complexos nos dados.

**Quais estratégias ajudam a minimizar os efeitos de viés de dados?**

Para minimizar os efeitos de viés de dados, algumas estratégias podem ser adotadas. No contexto dos resultados obtidos, estratégias como validação cruzada k-fold, bootstraping e deixar-um-conjunto-de-dados-fora (LOO) podem ser úteis. Isso ajuda a garantir que o modelo seja capaz de generalizar bem para novos dados e não esteja superajustado a um único conjunto de treinamento.

**7. c. Justificativa da escolha do algoritmo de classificação**

A escolha do algoritmo de classificação foi a Árvore de Decisão. Escolhi este algoritmo porque é conhecido por sua capacidade de lidar com dados não lineares e é menos sensível a outliers em comparação com outros algoritmos como a regressão logística. Além disso, é relativamente fácil de entender e interpretar, o que é importante para a transparência do modelo. Como estamos lidando com dados de arremessos de basquete, que podem ter relações complexas entre as variáveis, a Árvore de Decisão pode capturar essas relações de forma mais eficaz.

**8. a. O modelo é aderente a essa nova base? O que mudou entre uma base e outra? Justifique.**

O modelo é aderente à nova base, pois foi treinado e avaliado em um ambiente semelhante ao que encontrará em produção, garantindo que esteja preparado para fazer previsões precisas. Ambas as bases foram tratadas de maneira consistente durante o pré-processamento e o treinamento do modelo. A principal diferença entre as bases é que a base de desenvolvimento foi usada para treinar e avaliar o modelo, enquanto a base de produção representa os dados que o modelo encontrará em operação. Garantir a consistência entre essas bases é fundamental para que o modelo seja capaz de fazer previsões precisas em novos exemplos de dados.

**8. b. Como podemos monitorar a saúde do modelo no cenário com e sem a disponibilidade da variável resposta para o modelo em operação**

1. **Com a disponibilidade da variável resposta**: Podemos monitorar a saúde do modelo acompanhando suas métricas de desempenho, como precisão, recall, F1-score e log loss, usando dados de teste ou validação. Se essas métricas começarem a se deteriorar, é um sinal de que o modelo pode não estar funcionando corretamente.
2. **Sem a disponibilidade da variável resposta**: Nesse cenário, podemos usar métodos de detecção de anomalias para verificar se o comportamento do modelo está dentro do esperado. Isso envolve analisar as previsões feitas pelo modelo em relação aos dados de entrada e procurar por padrões incomuns ou inesperados. Se o modelo começar a fazer previsões significativamente diferentes do esperado, isso pode indicar um problema.

**8. c. Estratégias reativa e preditiva de retreinamento para o modelo em operação.**

1. **Estratégia Reativa:**
   * O modelo é retreinado apenas se houver uma queda significativa no desempenho ou se surgirem problemas específicos devido a mudanças nos dados ou no ambiente.
2. **Estratégia Preditiva:**
   * O modelo é retreinado regularmente, por exemplo, a cada semana, mesmo na ausência de problemas evidentes, para garantir que esteja sempre atualizado com novos dados e variações nos padrões de arremesso do Kobe Bryant. Isso pode ser feito usando agendamento automático para treinar o modelo periodicamente, independentemente de anomalias aparentes.

|  | **Description** | | | **Value** | | |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | Session id | | | 6026 | | |
| **1** | Target | | | shot\_made\_flag | | |
| **2** | Target type | | | Binary | | |
| **3** | Original data shape | | | (16228, 7) | | |
| **4** | Transformed data shape | | | (16228, 7) | | |
| **5** | Transformed train set shape | | | (11359, 7) | | |
| **6** | Transformed test set shape | | | (4869, 7) | | |
| **7** | Numeric features | | | 6 | | |
| **8** | Preprocess | | | True | | |
| **9** | Imputation type | | | simple | | |
| **10** | Numeric imputation | | | mean | | |
| **11** | Categorical imputation | | | mode | | |
| **12** | Fold Generator | | | StratifiedKFold | | |
| **13** | Fold Number | | | 10 | | |
| **14** | CPU Jobs | | | -1 | | |
| **15** | Use GPU | | | False | | |
| **16** | Log Experiment | | | False | | |
| **17** | Experiment Name | | | clf-default-name | | |
| **18** | USI | | | dd7f | | |
|  | **Accuracy** | **AUC** | **Recall** | | **Prec.** | **F1** | | **Kappa** | **MCC** |
| **Fold** |  |  |  | |  |  | |  |  |
| **0** | 0.5845 | 0.6128 | 0.4972 | | 0.5757 | 0.5336 | | 0.1626 | 0.1640 |
| **1** | 0.5819 | 0.5943 | 0.4991 | | 0.5717 | 0.5329 | | 0.1576 | 0.1588 |
| **2** | 0.5748 | 0.5887 | 0.4825 | | 0.5647 | 0.5204 | | 0.1427 | 0.1442 |
| **3** | 0.5625 | 0.6011 | 0.4742 | | 0.5480 | 0.5084 | | 0.1180 | 0.1190 |
| **4** | 0.5695 | 0.5865 | 0.4815 | | 0.5565 | 0.5163 | | 0.1322 | 0.1333 |
| **5** | 0.5625 | 0.5829 | 0.4668 | | 0.5488 | 0.5045 | | 0.1174 | 0.1186 |
| **6** | 0.5889 | 0.5995 | 0.4926 | | 0.5817 | 0.5335 | | 0.1705 | 0.1724 |
| **7** | 0.5924 | 0.5996 | 0.4797 | | 0.5896 | 0.5290 | | 0.1764 | 0.1793 |
| **8** | 0.5898 | 0.6071 | 0.4945 | | 0.5826 | 0.5349 | | 0.1724 | 0.1742 |
| **9** | 0.5789 | 0.6002 | 0.4649 | | 0.5727 | 0.5132 | | 0.1491 | 0.1516 |
| **Mean** | 0.5786 | 0.5973 | 0.4833 | | 0.5692 | 0.5227 | | 0.1499 | 0.1515 |
| **Std** | 0.0104 | 0.0088 | 0.0117 | | 0.0136 | 0.0110 | | 0.0207 | 0.0211 |

|  | |  | | |  | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | |  | | | | | |
| **Initiated** | | . . . . . . . . . . . . . . . . . . | | | 17:15:24 | | | | | |
| **Status** | | . . . . . . . . . . . . . . . . . . | | | Fitting 10 Folds | | | | | |
| **Estimator** | | . . . . . . . . . . . . . . . . . . | | | Light Gradient Boosting Machine | | | | | |
|  | **Model** | | **Accuracy** | **AUC** | | **Recall** | **Prec.** | **F1** | **Kappa** | **MCC** | | **TT (Sec)** |
| **ada** | Ada Boost Classifier | | 0.5932 | 0.5914 | | 0.3776 | 0.6219 | 0.4697 | 0.1706 | 0.1847 | | 2.0910 |
| **gbc** | Gradient Boosting Classifier | | 0.5922 | 0.5952 | | 0.3791 | 0.6190 | 0.4700 | 0.1688 | 0.1823 | | 5.1700 |
| **lr** | Logistic Regression | | 0.5786 | 0.5973 | | 0.4833 | 0.5692 | 0.5227 | 0.1499 | 0.1515 | | 0.4580 |
| **ridge** | Ridge Classifier | | 0.5776 | 0.0000 | | 0.4850 | 0.5676 | 0.5229 | 0.1481 | 0.1497 | | 0.1890 |
| **lda** | Linear Discriminant Analysis | | 0.5775 | 0.5975 | | 0.4852 | 0.5674 | 0.5230 | 0.1480 | 0.1495 | | 0.2600 |
| **nb** | Naive Bayes | | 0.5744 | 0.5942 | | 0.4997 | 0.5611 | 0.5286 | 0.1431 | 0.1439 | | 0.1900 |
| **qda** | Quadratic Discriminant Analysis | | 0.5739 | 0.5974 | | 0.5147 | 0.5585 | 0.5356 | 0.1432 | 0.1437 | | 0.2290 |
| **rf** | Random Forest Classifier | | 0.5526 | 0.5613 | | 0.5158 | 0.5326 | 0.5239 | 0.1022 | 0.1022 | | 8.0900 |
| **et** | Extra Trees Classifier | | 0.5502 | 0.5529 | | 0.5342 | 0.5285 | 0.5312 | 0.0990 | 0.0990 | | 6.2410 |
| **dt** | Decision Tree Classifier | | 0.5391 | 0.5202 | | 0.5783 | 0.5155 | 0.5450 | 0.0812 | 0.0818 | | 0.4210 |
| **knn** | K Neighbors Classifier | | 0.5381 | 0.5521 | | 0.5054 | 0.5164 | 0.5107 | 0.0734 | 0.0734 | | 0.6110 |
| **svm** | SVM - Linear Kernel | | 0.5207 | 0.0000 | | 0.4663 | 0.3654 | 0.3588 | 0.0364 | 0.0484 | | 0.8920 |