

# How good is your model?

SUPERVISED LEARNING WITH SCIKIT-LEARN



**George Boorman**

Core Curriculum Manager, DataCamp

# Classification metrics

- Measuring model performance with accuracy:
  - Fraction of correctly classified samples
  - Not always a useful metric

# Class imbalance

- Classification for predicting fraudulent bank transactions
  - 99% of transactions are legitimate; 1% are fraudulent
- Could build a classifier that predicts NONE of the transactions are fraudulent
  - 99% accurate!
  - But terrible at actually predicting fraudulent transactions
  - Fails at its original purpose
- Class imbalance: Uneven frequency of classes
- Need a different way to assess performance

# Confusion matrix for assessing classification performance

- Confusion matrix

Actual: Fraudulent
Actual: Legitimate

Predicted: Fraudulent	Predicted: Legitimate
--------------------------	--------------------------

True Positive	False Negative
False Positive	True Negative

# Assessing classification performance

Actual: Fraudulent
Actual: Legitimate

Predicted: Fraudulent	Predicted: Legitimate
--------------------------	--------------------------

True Positive	False Negative
False Positive	True Negative

# Assessing classification performance

Predicted: Fraudulent
--------------------------

Predicted: Legitimate
--------------------------

Actual: Fraudulent
--------------------

Actual: Legitimate
--------------------

True Positive
---------------

False Negative
----------------

False Positive
----------------

True Negative
---------------

# Assessing classification performance

Predicted: Fraudulent
--------------------------

Predicted: Legitimate
--------------------------

Actual: Fraudulent
--------------------

Actual: Legitimate
--------------------

True Positive
---------------

False Negative
----------------

False Positive
----------------

True Negative
---------------

# Assessing classification performance

	Predicted: Fraudulent	Predicted: Legitimate
Actual: Fraudulent	<b>True Positive</b>	<b>False Negative</b>
Actual: Legitimate	<b>False Positive</b>	<b>True Negative</b>



# Assessing classification performance

	Predicted: Fraudulent	Predicted: Legitimate
Actual: Fraudulent	True Positive	False Negative
Actual: Legitimate	False Positive	True Negative

# Assessing classification performance

Predicted: Fraudulent
--------------------------

Predicted: Legitimate
--------------------------

Actual: Fraudulent
--------------------

Actual: Legitimate
--------------------

True Positive
---------------

<b>False Negative</b>
-----------------------

False Positive
----------------

True Negative
---------------

# Assessing classification performance

	Predicted: Fraudulent	Predicted: Legitimate
Actual: Fraudulent	True Positive	False Negative
Actual: Legitimate	<b>False Positive</b>	True Negative

# Assessing classification performance

Predicted: Fraudulent	Predicted: Legitimate
--------------------------	--------------------------

Actual: Fraudulent
Actual: Legitimate

True Positive	False Negative
False Positive	True Negative

- Accuracy:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

# Precision

	Predicted: Fraudulent	Predicted: Legitimate
Actual: Fraudulent	True Positive	False Negative
Actual: Legitimate	False Positive	True Negative

- Precision

$$\frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

- High precision = lower false positive rate
- High precision: Not many legitimate transactions are predicted to be fraudulent

# Recall

	Predicted: Fraudulent	Predicted: Legitimate
Actual: Fraudulent	True Positive	False Negative
Actual: Legitimate	False Positive	True Negative

- Recall

$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- High recall = lower false negative rate
- High recall: Predicted most fraudulent transactions correctly

# F1 score

- F1 Score:  $2 * \frac{precision * recall}{precision + recall}$

# Confusion matrix in scikit-learn

```
from sklearn.metrics import classification_report, confusion_matrix
knn = KNeighborsClassifier(n_neighbors=7)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=42)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```



# Confusion matrix in scikit-learn

```
print(confusion_matrix(y_test, y_pred))
```

```
[[1106  11]  
 [ 183  34]]
```

# Classification report in scikit-learn

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.86	0.99	0.92	1117
1	0.76	0.16	0.26	217
accuracy			0.85	1334
macro avg	0.81	0.57	0.59	1334
weighted avg	0.84	0.85	0.81	1334

# Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

# Logistic regression and the ROC curve

SUPERVISED LEARNING WITH SCIKIT-LEARN



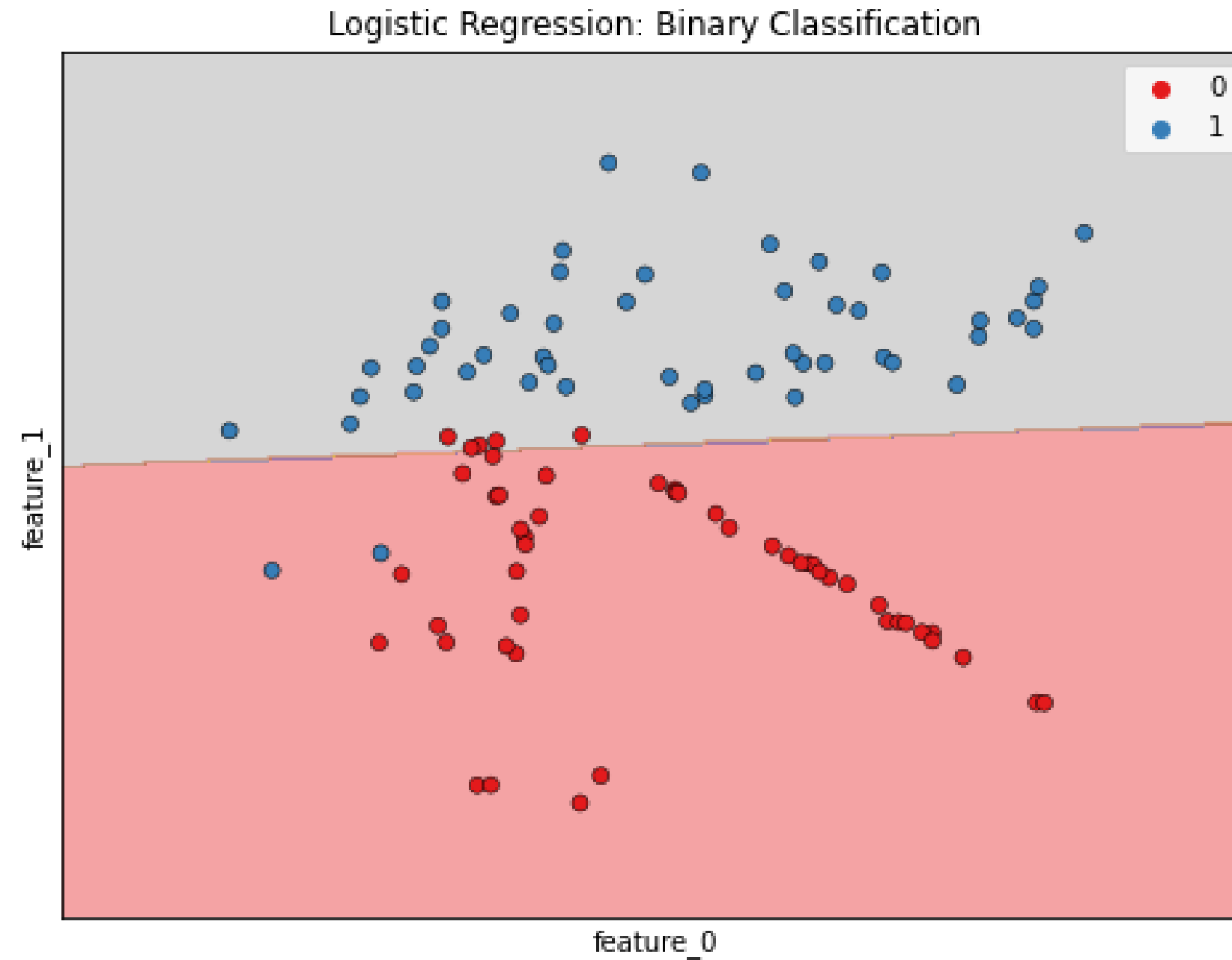
**George Boorman**

Core Curriculum Manager, DataCamp

# Logistic regression for binary classification

- Logistic regression is used for classification problems
- Logistic regression outputs probabilities
- If the probability,  $p > 0.5$ :
  - The data is labeled 1
- If the probability,  $p < 0.5$ :
  - The data is labeled 0

# Linear decision boundary



# Logistic regression in scikit-learn

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)

logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

# Predicting probabilities

```
y_pred_probs = logreg.predict_proba(X_test)[: , 1]  
print(y_pred_probs[0])
```

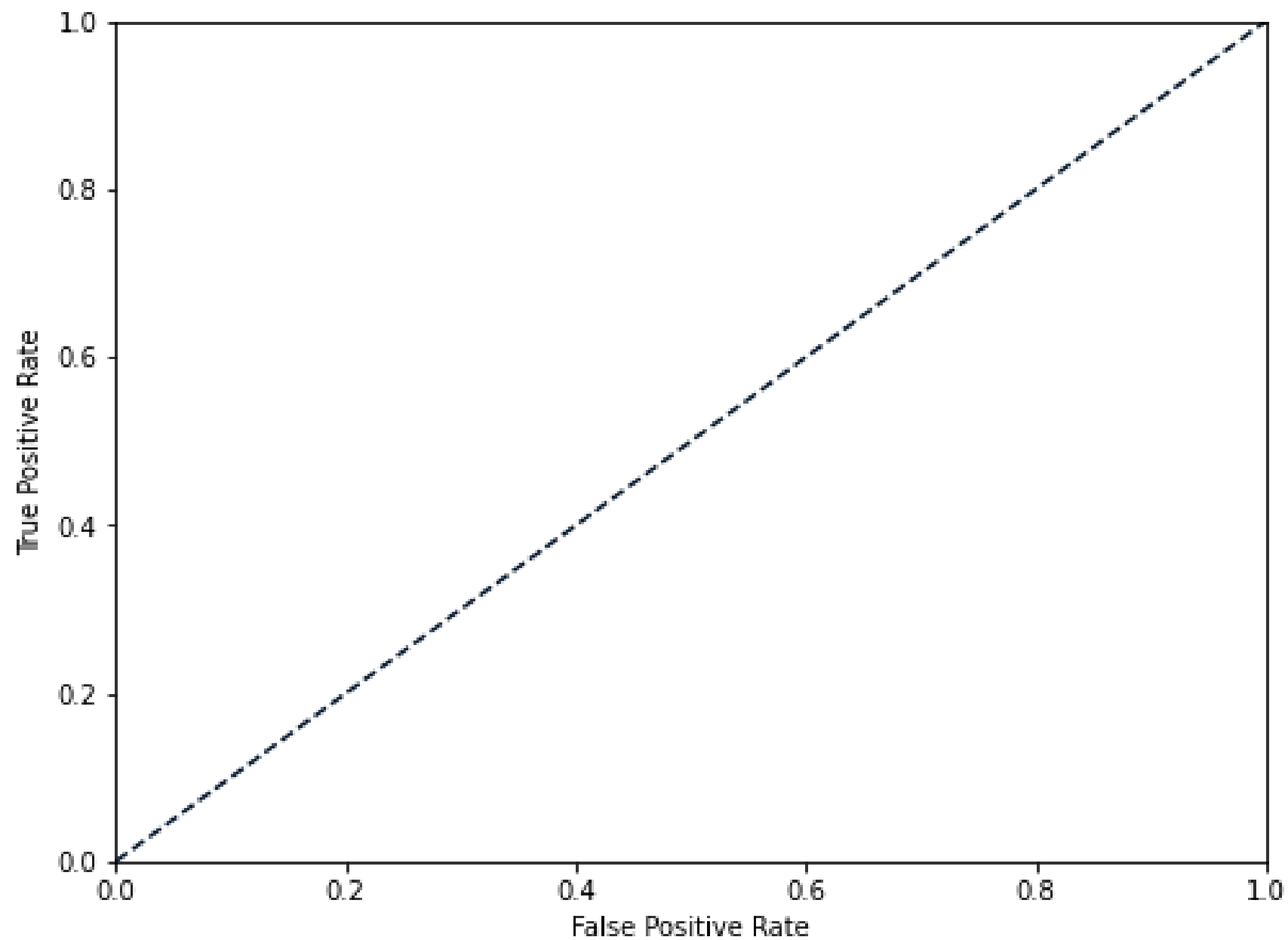
```
[0.08961376]
```



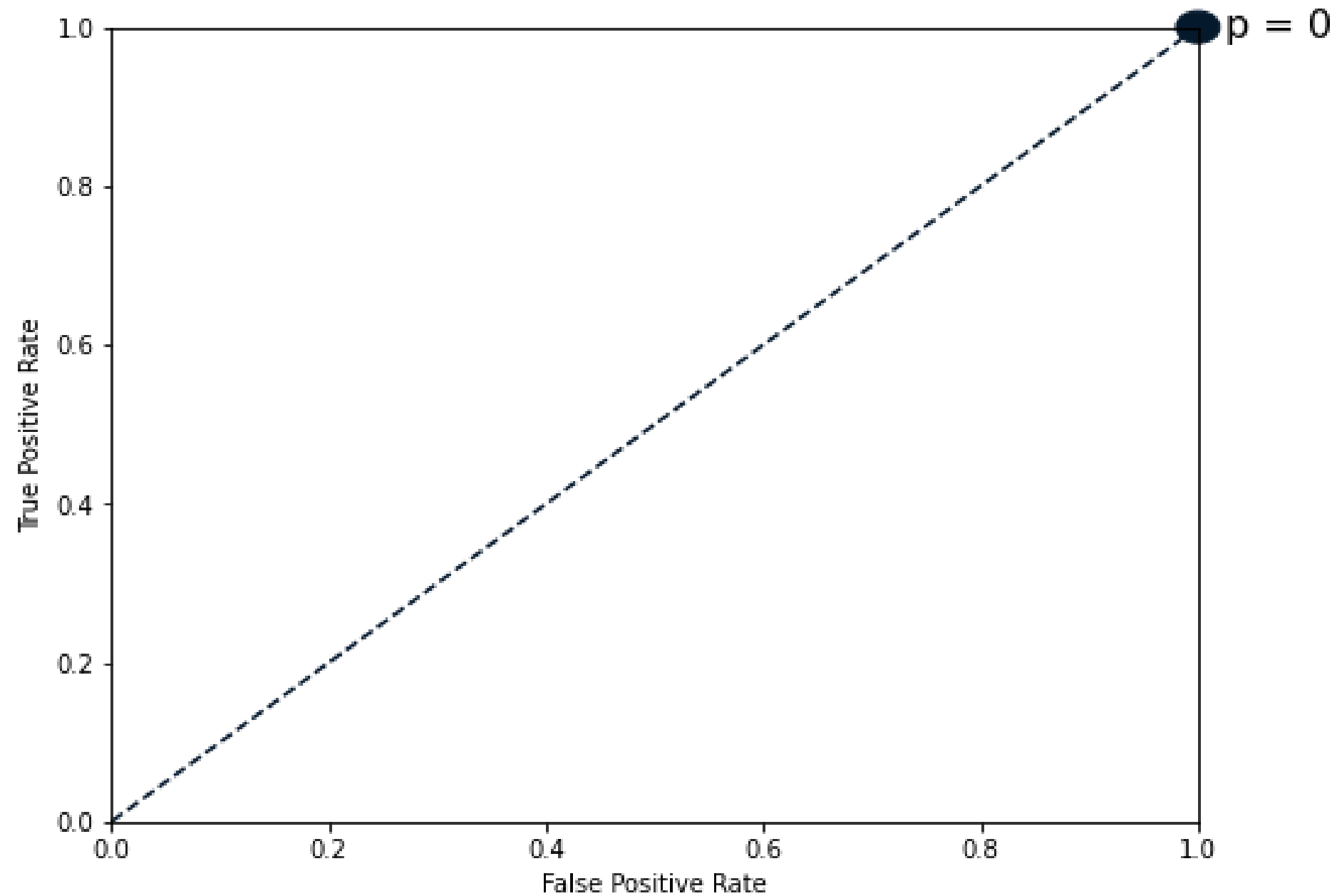
# Probability thresholds

- By default, logistic regression threshold = 0.5
- Not specific to logistic regression
  - KNN classifiers also have thresholds
- What happens if we vary the threshold?

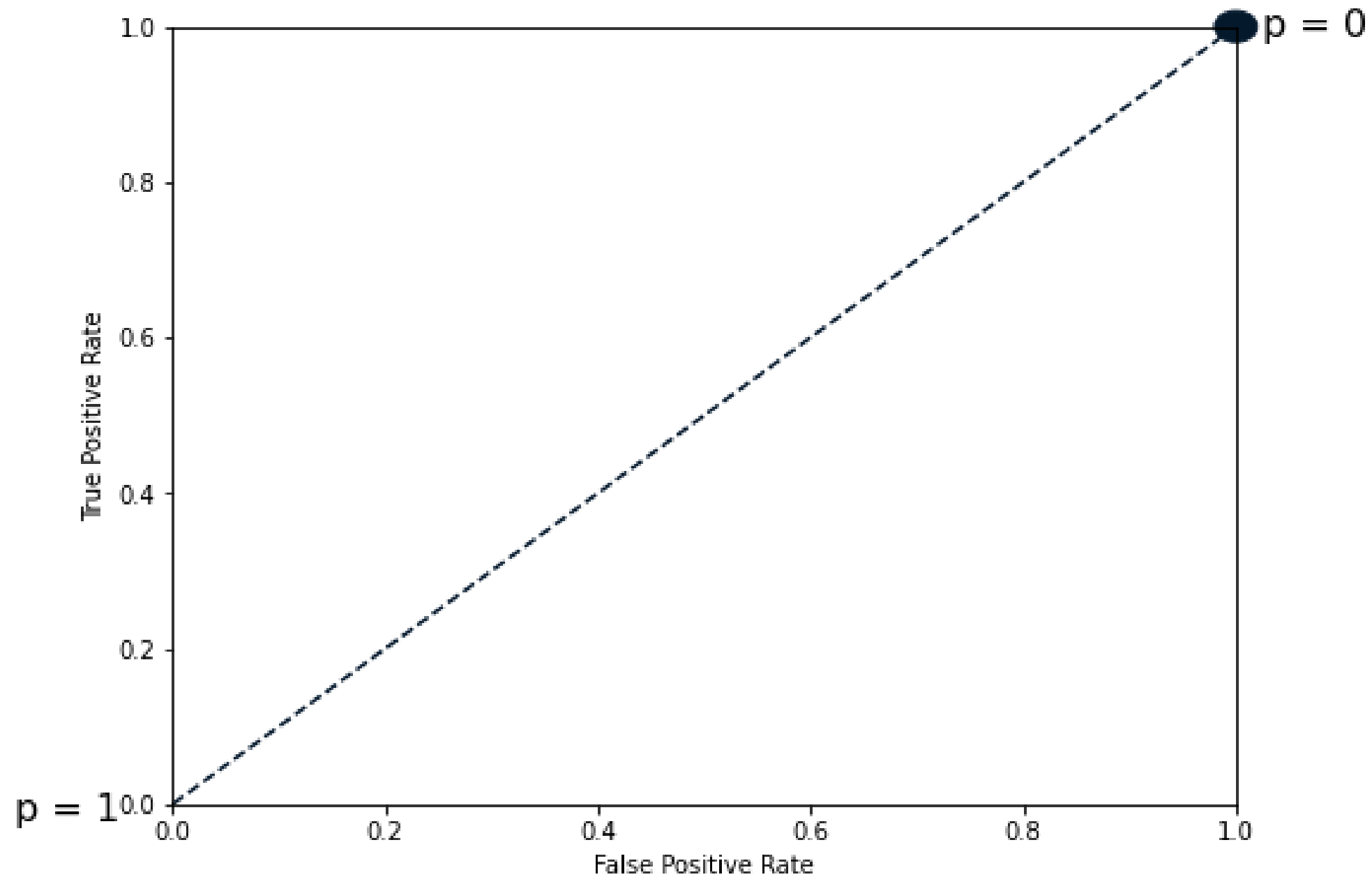
# The ROC curve



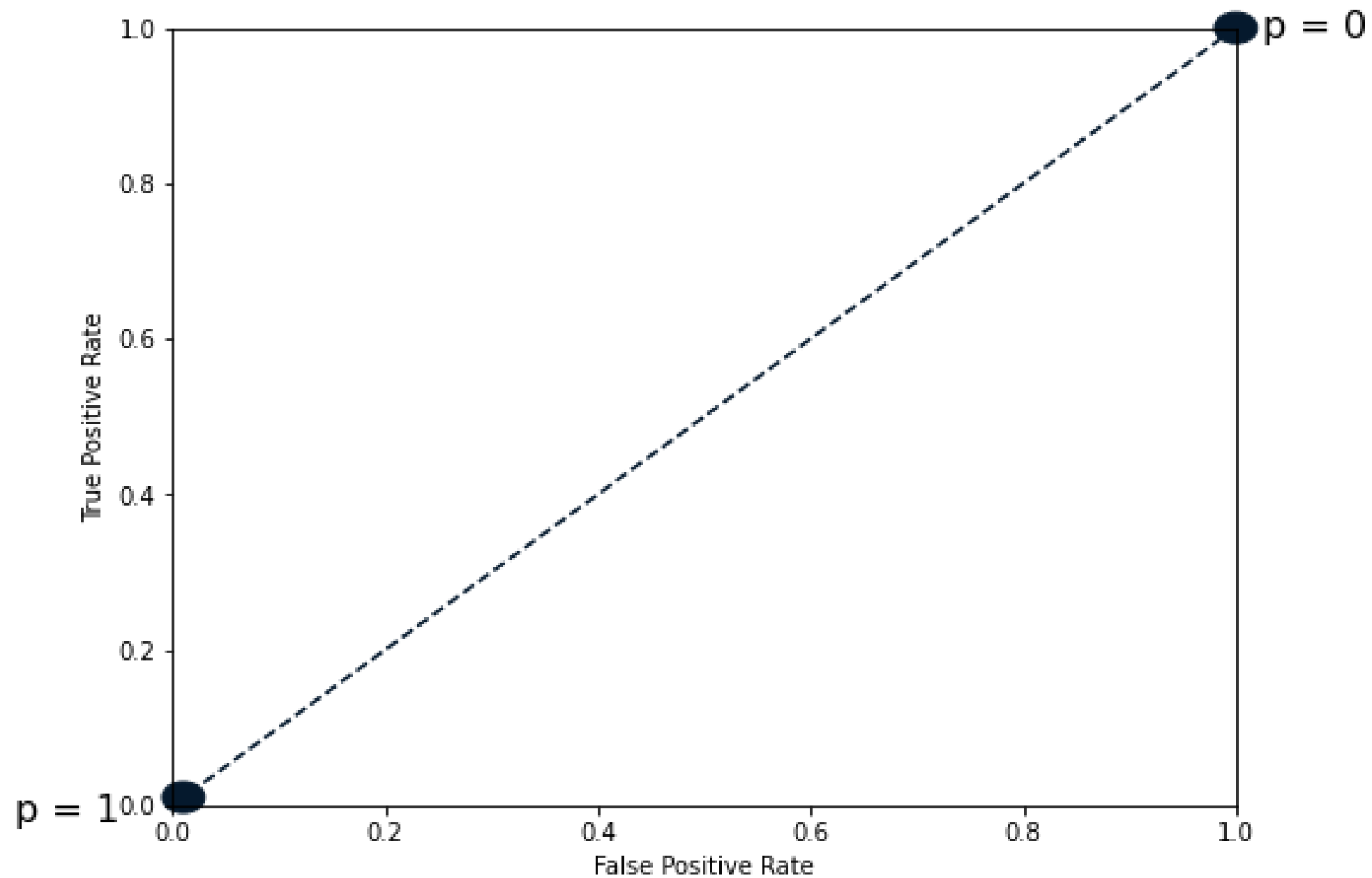
# The ROC curve



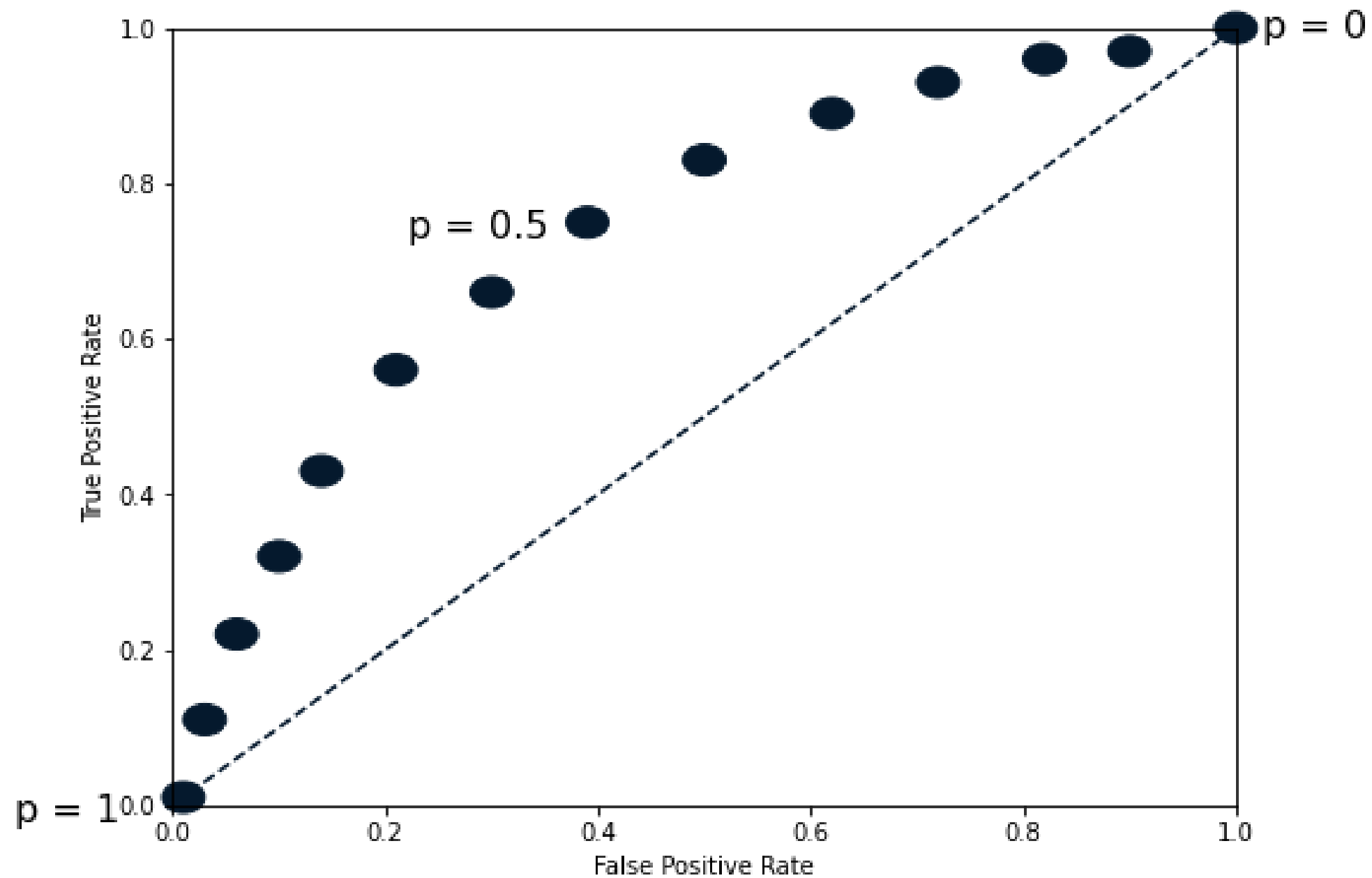
# The ROC curve



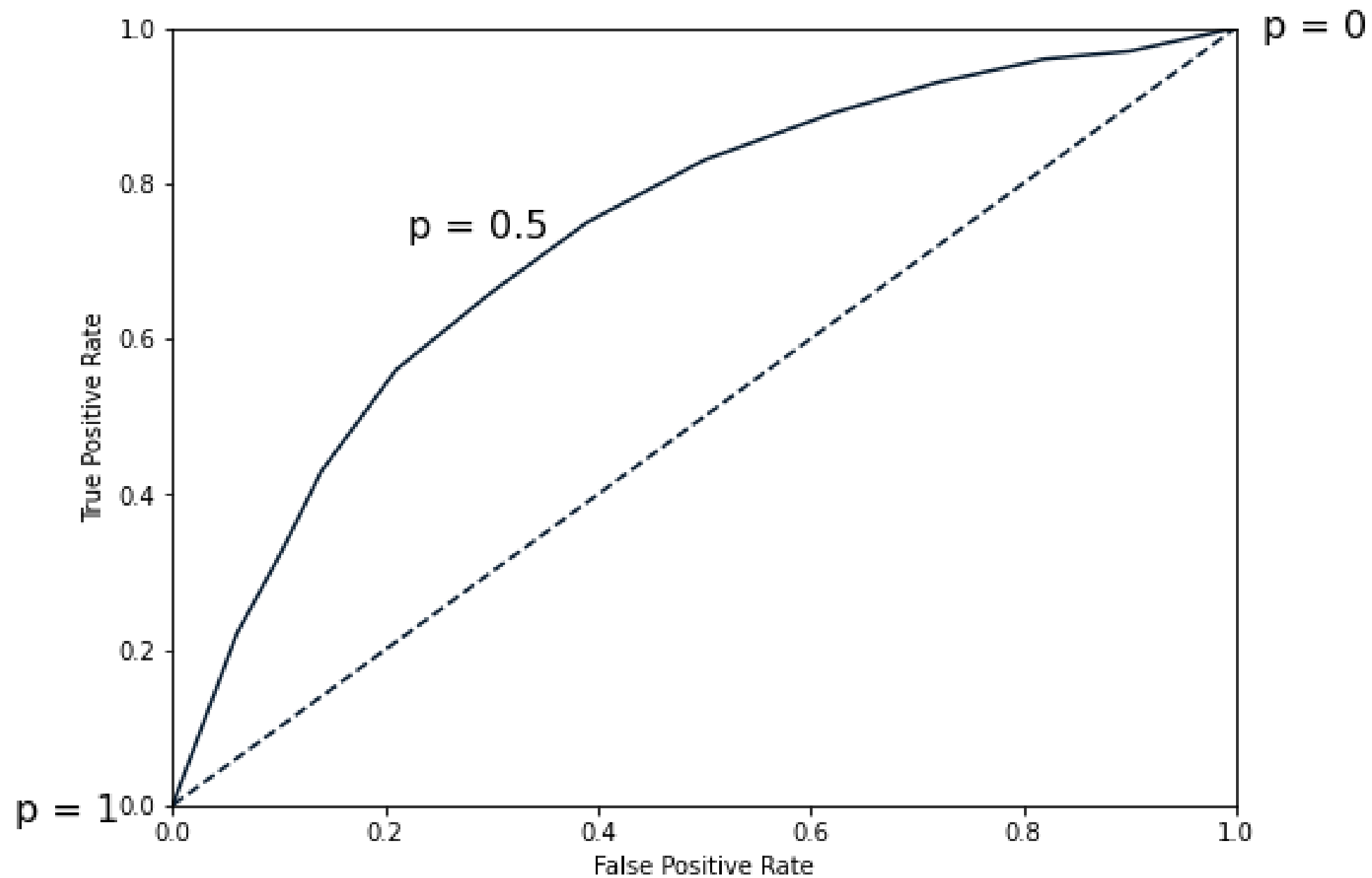
# The ROC curve



# The ROC curve



# The ROC curve

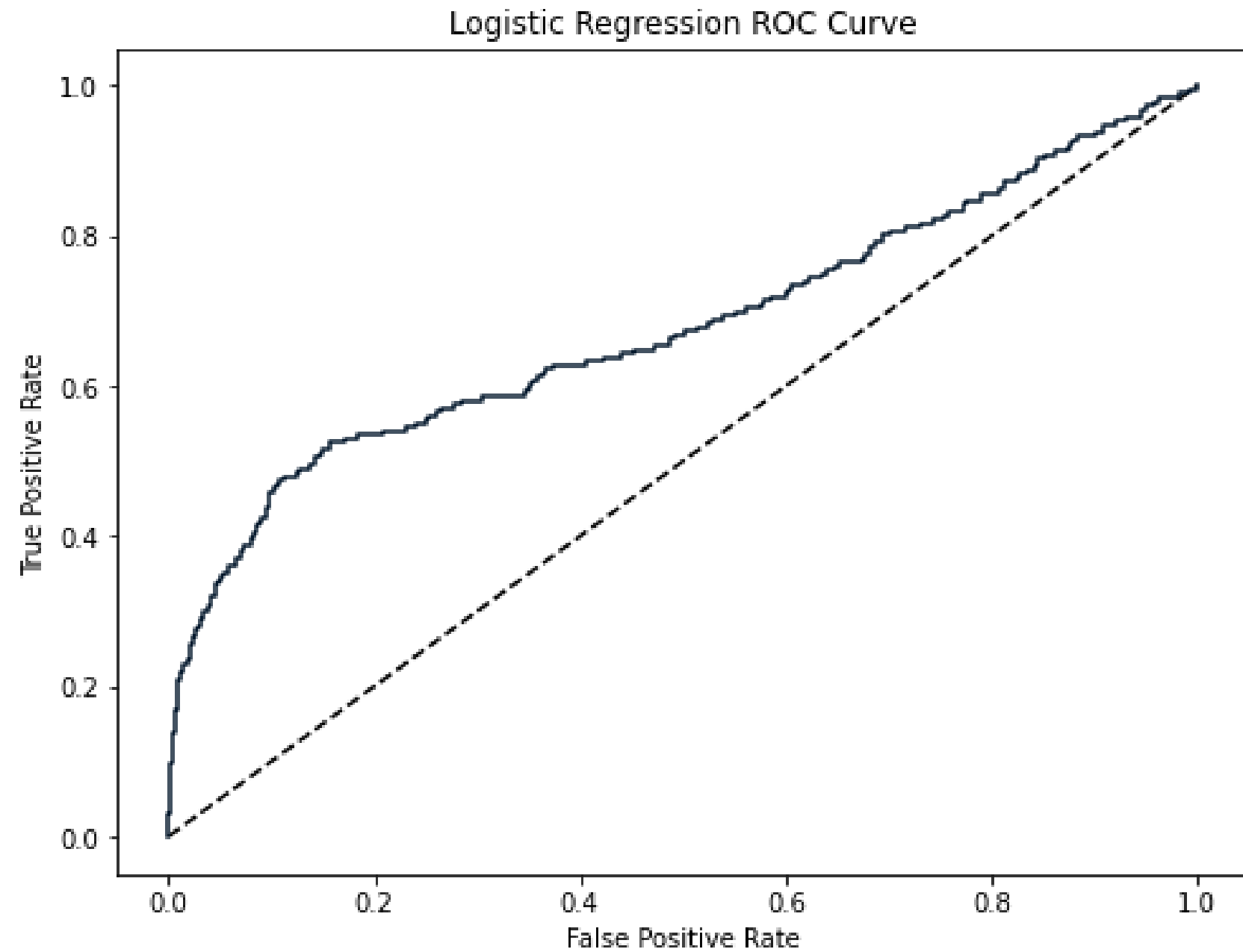


# Plotting the ROC curve

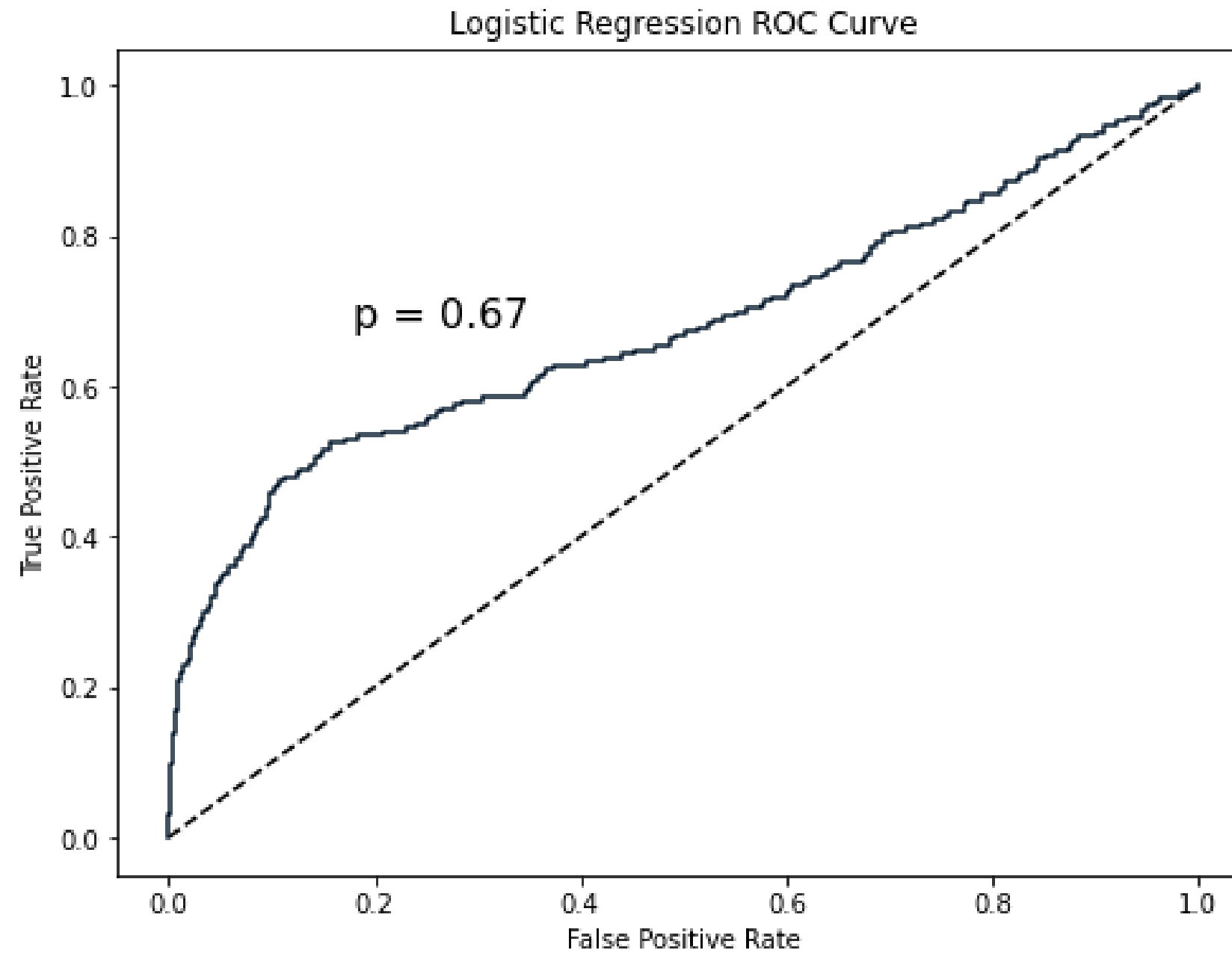
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show()
```



# Plotting the ROC curve



# ROC AUC



# ROC AUC in scikit-learn

```
from sklearn.metrics import roc_auc_score  
print(roc_auc_score(y_test, y_pred_probs))
```

```
0.6700964152663693
```

# Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

# Hyperparameter tuning

SUPERVISED LEARNING WITH SCIKIT-LEARN



**George Boorman**  
Core Curriculum Manager

# Hyperparameter tuning

- Ridge/lasso regression: Choosing `alpha`
- KNN: Choosing `n_neighbors`
- Hyperparameters: Parameters we specify before fitting the model
  - Like `alpha` and `n_neighbors`

# Choosing the correct hyperparameters

1. Try lots of different hyperparameter values
  2. Fit all of them separately
  3. See how well they perform
  4. Choose the best performing values
- This is called **hyperparameter tuning**
  - It is essential to use cross-validation to avoid overfitting to the test set
  - We can still split the data and perform cross-validation on the training set
  - We withhold the test set for final evaluation

# Grid search cross-validation

<b>n_neighbors</b>	11		
	8		
	5		
	2		
		euclidean	manhattan
		<b>metric</b>	



# Grid search cross-validation

<b>n_neighbors</b>	11	0.8716	0.8692
	8	0.8704	0.8688
	5	0.8748	0.8714
	2	0.8634	0.8646
		euclidean	manhattan
		<b>metric</b>	

# Grid search cross-validation

n_neighbors	11	0.8716	0.8692
	8	0.8704	0.8688
	5	0.8748	0.8714
	2	0.8634	0.8646
		euclidean	manhattan
		metric	

# GridSearchCV in scikit-learn

```
from sklearn.model_selection import GridSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {"alpha": np.arange(0.0001, 1, 10),
              "solver": ["sag", "lsqr"]}
ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf)
ridge_cv.fit(X_train, y_train)
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

```
{'alpha': 0.0001, 'solver': 'sag'}
0.7529912278705785
```

# Limitations and an alternative approach

- 3-fold cross-validation, 1 hyperparameter, 10 total values = 30 fits
- 10 fold cross-validation, 3 hyperparameters, 30 total values = 900 fits

# RandomizedSearchCV

```
from sklearn.model_selection import RandomizedSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {'alpha': np.arange(0.0001, 1, 10),
              "solver": ['sag', 'lsqr']}
ridge = Ridge()
ridge_cv = RandomizedSearchCV(ridge, param_grid, cv=kf, n_iter=2)
ridge_cv.fit(X_train, y_train)
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

```
{'solver': 'sag', 'alpha': 0.0001}
0.7529912278705785
```

# Evaluating on the test set

```
test_score = ridge_cv.score(X_test, y_test)  
print(test_score)
```

```
0.7564731534089224
```

# Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN