Advanced AI algorithms like time series forecasting and ensemble methods can be used to improve predictive accuracy in a variety of ways.

PHASE 2 DOCUMENTATION

PROJECT: *Titanic Machine Learning From - Disaster*

Time series forecasting algorithms are used to predict future values of a time series based on its past values. This can be useful for predicting things like customer demand, sales trends, and stock prices. Time series forecasting algorithms can be particularly effective when the data has a strong seasonal or cyclical pattern. Ensemble methods combine the predictions of multiple machine learning models to produce a more accurate prediction. This can be done in a variety of ways, such as averaging the predictions of the individual models or using a weighted average. Ensemble methods can be particularly effective for improving predictive accuracy when the data is noisy or complex.

Here are some specific examples of how time series forecasting and ensemble methods can be used to improve predictive accuracy:

- Predicting customer demand: A retailer can use time series forecasting to predict how much demand there will be for a particular product in the future. This information can be used to optimize inventory levels and avoid stockouts.
- Predicting sales trends: A company can use time series forecasting to predict future sales trends. This information can be used to make informed decisions about product development, marketing, and staffing.
- Predicting stock prices: Investors can use time series forecasting to predict future stock prices. This information can be used to make informed investment decisions.
- Predicting fraud: Banks can use ensemble methods to predict which transactions are likely to be fraudulent. This information can be used to prevent fraud and protect customers' accounts.
- Predicting medical diagnoses: Doctors can use ensemble methods to predict which patients are most likely to have a particular disease. This information can be used to provide early diagnosis and treatment.

Time series forecasting and ensemble methods are powerful tools that can be used to improve predictive accuracy in a variety of applications. However, it is important to choose the right algorithm for the specific problem at hand. It is also important to note that these algorithms are only as good as the data they are trained on. Therefore, it is important to have high-quality data that is representative of the problem you are trying to solve.

Here are some additional tips for improving predictive accuracy:

- Use a variety of data sources: The more data you have, the better your model will be able to learn and make accurate predictions. Try to use data from a variety of sources to get a more complete picture of the problem you are trying to solve.
- Clean and prepare your data: Before you train your model, it is important to clean and prepare your data. This includes removing outliers, handling missing values, and scaling your data.
- Use feature engineering: Feature engineering is the process of creating new features from existing data. This can help to improve the performance of your model by making the data more informative.
- Tune your model's hyperparameters: Hyperparameters are parameters that control the behavior of a machine learning model. Tuning your model's hyperparameters can help to improve predictive accuracy.
- Use a validation set: A validation set is a set of data that is not used to train your model. It is used to evaluate the performance of your model on unseen data. This can help to prevent overfitting and improve predictive accuracy.

By following these tips, you can improve the predictive accuracy of your machine learning models.

Time series forecasting algorithms are a type of predictive algorithm that are specifically designed to forecast future values of a time series based on its past values. Time series data is data that is collected over time, such as daily sales data, stock prices, or weather data.
Ensemble methods are a type of machine learning algorithm that combine multiple machine learning models to produce a more accurate prediction. Ensemble methods work by training multiple models on different subsets of the training data and then averaging or combining the predictions of the individual models.

Both time series forecasting algorithms and ensemble methods can be used to improve the predictive accuracy of AI models.

Time series forecasting algorithms can be used to improve the predictive accuracy of AI models by taking into account the temporal patterns in the data. For example, a time series forecasting algorithm could be used to predict future sales based on historical sales data, seasonal trends, and other factors.
Ensemble methods can be used to improve the predictive accuracy of AI models by reducing the risk of overfitting. Overfitting is a problem that occurs when a machine learning model learns the training data too well and is unable to generalize to new data. Ensemble methods work by combining the predictions of multiple models, which reduces the risk of overfitting and can improve the overall predictive accuracy of the model.

Here are some specific examples of advanced AI algorithms that can be used for time series forecasting and ensemble methods:

- Time series forecasting algorithms:
  - ARIMA (Autoregressive Integrated Moving Average)
  - Prophet
  - Exponential smoothing
- Ensemble methods:
  - Random forests
  - Gradient boosting machines
  - Stacking

Which algorithm is best to use will depend on the specific problem that you are trying to solve and the characteristics of your data. It is often a good idea to experiment with different algorithms to see which one produces the best results on your data set.

How to explore advanced AI algorithms for improved predictive accuracy

If you are interested in exploring advanced AI algorithms for improved predictive accuracy, here are some steps that you can take:

1. Identify the problem that you are trying to solve. What are you trying to predict? What kind of data do you have available?
2. Research different AI algorithms. There are many different AI algorithms available, each with its own strengths and weaknesses. Read research papers and blog posts to learn about different algorithms and how they have been used to solve similar problems.
3. Choose an algorithm to try. Once you have a good understanding of different AI algorithms, choose one to try on your data. There are many open source AI libraries available, such as scikit-learn and TensorFlow.
4. Train the model. Once you have chosen an algorithm, you need to train it on your data. This involves feeding the algorithm your data and letting it learn the patterns in the data.
5. Evaluate the model. Once the model is trained, you need to evaluate its performance on a held-out test set. This will give you an idea of how well the model will generalize to new data.
6. Deploy the model. If the model performs well on the test set, you can deploy it to production. This means making the model available so that it can be used to make predictions on new data.

It is important to note that exploring advanced AI algorithms can be a complex and time-consuming process. If you are not familiar with AI, it is a good idea to consult with an expert.

Time Series Forecasting

Time series forecasting is a type of predictive modeling that uses historical data to predict future values of a time series. Time series data is data that is collected over time, such as daily sales data, monthly stock prices, or annual temperature data.

Time series forecasting algorithms can be used to predict a variety of things, such as:

- Demand for products and services
- Customer churn
- Equipment failure
- Financial market movements
- Natural disasters

Some common time series forecasting algorithms include:

- Autoregressive integrated moving average (ARIMA): ARIMA is a statistical model that is often used to forecast time series data that is stationary, meaning that its mean and variance do not change over time.
- Prophet: Prophet is a forecasting algorithm developed by Facebook that is designed to be easy to use and accurate for a wide range of time series data.
- Exponential smoothing: Exponential smoothing is a family of forecasting algorithms that weight recent observations more heavily than older observations.
- Neural networks: Neural networks are a type of machine learning algorithm that can be used to forecast time series data by learning patterns from historical data.

Ensemble Methods

Ensemble methods are a type of machine learning algorithm that combines multiple machine learning models to produce a more accurate prediction. Ensemble methods work by training multiple models on different subsets of the training data and then averaging the predictions of the models.

Some common ensemble methods include:

- Bagging: Bagging works by creating multiple subsets of the training data by sampling with replacement. Each subset is then used to train a separate model. The predictions of the models are then averaged to produce the final prediction.
- Boosting: Boosting works by training multiple models sequentially. Each model is trained on the residuals of the previous model. The predictions of the models are then weighted and summed to produce the final prediction.
- Stacking: Stacking works by training a meta-model to predict the output of multiple base models. The base models are trained on the training data and the meta-model is trained on the predictions of the base models.

How to Use Advanced AI Algorithms for Improved Predictive Accuracy

To use advanced AI algorithms for improved predictive accuracy, you need to:

1. Choose the right algorithm for your problem. There are many different advanced AI algorithms available, so it is important to choose the one that is

best suited for your specific problem. Consider the type of data you have, the type of predictions you want to make, and the level of accuracy you need.

2. Prepare your data. Before you can train a model, you need to prepare your data. This may involve cleaning the data, handling missing values, and transforming the data into a format that is compatible with the algorithm you have chosen.
3. Train the model. Once you have prepared your data, you can train the model. This process can be time-consuming, depending on the size and complexity of your dataset.
4. Evaluate the model. Once the model is trained, you need to evaluate its performance on a held-out test set. This will help you to determine how well the model will generalize to new data.
5. Deploy the model. Once you are satisfied with the performance of the model, you can deploy it to production. This may involve integrating the model into a software application or making it available as a web service.

Here are some additional tips for improving the predictive accuracy of your models:

- Use a large and diverse dataset. The more data you have, the better your model will be able to learn the patterns in your data.
- Use feature engineering to create new features from your existing data. This can help to improve the predictive power of your model.
- Use regularization to prevent overfitting. Overfitting occurs when a model learns the training data too well and is unable to generalize to new data.
- Use ensemble methods. Ensemble methods can often achieve better predictive accuracy than individual models.

Examples of Using Advanced AI Algorithms for Improved Predictive Accuracy

Advanced AI algorithms are being used to improve predictive accuracy in a variety of industries. Here are a few examples:

- Finance: Advanced AI algorithms are being used to predict stock prices, foreign exchange rates, and other financial market movements.
- Retail: Advanced AI algorithms are being used to predict demand for products, optimize inventory levels, and personalize marketing campaigns.
- Healthcare: Advanced AI algorithms are being used to predict patient outcomes, diagnose diseases, and develop new treatments.

- Manufacturing: Advanced AI algorithms are being used to predict equipment failure, optimize production schedules, and improve quality control.

Python Programming:

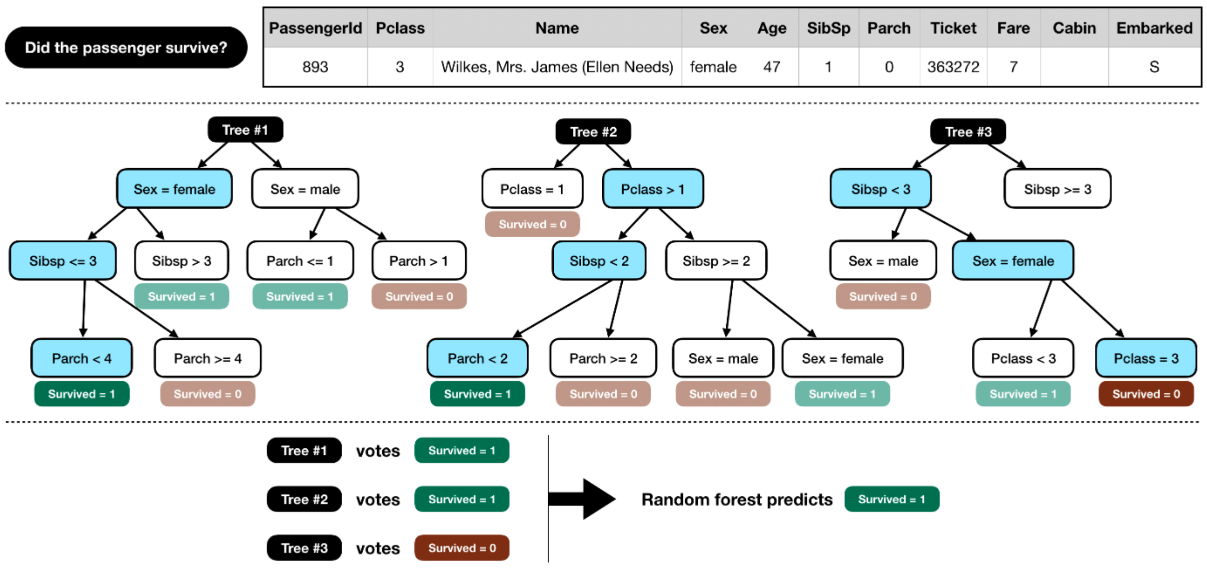
```
# Load in our libraries
import pandas as pd
import numpy as np
import re
import sklearn
import xgboost as xgb
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls

import warnings
warnings.filterwarnings('ignore')

# Going to use these 5 base models for the stacking
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, ExtraTreesClassifier)
from sklearn.svm import SVC
from sklearn.cross_validation import KFold
# Load in the train and test datasets
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')

# Store our passenger ID for easy access
PassengerId = test['PassengerId']
```

```
train.head(3)
full_data = [train, test]

# Some features of my own that I have added in
# Gives the length of the name
train['Name_length'] = train['Name'].apply(len)
test['Name_length'] = test['Name'].apply(len)
# Feature that tells whether a passenger had a cabin on the Titanic
train['Has_Cabin'] = train["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
test['Has_Cabin'] = test["Cabin"].apply(lambda x: 0 if type(x) == float else 1)

# Feature engineering steps taken from Sina
# Create new feature FamilySize as a combination of SibSp and Parch
for dataset in full_data:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1
# Create new feature IsAlone from FamilySize
for dataset in full_data:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1
# Remove all NULLS in the Embarked column
for dataset in full_data:
    dataset['Embarked'] = dataset['Embarked'].fillna('S')
# Remove all NULLS in the Fare column and create a new feature CategoricalFare
for dataset in full_data:
    dataset['Fare'] = dataset['Fare'].fillna(train['Fare'].median())
train['CategoricalFare'] = pd.qcut(train['Fare'], 4)
# Create a New feature CategoricalAge
for dataset in full_data:
    age_avg = dataset['Age'].mean()
    age_std = dataset['Age'].std()
    age_null_count = dataset['Age'].isnull().sum()
 age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std,
size=age_null_count)
    dataset['Age'][np.isnan(dataset['Age'])] = age_null_random_list
    dataset['Age'] = dataset['Age'].astype(int)
train['CategoricalAge'] = pd.cut(train['Age'], 5)
# Define function to extract titles from passenger names
def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    # If the title exists, extract and return it.
    if title_search:
        return title_search.group(1)
    return ""
# Create a new feature Title, containing the titles of passenger names
for dataset in full_data:
    dataset['Title'] = dataset['Name'].apply(get_title)
# Group all non-common titles into one single grouping "Rare"
for dataset in full_data:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr',
'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

for dataset in full_data:
    # Mapping Sex
    dataset['Sex'] = dataset['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

    # Mapping titles
    title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)
 # Mapping Embarked
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

    # Mapping Fare
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare']
= 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare']
                  = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

    # Mapping Age
    dataset.loc[ dataset['Age'] <= 16, 'Age']                                    = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age'] = 4 ;
# Feature selection
drop_elements = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']
train = train.drop(drop_elements, axis = 1)
train = train.drop(['CategoricalAge', 'CategoricalFare'], axis = 1)
test  = test.drop(drop_elements, axis = 1)
train.head(3)
colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(train.astype(float).corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
g = sns.pairplot(train[[u'Survived', u'Pclass', u'Sex', u'Age', u'Parch', u'Fare',
u'Embarked',
       u'FamilySize', u'Title']], hue='Survived', palette = 'seismic',size=1.2,diag_kind =
'kde',diag_kws=dict(shade=True),plot_kws=dict(s=10) )
g.set(xticklabels=[])
# Some useful parameters which will come in handy later on
ntrain = train.shape[0]
ntest = test.shape[0]
SEED = 0 # for reproducibility
NFOLDS = 5 # set folds for out-of-fold prediction
kf = KFold(ntrain, n_folds= NFOLDS, random_state=SEED)

# Class to extend the Sklearn classifier
class SklearnHelper(object):
    def __init__(self, clf, seed=0, params=None):
        params['random_state'] = seed
        self.clf = clf(**params)

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def predict(self, x):
        return self.clf.predict(x)

    def fit(self,x,y):
        return self.clf.fit(x,y)

    def feature_importances(self,x,y):
        print(self.clf.fit(x,y).feature_importances_)
def get_oof(clf, x_train, y_train, x_test):
    oof_train = np.zeros((ntrain,))
    oof_test = np.zeros((ntest,))
    oof_test_skf = np.empty((NFOLDS, ntest))

    for i, (train_index, test_index) in enumerate(kf):
        x_tr = x_train[train_index]
        y_tr = y_train[train_index]
        x_te = x_train[test_index]

        clf.train(x_tr, y_tr)

        oof_train[test_index] = clf.predict(x_te)
        oof_test_skf[i, :] = clf.predict(x_test)

    oof_test[:] = oof_test_skf.mean(axis=0)
    return oof_train.reshape(-1, 1), oof_test.reshape(-1, 1)
# Put in our parameters for said classifiers
# Random Forest parameters
rf_params = {
    'n_jobs': -1,
    'n_estimators': 500,
     'warm_start': True,
     #'max_features': 0.2,
    'max_depth': 6,
    'min_samples_leaf': 2,
    'max_features' : 'sqrt',
    'verbose': 0
}

# Extra Trees Parameters
```

```
et_params = {
    'n_jobs': -1,
    'n_estimators':500,
    #'max_features': 0.5,
    'max_depth': 8,
    'min_samples_leaf': 2,
    'verbose': 0
}

# AdaBoost parameters
ada_params = {
    'n_estimators': 500,
    'learning_rate' : 0.75
}

# Gradient Boosting parameters
gb_params = {
    'n_estimators': 500,
     #'max_features': 0.2,
 'max_depth': 5,
    'min_samples_leaf': 2,
    'verbose': 0
}

# Support Vector Classifier parameters
svc_params = {
    'kernel' : 'linear',
    'C' : 0.025
    }
# Create 5 objects that represent our 4 models
rf = SklearnHelper(clf=RandomForestClassifier, seed=SEED, params=rf_params)
et = SklearnHelper(clf=ExtraTreesClassifier, seed=SEED, params=et_params)
ada = SklearnHelper(clf=AdaBoostClassifier, seed=SEED, params=ada_params)
gb = SklearnHelper(clf=GradientBoostingClassifier, seed=SEED,
params=gb_params)
svc = SklearnHelper(clf=SVC, seed=SEED, params=svc_params)
# Create Numpy arrays of train, test and target ( Survived) dataframes to feed into
our models
```

```
y_train = train['Survived'].ravel()
train = train.drop(['Survived'], axis=1)
x_train = train.values # Creates an array of the train data
x_test = test.values # Creats an array of the test data
# Create our OOF train and test predictions. These base results will be used as new
features
et_oof_train, et_oof_test = get_oof(et, x_train, y_train, x_test) # Extra Trees
rf_oof_train, rf_oof_test = get_oof(rf,x_train, y_train, x_test) # Random Forest
ada_oof_train, ada_oof_test = get_oof(ada, x_train, y_train, x_test) # AdaBoost
gb_oof_train, gb_oof_test = get_oof(gb,x_train, y_train, x_test) # Gradient Boost
svc_oof_train, svc_oof_test = get_oof(svc,x_train, y_train, x_test) # Support Vector
Classifier

print("Training is complete")
rf_feature = rf.feature_importances(x_train,y_train)
et_feature = et.feature_importances(x_train, y_train)
ada_feature = ada.feature_importances(x_train, y_train)
gb_feature = gb.feature_importances(x_train,y_train)
[ 0.12512537  0.20195675  0.03187994  0.02117736  0.0720603   0.02351993
  0.10877493  0.06461801  0.06974652  0.01355575  0.26758514]
[ 0.12082488  0.37460384  0.02701211  0.01713043  0.05593931  0.02854512
  0.04782111  0.08303969  0.04500919  0.02174382  0.1783305 ]
[ 0.028  0.012  0.02   0.062  0.04   0.01   0.69   0.014  0.05   0.004
  0.07 ]
[ 0.07626914  0.03373915  0.10207353  0.03738547  0.10223908  0.04940839
  0.39897827  0.01836958  0.07035654  0.02056481  0.09061604]
rf_features = [0.10474135,  0.21837029,  0.04432652,  0.02249159,  0.05432591,
0.02854371
  ,0.07570305,  0.01088129 , 0.24247496,  0.13685733 , 0.06128402]
et_features = [ 0.12165657,  0.37098307  ,0.03129623 , 0.01591611 , 0.05525811 ,
0.028157
  ,0.04589793 , 0.02030357 , 0.17289562 , 0.04853517,  0.08910063]
ada_features = [0.028 ,   0.008  ,      0.012    ,     0.05866667,   0.032 ,       0.008
  ,0.04666667 ,  0.     ,      0.05733333,   0.73866667,   0.01066667]
gb_features = [ 0.06796144 , 0.03889349 , 0.07237845 , 0.02628645 , 0.11194395,
0.04778854
  ,0.05965792 , 0.02774745,  0.07462718,  0.4593142 ,  0.01340093]
cols = train.columns.values
```

```
# Create a dataframe with features
feature_dataframe = pd.DataFrame( {'features': cols,
     'Random Forest feature importances': rf_features,
     'Extra Trees  feature importances': et_features,
      'AdaBoost feature importances': ada_features,
    'Gradient Boost feature importances': gb_features
    })
# Scatter plot
trace = go.Scatter(
    y = feature_dataframe['Random Forest feature importances'].values,
    x = feature_dataframe['features'].values,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref = 1,
        size = 25,
#       size= feature_dataframe['AdaBoost feature importances'].values,
        #color = np.random.randn(500), #set color equal to a variable
        color = feature_dataframe['Random Forest feature importances'].values,
        colorscale='Portland',
        showscale=True
    ),
    text = feature_dataframe['features'].values
)
data = [trace]

layout= go.Layout(
    autosize= True,
    title= 'Random Forest Feature Importance',
    hovermode= 'closest',
#    xaxis= dict(
#        title= 'Pop',
#        ticklen= 5,
#        zeroline= False,
#        gridwidth= 2,
#    ),
    yaxis=dict(
        title= 'Feature Importance',
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig,filename='scatter2010')

# Scatter plot
trace = go.Scatter(
    y = feature_dataframe['Extra Trees  feature importances'].values,
    x = feature_dataframe['features'].values,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref = 1,
        size = 25,
#       size= feature_dataframe['AdaBoost feature importances'].values,
        #color = np.random.randn(500), #set color equal to a variable
        color = feature_dataframe['Extra Trees  feature importances'].values,
        colorscale='Portland',
        showscale=True
    ),
    text = feature_dataframe['features'].values
)
data = [trace]

layout= go.Layout(
    autosize= True,
    title= 'Extra Trees Feature Importance',
    hovermode= 'closest',
#     xaxis= dict(
#         title= 'Pop',
#         ticklen= 5,
#         zeroline= False,
#         gridwidth= 2,
#     ),
    yaxis=dict(
        title= 'Feature Importance',
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig,filename='scatter2010')
# Scatter plot
trace = go.Scatter(
    y = feature_dataframe['AdaBoost feature importances'].values,
    x = feature_dataframe['features'].values,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref = 1,
        size = 25,
#       size= feature_dataframe['AdaBoost feature importances'].values,
        #color = np.random.randn(500), #set color equal to a variable
        color = feature_dataframe['AdaBoost feature importances'].values,
        colorscale='Portland',
        showscale=True
    ),
    text = feature_dataframe['features'].values
)
data = [trace]

layout= go.Layout(
    autosize= True,
title= 'AdaBoost Feature Importance',
    hovermode= 'closest',
#     xaxis= dict(
#         title= 'Pop',
#         ticklen= 5,
#         zeroline= False,
#         gridwidth= 2,
#     ),
    yaxis=dict(
        title= 'Feature Importance',
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig,filename='scatter2010')

# Scatter plot
trace = go.Scatter(
    y = feature_dataframe['Gradient Boost feature importances'].values,
    x = feature_dataframe['features'].values,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref = 1,
        size = 25,
        size= feature_dataframe['AdaBoost feature importances'].values,
        #color = np.random.randn(500), #set color equal to a variable
        color = feature_dataframe['Gradient Boost feature importances'].values,

colorscale='Portland',
        showscale=True
    ),
    text = feature_dataframe['features'].values
)
data = [trace]

layout= go.Layout(
    autosize= True,
    title= 'Gradient Boosting Feature Importance',
    hovermode= 'closest',
#     xaxis= dict(
#         title= 'Pop',
#         ticklen= 5,
#         zeroline= False,
#         gridwidth= 2,
#    ),
    yaxis=dict(
        title= 'Feature Importance',
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig,filename='scatter2010')
# Create the new column containing the average of values

feature_dataframe['mean'] = feature_dataframe.mean(axis= 1) # axis = 1 computes
the mean row-wise
feature_dataframe.head(3)
y = feature_dataframe['mean'].values
x = feature_dataframe['features'].values
data = [go.Bar(
            x= x,
             y= y,
            width = 0.5,
            marker=dict(
               color = feature_dataframe['mean'].values,
            colorscale='Portland',
            showscale=True,
            reversescale = False
            ),
            opacity=0.6
        )]

layout= go.Layout(
    autosize= True,
    title= 'Barplots of Mean Feature Importance',
    hovermode= 'closest',
#    xaxis= dict(
#        title= 'Pop',
#        ticklen= 5,
#        zeroline= False,
#         gridwidth= 2,
#     ),
    yaxis=dict(
        title= 'Feature Importance',
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='bar-direct-labels')
base_predictions_train = pd.DataFrame( {'RandomForest': rf_oof_train.ravel(),
     'ExtraTrees': et_oof_train.ravel(),
     'AdaBoost': ada_oof_train.ravel(),
      'GradientBoost': gb_oof_train.ravel()
    })
base_predictions_train.head()
data = [
    go.Heatmap(
        z= base_predictions_train.astype(float).corr().values ,
        x=base_predictions_train.columns.values,
        y= base_predictions_train.columns.values,
          colorscale='Viridis',
            showscale=True,
            reversescale = True
    )
]
py.iplot(data, filename='labelled-heatmap')
x_train = np.concatenate(( et_oof_train, rf_oof_train, ada_oof_train, gb_oof_train,
svc_oof_train), axis=1)
x_test = np.concatenate(( et_oof_test, rf_oof_test, ada_oof_test, gb_oof_test,
svc_oof_test), axis=1)
gbm = xgb.XGBClassifier(
    #learning_rate = 0.02,
 n_estimators= 2000,
 max_depth= 4,
 min_child_weight= 2,
 #gamma=1,
 gamma=0.9,
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread= -1,
 scale_pos_weight=1).fit(x_train, y_train)
predictions = gbm.predict(x_test)
# Generate Submission File
StackingSubmission = pd.DataFrame({ 'PassengerId': PassengerId,
                            'Survived': predictions })
StackingSubmission.to_csv("StackingSubmission.csv", index=False)
```



PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47	1	0	363272	7		S

Conclusion

Advanced AI algorithms can be used to improve predictive accuracy in a wide variety of applications. By choosing the right algorithm, preparing your data carefully, and evaluating and deploying the model correctly, you can gain valuable insights from your data and make better decisions.