

Out[74]:

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	0	0	1	0	3
1	1	1	1	2	3	1	3	0	2
2	1	3	1	1	1	0	2	1	3
3	1	1	1	2	3	0	3	0	2
4	0	3	0	2	1	0	1	1	6
5	0	3	0	1	1	2	1	1	3
6	0	1	0	3	3	0	1	1	3
7	0	3	0	0	2	0	4	0	0
8	1	3	1	1	1	0	3	0	3
9	1	2	1	0	2	1	3	0	0

And the test dataset.

In [76]: `test_df.head(10)`

Out[76]:

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	0	2	1	1	6
1	893	3	1	2	0	0	3	0	6
2	894	2	0	3	1	2	1	1	6
3	895	3	0	1	1	0	1	1	3
4	896	3	1	1	1	0	3	0	3
5	897	3	0	0	1	0	1	1	0
6	898	3	1	1	0	2	2	1	3
7	899	2	0	1	2	0	1	0	2
8	900	3	1	1	0	1	3	1	3
9	901	3	0	1	2	0	1	0	3

Model, predict and solve

Now we are ready to train a model and predict the required solution. There are 60+ predictive modelling algorithms to choose from. We must understand the type of problem and solution requirement to narrow down to a select few models

which we can evaluate. Our problem is a classification and regression problem. We want to identify relationship between output (Survived or not) with other variables or features (Gender, Age, Port...). We are also performing a category of machine learning which is called supervised learning as we are training our model with a given dataset. With these two criteria - Supervised Learning plus Classification and Regression, we can narrow down our choice of models to a few. These include:

- Logistic Regression
- KNN or k-Nearest Neighbors
- Support Vector Machines
- Naive Bayes classifier
- Decision Tree
- Random Forrest
- Perceptron
- Artificial neural network
- RVM or Relevance Vector Machine

```
In [78]: X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.drop("PassengerId", axis=1).copy()
X_train.shape, Y_train.shape, X_test.shape
```

```
Out[78]: ((891, 8), (891,), (418, 8))
```

Logistic Regression is a useful model to run early in the workflow. Logistic regression measures the relationship between the categorical dependent variable (feature) and one or more independent variables (features) by estimating probabilities using a logistic function, which is the cumulative logistic distribution. Reference [Wikipedia](#).

Note the confidence score generated by the model based on our training dataset.

```
In [80]: # Logistic Regression

logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
acc_log
```

```
Out[80]: 80.36
```

We can use Logistic Regression to validate our assumptions and decisions for feature creating and completing goals. This can be done by calculating the

coefficient of the features in the decision function.

Positive coefficients increase the log-odds of the response (and thus increase the probability), and negative coefficients decrease the log-odds of the response (and thus decrease the probability).

- Sex is highest positive coefficient, implying as the Sex value increases (male: 0 to female: 1), the probability of Survived=1 increases the most.
- Inversely as Pclass increases, probability of Survived=1 decreases the most.
- This way Age*Class is a good artificial feature to model as it has second highest negative correlation with Survived.
- So is Title as second highest positive correlation.

```
In [82]: coeff_df = pd.DataFrame(train_df.columns.delete(0))
coeff_df.columns = ['Feature']
coeff_df["Correlation"] = pd.Series(logreg.coef_[0])

coeff_df.sort_values(by='Correlation', ascending=False)
```

```
Out [82]:
```

	Feature	Correlation
1	Sex	2.201445
5	Title	0.397484
2	Age	0.286911
4	Embarked	0.261583
6	IsAlone	0.126942
3	Fare	-0.086368
7	Age*Class	-0.310963
0	Pclass	-0.750392

Next we model using Support Vector Machines which are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training samples, each marked as belonging to one or the other of **two categories**, an SVM training algorithm builds a model that assigns new test samples to one category or the other, making it a non-probabilistic binary linear classifier. Reference [Wikipedia](#).

Note that the model generates a confidence score which is higher than Logistics Regression model.

```
In [84]: # Support Vector Machines
```

```
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
acc_svc
```

Out [84]: 78.23

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. A sample is classified by a majority vote of its neighbors, with the sample being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. Reference [Wikipedia](#).

KNN confidence score is better than Logistics Regression but worse than SVM.

```
In [86]: knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
acc_knn
```

Out [86]: 84.85

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features) in a learning problem. Reference [Wikipedia](#).

The model generated confidence score is the lowest among the models evaluated so far.

```
In [88]: # Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
acc_gaussian
```

Out [88]: 72.28

The perceptron is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers,

belongs to some specific class or not). It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time. Reference [Wikipedia](#).

```
In [90]: # Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
Y_pred = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
acc_perceptron
```

Out[90]: 78.34

```
In [91]: # Linear SVC

linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)
Y_pred = linear_svc.predict(X_test)
acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
acc_linear_svc
```

Out[91]: 78.9

```
In [92]: # Stochastic Gradient Descent

sgd = SGDClassifier()
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
acc_sgd
```

Out[92]: 71.94

This model uses a decision tree as a predictive model which maps features (tree branches) to conclusions about the target value (tree leaves). Tree models where the target variable can take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Reference [Wikipedia](#).

The model confidence score is the highest among models evaluated so far.

```
In [94]: # Decision Tree
```

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100,
acc_decision_tree
```

Out [94]: 86.76

The next model Random Forests is one of the most popular. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees ($n_{\text{estimators}}=100$) at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Reference [Wikipedia](#).

The model confidence score is the highest among models evaluated so far. We decide to use this model's output (Y_{pred}) for creating our competition submission of results.

In [96]: *# Random Forest*

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100,
acc_random_forest
```

Out [96]: 86.76

Model evaluation

We can now rank our evaluation of all the models to choose the best one for our problem. While both Decision Tree and Random Forest score the same, we choose to use Random Forest as they correct for decision trees' habit of overfitting to their training set.

```
In [98]: models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
    'Random Forest', 'Naive Bayes', 'Perceptron',
    'Stochastic Gradient Decent', 'Linear SVC',
    'Decision Tree'],
    'Score': [acc_svc, acc_knn, acc_log,
    acc_random_forest, acc_gaussian, acc_perceptron,
    acc_sgd, acc_linear_svc, acc_decision_tree]})
models.sort_values(by='Score', ascending=False)
```

Out [98]:

	Model	Score
3	Random Forest	86.76
8	Decision Tree	86.76
1	KNN	84.85
2	Logistic Regression	80.36
7	Linear SVC	78.90
5	Perceptron	78.34
0	Support Vector Machines	78.23
4	Naive Bayes	72.28
6	Stochastic Gradient Decent	71.94

Our submission to the competition site Kaggle results in scoring 3,883 of 6,082 competition entries. This result is indicative while the competition is running. This result only accounts for part of the submission dataset. Not bad for our first attempt. Any suggestions to improve our score are most welcome.

References

This notebook has been created based on great work done solving the Titanic competition and other sources.

- [A journey through Titanic](#)
- [Getting Started with Pandas: Kaggle's Titanic Competition](#)
- [Titanic Best Working Classifier](#)

Improved

In [172]...

```

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

# SVM
svc_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='rbf', random_state=0))
])
param_grid_svc = {'svc__C': [0.5, 1, 10], 'svc__gamma': ['scale', 'auto']}
grid_svc = GridSearchCV(svc_pipe, param_grid_svc, cv=5)
grid_svc.fit(X_train, Y_train)

```

```
acc_svc = round(grid_svc.best_score_ * 100, 2)

# SGD (with scaling + tuned loss)
sgd_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('sgd', SGDClassifier(random_state=0, max_iter=1000, tol=1e-3))
])
param_grid_sgd = {'sgd__loss': ['hinge', 'log_loss'], 'sgd__alpha': [0.001, 0.01, 0.1, 1]}
grid_sgd = GridSearchCV(sgd_pipe, param_grid_sgd, cv=5)
grid_sgd.fit(X_train, Y_train)
acc_sgd = round(grid_sgd.best_score_ * 100, 2)

results = {
    "SVM": acc_svc,
    "SGD": acc_sgd,
}
results
```

Out[172... {'SVM': 81.48, 'SGD': 79.01}